International Language Environments Guide

Part No: 817–2521–11
November 2010

101210@25097

Contents

Figures

Tables

Preface

The *International Language Environments Guide* introduces the internationalization features that are new to the Oracle Solaris operating system (Oracle Solaris OS). The guide contains information on how to use the current Oracle Solaris release to build global software products that support a variety of languages and cultural conventions.

In addition, the guide provides pointers to other documentation that includes further information on the internationalization features in this release.

Note – All of the information related to operating systems in the guide pertains to the Oracle Solaris operating system.

This preface includes the following sections.

Note – This Oracle Solaris release supports systems that use the SPARC and x86 families of processor architectures: UltraSPARC, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems appear in the *Solaris 10 Hardware Compatibility List*. This document cites any implementation differences between the platform types.

In this document the term "x86" refers to 64-bit and 32-bit systems manufactured using processors compatible with the AMD64 or Intel Xeon/Pentium product families. For supported systems, see the *Oracle Solaris Hardware Compatibility List*.

About This Book

This guide written for software developers and system administrators who design and support global applications in the current Oracle Solaris operating system.

The guide assumes that you have a working knowledge of the C programming language.

How This Guide Is Organized

The chapters in this guide are organized as follows:

- Chapter 1, "Oracle Solaris Internationalization Overview," describes the new internationalization and localization features that are available in the current Oracle Solaris release.
- Chapter 2, "General Internationalization Features," provides introductory information on Code Set Independence (CSI), the locale database, the `libc` APIs, and other internationalization features.
- Chapter 3, "Localization in the Oracle Solaris Environment," provides information on the locales, fonts, and keyboards that are supported for use in the current Oracle Solaris operating system.
- Chapter 4, "Supported Asian Locales," describes the Japanese, Hindi, and Thai localization support that is offered in the current Oracle Solaris release.
- Chapter 5, "Overview of `UTF-8` Locale Support," provides information on the available input methods and code conversion functionality supported for use in the current Oracle Solaris operating system.
- Chapter 6, "Complex Text Layout," describes the Complex Text Layout (CTL) extensions that enable Motif APIs to support writing systems that require complex transformations between logical and physical text representations. Writing systems that require complex transformations include Arabic, Hebrew, and Thai.
- Chapter 7, "Print Filter Enhancement With `mp`," explains printing support with particular emphasis on the `mp` print filter.
- Appendix A, "Compose and Dead Key Input," describes the commonly used compose key sequences in the different input modes and key support.
- Appendix B, "Language Support Features and Enhancements," describes the language support enhancements introduced in Solaris with different versions.

Related Books and Sites

The following books offer further information on the topics discussed in this guide:

- Oracle Solaris internationalization:

  Tuthill, Bill, and David Smallberg. *Creating Worldwide Software: Solaris International Developer's Guide*, 2nd edition. Mountain View, California, Sun Microsystems Press, 1997. This book is available through *http://mailto:books@sun.com* and *http://www.sun.com/books/*. The book offers a general overview of the internationalization process in the Oracle Solaris operating system.
- Oracle Solaris Common Desktop Environment:

  The *Oracle Solaris Common Desktop Environment: Programmer's Guide* is part of the CDE Developer's Collection that is shipped on the Oracle Solaris documentation CD.
- Chinese and Korean Solaris locales:

  Korean Solaris User's Guide
  Simplified Chinese Solaris User's Guide
  Traditional Chinese Solaris User's Guide
- OSF/Motif application development:

  The *OSF/Motif Programmer's Guide, Release 1.2*, Englewood Cliffs, New Jersey, Prentice-Hall, 1993. This book is the Open Software Foundations (OSF) guide on how to use the OSF/Motif application programming interface to create Motif applications.

Documentation, Support, and Training

See the following web sites for additional resources:

- Documentation (`http://docs.sun.com`)
- Support (`http://www.oracle.com/us/support/systems/index.html`)
- Training (`http://education.oracle.com`) – Click the Sun link in the left navigation bar.

Oracle Software Resources

Oracle Technology Network (`http://www.oracle.com/technetwork/index.html`) offers a range of resources related to Oracle software:

- Discuss technical problems and solutions on the Discussion Forums (`http://forums.oracle.com`).
- Get hands-on step-by-step tutorials with Oracle By Example (`http://www.oracle.com/technetwork/tutorials/index.html`).
- Download Sample Code (`http://www.oracle.com/technology/sample_code/index.html`).

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P–2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	`$`
Bash shell, Korn shell, and Bourne shell for superuser	`#`
C shell	`machine_name%`
C shell for superuser	`machine_name#`

CHAPTER 1

Oracle Solaris Internationalization Overview

This chapter introduces the new features and the key concepts of Oracle Solaris internationalization and localization. The chapter covers the following topics.

- "New Internationalization and Localization Features" on page 18
- "Internationalization and Localization Overview" on page 20
- "What Is a Locale?" on page 22
- "Using Locale Categories for Localization" on page 25
- "Language Word and Letter Differences" on page 30
- "Keyboard Differences" on page 33
- "Differences in Paper Sizes" on page 33

Oracle Solaris Internationalization Architecture

The current Oracle Solaris release includes a number of new features, including Unicode 4.0 support for the UTF-8 locales, enhanced keyboard support, and several improvements to the `mp` print filter.

The Oracle Solaris internationalization architecture eases the development, the deployment, and the management of applications and language services around the world. A single multilingual product provides support for 55 different languages and 345 locales. In addition, support is available for the complex text layout that is required for Thai and Hindi scripts. Bidirectional text capability is also supported for languages such as Arabic and Hebrew.

Input methods, character sets, codeset conversion, and other language-related features are supported for a number of different Oracle Solaris locales. You can deploy applications in multiple language environments by following standard APIs. You can also customize language attributes, change converter tables, or add a new input method editor in the Oracle Solaris environment.

The Oracle Solaris 10 globalization framework enables you to follow a common reference implementation to enhance the compatibility and the interoperability of global applications.

The codeset independent approach to globalization enables you to operate in both native language and Unicode locales. The Oracle Solaris framework provides the power to scale across platforms. A rich set of data converters ensures interoperability between various encodings and different third-party platforms.

The Oracle Solaris platform also enables multinational corporations to scale their server administration worldwide. Unlike competing platforms, the Oracle Solaris platform uses a service-based approach to administration of language services. Server administrators can enable language services remotely across a worldwide network, regardless of the client system. This client-independent approach enables system upgrades without changing client applications. For example, a user does not have to change a local client application in order to read email in Arabic sent from an Internet cafe in Paris.

New Internationalization and Localization Features

The following new features are available in the current Oracle Solaris release. More information about each feature can be found at Appendix B, "Language Support Features and Enhancements."

- Unicode Version 3.2 and 4.0 Support

  Unicode Version 4.0 introduces 1226 new characters over Unicode Version 3.2. This version also includes both normative changes and informative changes as described in Unicode Standard 4.0 (ISBN 0-321-18578-1).

  Unicode 3.2 defines more strict UTF-8 byte sequences as "UTF-8 Corrigendum".

Code Points	1st Byte	2nd Byte	3rd Byte	4th Byte
U+0000..U+007F	00..7F			
U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	A0..BF	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80..9F	80..BF	
U+D800..U+DFFF	ill-formed			
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	90..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..8F	80..BF	80..BF

These sequences exclude the surrogate code points between U+D800 and U+DFFF. The sequences also inhibit any other illegal byte values. To comply with the new definition, Unicode locale methods and the UTF-8 iconv modules are enhanced to detect the newly defined UTF-8 invalid byte sequences. For more informations, see "Unicode Version 4.0 Support" on page 198.

- Auto encoding finder

  The auto encoding finder is a utility for global character handling. Through a general-purpose interface, the auto encoding finder provides an easy way to detect the encoding of a particular file or string. Encoding detection simplifies access to various language character encodings. For example, the utility simplifies the display of web pages that do not specify encoding information. Search engines, knowledge databases, and machine translation tools might also need to detect the encoding of the language data being accessed. The Auto Encoding Finder tool simplifies this process.

  For more information, see the `auto_ef(1)` or `libauto_ef(3LIB)` man pages.

- Locale administrator

  The locale administrator enables you to query and configure the locales for a Oracle Solaris operating system through a command-line interface. Using the `localeadm(1M)` tool, you can display information about locale packages that are installed on the system or that reside on a particular device or directory. You can add and remove locales on the current system on a per-region basis. For more information see "Software Support for Localization" on page 53.

- Locale Creator

  Locale Creator is a command line and graphical user interface tool that enables users to create and install Oracle Solaris locales. Using Locale Creator users can create installable Oracle Solaris packages containing customized locale data of a specific locale. After the created package has been installed, the user has a fully-working locale on the system. For more information, see the following:

  - `localectr` command at `/usr/bin/localectr -h`
  - `localectr(1M)` man page
  - Oracle Solaris Locale Creator

- `iconv` Code Conversions

  Various new `iconv` code conversions between single-byte PC and Windows code pages and various Unicode forms have been added. For more information, see the `iconv_en_US.UTF-8(5)` man page.

- Oracle Solaris Unicode Locales

  New Unicode locales are added to Oracle Solaris. The new locales are available on system login. In addition, all EMEA, Central and South American locales have been migrated to Common Locale Data Repository (CLDR). For details, see "Supported Locales" on page 55. For information on CLDR, see Common Locale Data Repository (CLDR)

- Input Method Support

New Internet Intranet Input Method Framework (IIIMF), new Language Engines and EMEA Keyboard Layout Emulation Support has been added. For more information, see: "IIIMF and Language Engines " on page 195 and Appendix B, "Language Support Features and Enhancements." For more information, see "Input Method Features" on page 193.

- Keyboard Layouts Support

  New keyboard layouts have been integrated into current version of Oracle Solaris. For more information see "Keyboard Support in the Oracle Solaris Environment" on page 65.
- `setxkbmap`

  A new feature for switching keyboard layouts has been integrated into Oracle Solaris and is available for the Xorg Server. `setxkbmap` enables switching the keyboard layout simultaneously when using Xorg Server. This command maps the keyboard using the layout determined by various options specified on the command line. For information, see the `setxkbmap` man pages.

Internationalization and Localization Overview

Internationalization and localization are different procedures. *Internationalization* is the process of making software portable between languages or regions, while *localization* is the process of adapting software for specific languages or regions. Internationalized software can be developed using interfaces that modify program behavior at runtime in accordance with specific cultural requirements. Localization involves establishing online information to support a language or region, called a *locale*.

Unlike software that must be completely rewritten before it can work with different native languages and customs, internationalized software does not require rewriting. The internationalized software can be ported from one locale to another without change. The Oracle Solaris system is internationalized, providing the infrastructure and interfaces you need to create internationalized software.

Basic Steps in Internationalization

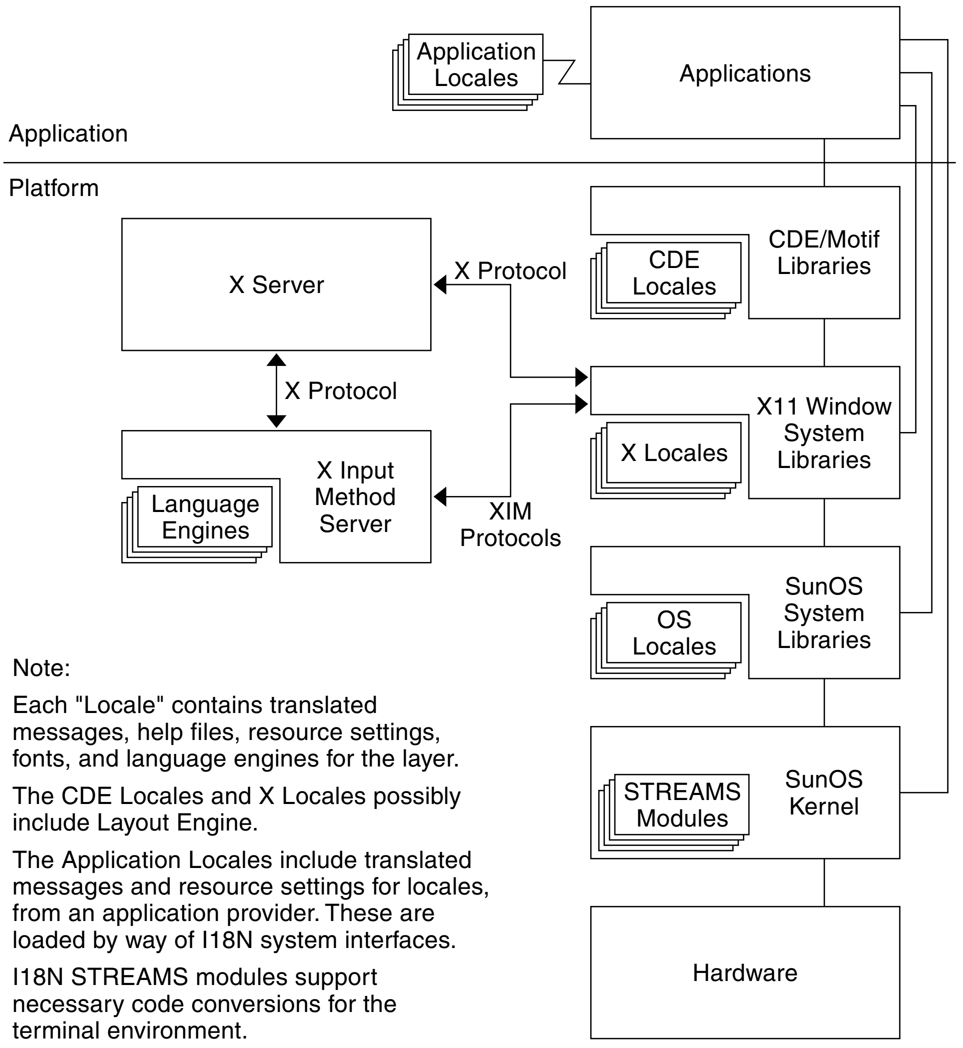
An internationalized application's executable image is portable between languages and regions. To internationalize software:

- Use the interfaces described in this book to create software with an environment that can be modified by dynamically recompiling.
- Divide software into executable code and all the messages that the user might see. Keep the message strings in a message catalog.

Message strings are translated for a language or region. A *locale* includes the message strings and methods to specify sorting.

To use a localized version of a product, the user sets certain environment variables. The product then displays messages that are translated into the language of the locale. Date, time, currency, and other information is formatted and displayed according to locale-specific conventions. Message translations and online help contents are provided throughout different layers, as illustrated in the following diagram.

FIGURE 1–1 Functions and Structure of Locales in the Oracle Solaris operating system



Note:

Each "Locale" contains translated messages, help files, resource settings, fonts, and language engines for the layer.

The CDE Locales and X Locales possibly include Layout Engine.

The Application Locales include translated messages and resource settings for locales, from an application provider. These are loaded by way of I18N system interfaces.

I18N STREAMS modules support necessary code conversions for the terminal environment.

Localization Functions in Oracle Solaris Interfaces

The OS (operating system) locale layer provides the basic locale database and functions that are plugged into the OS system interface at the application's runtime. Applications access these OS locale modules through standard APIs.

The X11 locale layer provides the interface to the X input method and X output method to X11 applications for local text input and display. Fonts enable applications to display characters from various languages.

CDE/Motif is built on top of the X11 window system. Hence, CDE/Motif can utilize the X11 locale capability through X11 APIs. Oracle Solaris localizations have various locale-specific configurations for CDE applications in order to make the desktop functional within the target locale. Message translations and online help contents are provided throughout different layers.

What Is a Locale?

A key concept for application programs is that of a program's *locale*. The locale is an explicit model and definition of a native-language environment. The notion of a locale is explicitly defined and included in the library definitions of the ANSI C Language standard.

A locale consists of a number of categories for which country-dependent formatting or other specifications exist. A program's locale defines its code sets, date and time formatting conventions, monetary conventions, decimal formatting conventions, and collation (sort) order.

A locale can be composed of a base language, country (territory) of use, and an optional codeset. Codeset is usually assumed. For example, German is `de`, an abbreviation for Deutsch, while Swiss German is `de_CH`, `CH` being an abbreviation for Confederation Helvetica. This convention allows for specific differences by country, such as currency unit notation.

More than one locale can be associated with a particular language, which allows for regional differences. For example, an English-speaking user in the United States can select the `en_US` locale (English for the United States), while an English-speaking user in Great Britain can select `en_GB` (English for Great Britain).

Generally the locale name is specified by the `LANG` environment variable. Locale categories are subordinate to `LANG` but can be set separately, in which case they override `LANG`. If the `LC_ALL` operand is set, it overrides `LANG` and all the separate locale categories.

The locale naming convention is:

language[*_territory*][*.codeset*] [*@modifier*]

where a two-letter *language* code is from ISO 639, a two-letter *territory* code is from ISO 3166, *codeset* is the name of the codeset that is being used in the locale, and *modifier* is the name of the characteristics that differentiate the locale from the locale without the modifier.

All Oracle Solaris product locales preserve the Portable Character Set characters with US-ASCII code values.

For more information on the portable character set, refer to “X/Open CAE Specification: System Interface Definitions, Issue 5” (ISBN 1–85912–186–1).

A single locale can have more than one locale name. For example, `POSIX` is the same locale as `C`.

C Locale – the Default Locale

The C locale, also known as the POSIX locale, is the POSIX system default locale for all POSIX-compliant systems. The Oracle Solaris operating system is a POSIX system. The Single UNIX Specification, Version 3, defines the C locale. Register to read and download the specification at: `http://www.unix.org/version3/online.html.`

You can specify that your internationalized programs run in the C locale, in one of two ways:

- Unset all locale environment variables.

  ```
  system% unsetenv LC_ALL LANG LC_CTYPE LC_COLLATE LC_NUMERIC \
                      LC_TIME LC_MONETARY LC_MESSAGES
  ```

  Unsets all locale environment variables. Runs the application in the C locale.
- Explicitly set the locale to C or POSIX.

  ```
  system% setenv LC_ALL C
  system% setenv LANG C
  ```

  Some applications check the LANG environment variables without actually calling `setlocale`(3C) to reference the current locale. In this case, `setenv` explicitly sets the C locale by specifying the LC_ALL and LANG locale environment variables. For the precedence relationship among locale environment variables, see the `setlocale`(3C) man page.

To check the current locale settings in a terminal environment, run the `locale`(1) command.

```
system% locale
```

Full and Partial Locales

A full Oracle Solaris locale has all the listed functions and the localized system messages in the relevant language. *Partial locales* have no localized messages installed. All locales in the Oracle Solaris environment are capable of displaying localized messages, provided that localized messages for the relevant language are installed. For example, the following locales can be either partial or full locales:

- `de_DE.ISO8859–1`

- `de_DE.ISO8859–15`
- `de_DE.UTF-8`
- `de_AT.ISO8859–1`
- `de_AT.ISO8859–15`
- `de_CH.ISO8859–1`

When the German message translations are installed from the Oracle Solaris DVD, all of the above locales become *full locales* because they have access to a fully translated desktop. The Oracle Solaris DVD contains message translations for the following languages and locales:

- German
- French
- Spanish
- Brazilian Portuguese
- Italian
- Japanese
- Korean
- Simplified Chinese locale
- Traditional Chinese locale

All partial and full locales as well as message translations are available on the Oracle Solaris DVD.

Behavior Affected by Locales

Different cultures often use different conventions to format numbers, to write the date and time, to delimit words and phrases, or to quote written and spoken material. A locale determines how the following operations, files, formats, and expressions are handled for different regions:

- Encoding and processing of text data
- Language identification and encoding of resource files
- Rendering and layout of text strings
- Interchange of text between clients
- Input method selection to meet the codeset and text processing requirements of the chosen script
- Font and icon files that are culturally specific
- Actions and file types
- User Interface Definition (UID) files
- Date and time formats
- Numeric formats
- Monetary formats

- Collation order
- Regular expression handling specific to the locale
- Format for informative and diagnostic messages and interactive responses

The Oracle Solaris environment separates language and culture-dependent information from the application and saves the information outside the application. This method eliminates the need to translate, rewrite, or recompile the application for each market. The only requirement to enter a new market is to localize the external information to the local language and customs.

Locale Categories

The locale categories are as follows:

`LC_CTYPE`	Controls the behavior of character handling functions.
`LC_TIME`	Specifies date and time formats, including month names, days of the week, and common full and abbreviated representations.
`LC_MONETARY`	Specifies monetary formats, including the currency symbol for the locale, thousands separator, sign position, the number of fractional digits, and so forth.
`LC_NUMERIC`	Specifies the decimal delimiter (or radix character), the thousands separator, and the grouping.
`LC_COLLATE`	Specifies a collation order and regular expression definition for the locale.
`LC_MESSAGES`	Specifies the language in which the localized messages are written, and affirmative and negative responses of the locale (`yes` and `no` strings and expressions).
`LO_LTYPE`	Specifies the layout engine that provides information about language rendering. Language rendering (or text rendering) depends on the shape and direction attributes of a script.

Using Locale Categories for Localization

The localization of a product should be done in consultation with native users in that target language or region. Certain information styles and formats might seem perfectly obvious and universal to the developer. However, to the user these formats could look awkward, wrong, or even offensive. The following sections describe the elements in the Oracle Solaris operating system that you can customize to meet the localization requirements for your product.

Time Formats

The following table shows some of the ways in which different locales write 11:59 P.M.

TABLE 1–1 International Time Formats

Locale	Format
Canadian	23:59
Finnish	23.59
German	23.59 Uhr
Norwegian	23.59
Thai	23:59
British English	23:59

Time is represented by both a 12-hour clock and a 24-hour clock. The hour and minute separator can be either a colon (:) or a period (.) or a dash (-).

Time zone splits occur between and within countries. Although a time zone can be described in terms of how many hours it is ahead of, or behind, Coordinated Universal Time, UTC (or Greenwich Mean Time, GMT), this number is not always an integer. For example, Newfoundland is in a time zone that is half an hour different from the adjacent time zone.

Daylight Savings Time (DST) starts and ends on dates that can vary from country to country. Many countries do not implement DST at all. Additionally, Daylight Savings Time can vary within a time zone. In the U.S. for example, the implementation is a state decision.

Date Formats

The following table shows some of the date formats used around the world. Variations can exist even within a country.

TABLE 1–2 International Date Formats

Locale	Convention	Example
Canadian (English)	dd/mm/yy	16/07/10
Danish	dd/mm/yy	16/07/10
Finnish	dd.mm.yyyy	16.07.2010
French	dd/mm/yy	16/07/10

TABLE 1–2 International Date Formats *(Continued)*

Locale	Convention	Example
German	dd.mm.yy	16.07.10
Italian	dd/mm/yy	16/07/10
Norwegian	dd.mm.yy	16.07.10
Spanish	dd/mm/yy	16/07/10
Swedish	yyyy-mm-dd	2010–07–16
Great Britain	dd/mm/yyyy	16/07/2010
United States	mm/dd/yy	07/16/10
Thai	mm/dd/yyyy	07/16/2010

Number Formats

Great Britain and the United States are two of the few places in the world that use a period to indicate the decimal place. Many other countries use a comma instead. The decimal separator is also called the *radix* character. Likewise, while Great Britain and the United States use a comma to separate groups of thousands, many other countries use a period instead, and some countries separate thousands groups with a thin space.

Data files containing locale-specific formats are frequently misinterpreted when transferred to a system in a different locale. For example, a file containing numbers in a French format is not useful to a British-specific program.

The following table shows some commonly used numeric formats.

TABLE 1–3 International Numeric Conventions

Locale	Large Number
Canadian (English)	4,294,967.00
Danish	4.294 967.295,00
Finnish	4 294 967 295,00
French	4 294 967 295,00
German	4,294,967.00
Italian	4.294.967,00
Norwegian	4.294.967.295,00
Spanish	4.294.967.295,00

TABLE 1–3 International Numeric Conventions *(Continued)*

Locale	Large Number
Swedish	4 294 967 295,00
Great Britain	4,294,967,295.00
United States	4,294,967,295.00
Thai	4,294,967,295.00

Note – No particular locale conventions exist that specify how to separate numbers in a list.

International Monetary Formats

Currency units and presentation order vary greatly around the world. Local and international symbols for currency can differ. The following table shows monetary formats in some countries.

TABLE 1–4 International Monetary Conventions

Locale	Currency	Example
Canadian (English)	Dollar ($)	$1,234.56
Canadian (French)	Dollar ($)	1 234,56$
Danish	Kroner (kr)	Kr 1.234,56
Finnish	Euro (€)	€ 1 234,56
French	Euro (€)	€ 1,234
Japanese	Yen (¥)	¥ 1,234
Norwegian	Krone (kr)	kr 1.234,56
Swedish	Krona (Kr)	1 234,56 Kr
Great Britain	Pound (£)	£1,234.56
United States	Dollar ($)	$1,234.56
Thai	Baht	2539 Baht

The current release supports the Euro currency. Local currency symbols are still available for backward compatibility.

TABLE 1–5 User Locales That Support the Euro Currency

Region	Locale Name	ISO Code Set
Austria	`de_AT.ISO8859-15`	8859-15
Belgium (French)	`fr_BE.ISO8859-15`	8859-15
Belgium (Flemish)	`nl_BE.ISO8859-15`	8859-15
Denmark	`da_DK.ISO8859-15`	8859-15
Estonia	`et_EE.ISO8859–15`	8859–15
Finland	`fi_FI.ISO8859-15`	8859-15
France	`fr_FR.ISO8859-15`	8859-15
Germany	`de_DE.ISO8859-15`	8859-15
Great Britain	`en_GB.ISO8859-15`	8859-15
Ireland	`en_IE.ISO8859-15`	8859-15
Italy	`it_IT.ISO8859-15`	8859-15
Netherlands	`nl_NL.ISO8859-15`	8859-15
Portugal	`pt_PT.ISO8859-15`	8859-15
Catalan Spain	`ca_ES.ISO8859-15`	8859–15
Spain	`es_ES.ISO8859-15`	8859-15
Sweden	`sv_SE.ISO8859-15`	8859-15
U.S.A.	`en_US.ISO8859-15`	8859-15

Euro locales are based on the ISO8859–15 code set.

Keep in mind that a *converted* currency amount can require a different amount of space than the original amount, for example, $1,000 can become 1 000,00 Kr.

The current status of the locale settings for locales within the euro zone is illustrated for the `LC_MONETARY` operand of the `locale` utility. The status for Germany, for example, is shown in the following table.

TABLE 1–6 German Locale and Corresponding `LC_MONETARY` Operand

Locale	LC_MONETARY
`de_DE.ISO8859–1`	DM
`de_DE.ISO8859–15`	Euro

TABLE 1–6 German Locale and Corresponding `LC_MONETARY` Operand *(Continued)*

Locale	LC_MONETARY
`de_DE.UTF-8`	Euro
`de_DE.ISO8859–15@euro`	Euro
`de_DE.UTF-8@euro`	Euro

Language Word and Letter Differences

This section describes important differences between languages.

Word Delimiters

In English, words are usually separated by a space character. Languages such as Chinese, Japanese, and Thai, however, often have no delimiter between words.

Sort Order

Sorting order for particular characters is not the same in all languages. For example, the character “ö” sorts with the ordinary “o” in Germany, but sorts separately in Sweden, where it is the last letter of the alphabet. In some languages, characters have weight to determine the priority of the character sequences. For example, the Thai dictionary defines sorting through the sequences of characters that have different weights.

Character Sets

Character sets can differ in the number of alphabetic characters and special characters. While the English alphabet contains only 26 characters, some languages contain many more characters. Japanese, for example, can contain over 20,000 characters and Chinese can contain an even higher number of characters.

Western European Alphabets

The alphabets of most western European countries are similar to the standard 26-character alphabet used in English-speaking countries. These alphabets often also include some additional basic characters, some marked or accented characters, and some ligatures.

Japanese Text

Japanese text is composed of three different scripts mixed together:

- Kanji ideographs derived from Chinese
- Hiragana and Katakana, two phonetic scripts (or syllabaries)

Although each character in Hiragana has an equivalent in Katakana, Hiragana is the most common script, with cursive rather than block-like letter forms. Kanji characters are used to write root words. Katakana is mostly used to represent “foreign” words, that is, words imported from languages other than Japanese.

Kanji has tens of thousands of characters, but the number commonly used has declined steadily over the years. Now only about 3500 are frequently used, although the average Japanese writer has a vocabulary of about 2000 Kanji characters. Nonetheless, computer systems must support more than 7000 characters in accordance with the Japan Industry Standard (JIS) requirements. In addition, there are about 170 Hiragana and Katakana characters. On average, 55% of Japanese text is Hiragana, 35% Kanji, and 10% Katakana. Arabic numerals and Roman letters are also present in Japanese text.

Although completely avoiding the use of Kanji is possible, most Japanese readers find a text that is composed without any Kanji hard to understand.

Korean Text

Korean text can be written using a phonetic writing system called Hangul. Hangul has more than 11,000 characters, which consist of consonants and vowels known as jamos. About 3000 characters from the entire Hangul vocabulary of characters are usually used in Korean computer systems. Korean also uses ideographs based on the set invented in China, called Hanja. Korean text requires over 6000 Hanja characters. Hanja is used mostly to avoid confusion when Hangul would be ambiguous. Hangul characters are formed by combining consonants and vowels. After these characters are combined, they can compose one syllable, which is a Hangul character. Hangul characters are often arranged in a square, so that the group takes up the same space as a Hanja character. Arabic numerals, Roman letters, and special symbol characters are also present in Korean text.

Thai Text

A Thai character can be defined as a column position on a display screen with four display cells. Each column position can have up to three characters. The composition of a display cell is based on the Thai character's classification. Some Thai characters can be composed with another character's classification. If both characters can be composed together, both characters are in the same cell. Otherwise, they are in separate cells.

Chinese Text

Chinese usually consists entirely of characters from the ideographic script called Hanzi.

- In the People's Republic of China (PRC) there are about 7000 commonly used Hanzi characters in the GB2312 (`zh` locale), more than 20,000 characters in the GBK charset (`zh.GBK` locale), and about 30,000 characters in the GB18030-2000 charset (`zh_CN.GB18030` locale), including all CJK extension A characters defined in Unicode 3.0.
- In Taiwan, the most frequently used charsets are the CNS11643-1992 (`zh_TW` locale) and the Big5 (`zh_TW.BIG5` locale). They share about 13,000 Hanzi characters.
- In Hong Kong, 4702 characters have been added into the Big5 charset to become the Big5-HKSCS charset (`zh_HK.BIG5HK`).

If a character is not a root character, it usually consists of two or more parts, two being most common. In two-part characters, one part generally represents meaning, and the other represents pronunciation. Occasionally both parts represent meaning. The radical is the most important element, and characters are traditionally arranged by radical, of which there are several hundred. A single sound can be represented by many different characters, which are not interchangeable in usage. A single character can have different sounds.

Some characters are more appropriate than others in a given context. The appropriate character is distinguished phonetically by the use of tones. By contrast, spoken Japanese and Korean lack tones.

Several phonetic systems represent Chinese. In the People's Republic of China the most common is *pinyin*, which uses Roman characters and is widely employed in the West for place names such as Beijing. The Wade-Giles system is an older phonetic system, formerly used for place names such as Peking. In Taiwan *zhuyin* (or *bopomofo*), a phonetic alphabet with unique letter forms, is often used instead.

Hebrew Text

Hebrew text is used for writing scripts in the Hebrew and Yiddish languages. Hebrew uses a bidirectional script. Hebrew letters are written and read from right to left, while numbers are read from left to right. Any English text that is embedded in Hebrew text is also read from left to right.

Hebrew uses a 27-character alphabet, and takes punctuation marks and numbers from the standard Latin (or English) alphabet. Hebrew text also includes vowel and pronunciation marks. These marks appear either as a dot (dagesh) inside the base character, vowel marks below the character, or accents to the upper left of the character. These marks are generally only used in liturgical text, and are rarely seen in day-to-day use. Hebrew has no uppercase letters.

Hindi Text

Hindi text is written in a script called Devanagari, which means the writing of the gods. Hindi is a phonetic language, and is written as a series of syllables. Each syllable is built up of alphabetic pieces (the Devanagari characters) of three types: consonant letters, independent vowels, and dependent vowel signs. The syllable itself consists of a consonant and vowel core, with an optional preceding consonant. Unlike English, which starts from a baseline, Devanagari characters hang from a horizontal line (called the head stroke) written at the top of the characters. These characters can combine or change shape depending on their context. Like Hebrew, Hindi text makes no distinction between uppercase and lowercase letters.

Keyboard Differences

Not all characters on the U.S. keyboard appear on other keyboards. Similarly, other keyboards often contain many characters not visible on the U.S. keyboard.

Any keyboard can be used to input characters from any locale because input is handled by the Oracle Solaris operating system.

Note – On SPARC® and on x86 based platform machines, the Compose key can be used to produce any Latin character with a diacritic in any of the supported ISO8859 character sets. The Compose key can be used with Latin-based locales, but not with Korean, Chinese, or Japanese locales, except the UTF-8 locales.

Differences in Paper Sizes

Within each country, a small number of paper sizes are commonly used. Normally, one of those sizes is much more common than the others. Most countries follow ISO Standard 216: "Writing paper and certain classes of printed matter-Trimmed sizes-A and B series."

Internationalized applications should not make assumptions about the page sizes available to them. The Oracle Solaris system provides no support for tracking the output page size. This tracking is the responsibility of the application program. The following table shows common international page sizes.

TABLE 1–7 Common International Page Sizes

Paper Type	Dimensions	Countries
ISO A4	21.0 cm by 29.7 cm	Everywhere except U.S.
ISO A5	14.8 cm by 21.0 cm	Everywhere except U.S.
JIS B4	25.9 cm by 36.65 cm	Japan
JIS B5	18.36 cm by 25.9 cm	Japan
U.S. Letter	8.5 inches by 11 inches	U.S. and Canada
U.S. Legal	8.5 inches by 14 inches	U.S. and Canada

CHAPTER 2

General Internationalization Features

This chapter discusses several internationalization features contained in the Oracle Solaris operating system. The chapter covers the following topics.

Support for Code Set Independence

EUC is an abbreviation for Extended UNIX® Code. The Oracle Solaris operating system supports non-EUC encodings such as PC-Kanji (better known as Shift_JIS) in Japan, Big5 in Taiwan, and GBK in the People's Republic of China. Because a large part of the computer market demands non-EUC codeset support, the current Oracle Solaris environment provides a solid framework to enable both EUC and non-EUC code set support. This support is called *Code Set Independence*, or CSI.

The goal of CSI is to remove dependencies on specific code sets or encoding methods from Oracle Solaris operating system libraries and commands. The CSI architecture enables the Oracle Solaris operating system to support any UNIX file system safe encoding. CSI supports a number of new code sets, such as UTF-8, PC-Kanji, and Big5.

CSI Approach

Code set independence enables application and platform software developers to keep their code independent of any encoding, such as UTF-8. CSI also provides the ability to adopt any new encoding without having to modify the source code. This architecture approach differs from Java internationalization because applications do not have to be to be UTF-16–dependent.

Many existing internationalized applications (for example, Motif) automatically inherit CSI support from the underlying system. These applications work in the new locales without modification.

CSI is inherently independent from any code sets. However, the following assumptions about file code encodings (code sets) still apply to the current Oracle Solaris system:

- File code is a superset of ASCII.
- NULL byte value (0x00) does not appear as part of multibyte character bytes for support of null-terminated multibyte character strings.
- ASCII Slash character byte value (0x2f) does not appear as part of multibyte character bytes for support of the UNIX path names.

CSI-enabled Commands

This section lists the CSI-enabled commands in the current Oracle Solaris environment. The man page for each command includes an attribute section that indicates whether the command is CSI-enabled.

All commands are in the `/usr/bin` directory, unless otherwise noted.

`/usr/lib/diffh`	`/usr/xpg4/bin/ls`	`apropos`
`/usr/sbin/accept`	`/usr/xpg4/bin/more`	`batch`
`/usr/sbin/reject`	`/usr/xpg4/bin/mv`	`bdiff`
`/usr/ucb/lpr`	`/usr/xpg4/bin/nice`	`cancel`
`/usr/xpg4/bin/awk`	`/usr/xpg4/bin/nohup`	`cat`
`/usr/xpg4/bin/cp`	`/usr/xpg4/bin/od`	`catman`
`/usr/xpg4/bin/date`	`/usr/xpg4/bin/pr`	`chgrp`
`/usr/xpg4/bin/du`	`/usr/xpg4/bin/rm`	`chmod`
`/usr/xpg4/bin/ed`	`/usr/xpg4/bin/sed`	`chown`
`/usr/xpg4/bin/edit`	`/usr/xpg4/bin/sort`	`cmp`
`/usr/xpg4/bin/egrep`	`/usr/xpg4/bin/tail`	`col`
`/usr/xpg4/bin/env`	`/usr/xpg4/bin/tr`	`comm`
`/usr/xpg4/bin/ex`	`/usr/xpg4/bin/vedit`	`compress`
`/usr/xpg4/bin/expr`	`/usr/xpg4/bin/vi`	`cpio`
`/usr/xpg4/bin/fgrep`	`/usr/xpg4/bin/view`	`csh`
`/usr/xpg4/bin/lp`	`acctcom`	`csplit`

`cut`	`mkdir`	`settime`
`diff`	`msgfmt`	`sh`
`diff3`	`news`	`split`
`disable`	`nroff`	`strconf`
`echo`	`pack`	`strings`
`expand`	`paste`	`sum`
`file`	`pcat`	`tabs`
`find`	`pg`	`tar`
`fold`	`printf`	`tee`
`ftp`	`priocntl`	`touch`
`gencat`	`ps`	`tty`
`geteopt`	`pwd`	`uncompress`
`getoptcvt`	`rcp`	`unexpand`
`head`	`red`	`uniq`
`join`	`remsh`	`unpack`
`jsh`	`rksh`	`wc`
`kill`	`rsh`	`whatis`
`ksh`	`rsmdir`	`write`
`lp`	`script`	`xargs`
`man`	`sdiff`	`zcat`

CSI-enabled Libraries

Nearly all functions in `libc` (`/usr/lib/libc.so`) are CSI-enabled. However, the following functions in `libc` are not CSI-enabled and therefore are EUC-dependent functions:

- `csetcol()`
- `csetlen()`
- `csetno()`
- `euccol()`
- `euclen()`
- `eucscol()`
- `getwidth()`
- `wcsetno()`

In the current Oracle Solaris environment, `libgen` `/usr/ccs/lib/libgen.a` and `libcurses` `/usr/ccs/lib/libcurses.a` are internationalized but not CSI-enabled.

Locale Database

The locale database format and structure is private and subject to change in a future release. When you develop internationalized applications, you use the internationalization APIs in `libc`. These APIs are described in "Internationalization APIs in `libc`" on page 41, rather than linking to the locale database.

Note – When you work in the Oracle Solaris environment, use the locale databases that are included with the current Oracle Solaris release. Do not use locales from previous Oracle Solaris versions.

Process Code Format

The process code format, which is also known as wide-character code format in the Oracle Solaris operating system, is private and subject to change in a future release. Therefore, when you develop an international application, do not assume that the process code format is the same. Instead, use the internationalization APIs in `libc` described in "Internationalization APIs in `libc`" on page 41.

Note – The process code for all Unicode locales is in UTF 32 representation. For more detail on UTF 32, refer to the Unicode Standard Annex #19: UTF 32 and Unicode Standard Annex #27: Unicode 3.1 from the Unicode Consortium or http://www.unicode.org/.

Multibyte Support Environment

A multibyte character is a character that cannot be stored in a single byte, such as Chinese, Japanese, or Korean characters. These characters require 2, 3, or 4 bytes of storage. A more precise definition can be found in ISO/IEC 9899:1990 subclause 3.13.

The Amendment 1 to ANSI C, which is also known as ISO/IEC 9899:1990, added new internationalization features, collectively known as the Multibyte Support Environment (MSE). Amendment 1 defines additional internationalization APIs for multibyte code sets with state and also for better wide-character handling support.

The programming model enables these multibyte characters to be read in as logical units and stored internally as wide characters. These wide characters can be processed by the program as logical entities. Finally, these wide characters can be written out, undergoing appropriate translation, as logical units.

This procedure is analogous to the way single-byte characters are read in, manipulated, and written out again. The MSE enables programs to handle multibyte characters using the same programming model that is used for single-byte characters.

Dynamically Linked Applications

You can link applications with the system libraries, such as `libc`, by using dynamic linking or static linking. Any application that requires internationalization features in the system libraries must be dynamically linked. If the application has been statically linked, the operation to set the locale to anything other than C and POSIX using the `setlocale` function will fail. Statically linked applications can operated only in C and POSIX locales.

By default, the linker program tries to link the application dynamically. If the command-line options to the linker and the compiler include `-Bstatic` or `-dn` specifications, your application might be statically linked. You can check whether an existing application is dynamically linked using the `/usr/bin/ldd` command.

For example, the response to the following command indicates that the `/sbin/sh` command is not a dynamically linked program:

```
% /usr/bin/ldd /sbin/sh
ldd: /sbin/sh: file is not a dynamic executable or shared object
```

The response to the following command indicates that the `/usr/bin/ls` command has been dynamically linked with two libraries, `libc.so.1` and `libdl.so.1`.

```
% /usr/bin/ldd /usr/bin/ls
libc.so.1 =>     /usr/lib/libc.so.1
libdl.so.1 => /usr/lib/libdl.so.1
```

Changed Interfaces

`libw` and `libintl` have moved to `libc` and are no longer in `libw` and `libintl`.

The shared objects ensure runtime compatibility for existing applications and, together with the archives, provide compilation environment compatibility for building applications. However, you no longer must build applications against `libw` or `libintl`.

The following list shows the stub entry points in `libw`:

```
fgetwc       isnumber      iswlower     strtows
fgetws       isphonogram   iswprint     towlower
fputwc       isspecial     iswpunct     towupper
fputws       iswalnum      iswspace     ungetwc
getwc        iswalpha      iswupper     watoll
getwchar     iswcntrl      iswxdigit    wcscat
getws        iswctype      putwc        wcschr
isenglish    iswdigit      putwchar     wcsclen
isideogram   iswgraph      putws        wcscmp
```

`wcscoll`	`wcstok`	`wscmp`	`wspbrk`
`wcscpy`	`wcstol`	`wscol`	`wsprintf`
`wcscspn`	`wcstoul`	`wscoll`	`wsrchr`
`wcsftime`	`wcswcs`	`wscpy`	`wsscanf`
`wscncat`	`wcswidth`	`wscspn`	`wsspn`
`wcsncmp`	`wcsxfrm`	`wsdup`	`wstod`
`wcsncpy`	`wctype`	`wslen`	`wstok`
`wcspbrk`	`wcwidth`	`wsncasecmp`	`wstol`
`wcsrchr`	`wscasecmp`	`wsncat`	`wstoll`
`wcsspn`	`wscat`	`wsncmp`	`wstostr`
`wcstod`	`wschr`	`wsncpy`	`wsxfrm`

The following list shows the stub entry points in `libintl`:

```
bindtextdomain
dcgettext
dgettext
gettext
textdomain
```

`ctype` Macros

Character classification and character transformation macros are defined in `/usr/include/ctype.h`. The current Oracle Solaris environment provides a set of `ctype` macros that support character classification and transformation semantics defined by XPG4. For all `XPG4` and `XPG4.2` applications to automatically access new macros, one of the following conditions must be met:

- `_XPG4_CHAR_CLASS` is defined.
- `_XOPEN_SOURCE` and `_XOPEN_VERSION=4` are defined.
- `_XOPEN_SOURCE` and `_XOPEN_SOURCE_EXTENDED=1` are defined.

Because `_XOPEN_SOURCE`, `_XOPEN_VERSION`, and `_XOPEN_SOURCE_EXTENDED` bring in extra `XPG4` related features in addition to new `ctype` macros, non-`XPG4` or `XPG4.2` applications should use `__XPG4_CHAR_CLASS__`.

Corresponding `ctype` functions also exist. The current Oracle Solaris environment functions also support `XPG4` semantics.

Internationalization APIs in `libc`

The current Oracle Solaris environment offers two sets of APIs:

- Multibyte (file codes)
- Wide characters (process code)

Wide-character codes are fixed-width units of logical entities. Therefore, you do not have to keep track of maintaining proper character boundaries when you are using multibyte characters.

When a program takes input from a file, you can convert your file's multibyte data into wide-character process code directly with input functions like `fscanf` and `fwscanf` or by using conversion functions like `mbtowc` and `mbsrtowcs` after the input. To convert output data from wide-character format to multibyte character format, use output functions like `fwprintf` and `fprintf` or apply conversion functions like `wctomb` and `wcsrtombs` before the output.

The tables in the remainder of this chapter describe the internationalization APIs included in the current Oracle Solaris system.

The following table describes the messaging function APIs in `libc`.

TABLE 2–1 Messaging Functions in `libcp`

Library Routine	Description
`bindtextdomain()`	Bind the path for a message domain
`catclose()`	Close a message catalog
`catgets()`	Read a program message
`catopen()`	Open a message catalog
`dcgettext()`	Get a message from a message catalog with domain and category specified
`dgettext()`	Get a message from a message catalog with domain specified
`gettext()`	Retrieve a text string from the message database
`textdomain()`	Set and query the current domain

The following table describes the code conversion function APIs in `libc`.

TABLE 2–2 Code Conversion in `libc`

Library Routine	Description
`iconv()`	Convert codes

TABLE 2–2 Code Conversion in `libc` *(Continued)*

Library Routine	Description
`iconv_close()`	Deallocate the conversion descriptor
`iconv_open()`	Allocate the conversion descriptor

The following table describes the regular expression APIs in `libc`.

TABLE 2–3 Regular Expressions in `libc`

Library Routine	Description
`fnmatch()`	Match file name or path name
`regcomp()`	Compile the regular expression
`regerror()`	Provide a mapping from error codes to error messages
`regexec()`	Execute regular expression matching
`regfree()`	Free memory allocated by `regcomp()`

The following table describes the wide character function APIs in `libc`.

TABLE 2–4 Wide Character Class in `libc`

Library Routine	Description
`wctrans()`	Define character mapping
`wctype()`	Define character class

The following table lists the modify and query locale in `libc`.

TABLE 2–5 Modify and Query Locale in `libc`

Library Routine	Description
`setlocale()`	Modify and query a program's locale

The following table lists the query locale data in `libc`.

TABLE 2–6 Query Locale Data in `libc`

Library Routine	Description
`localeconv()`	Get monetary and numeric formatting information of current locale

TABLE 2–6 Query Locale Data in `libc` *(Continued)*

Library Routine	Description
`nl_langinfo()`	Get language and cultural information of current locale

The following table describes the character classification function APIs in `libc`.

TABLE 2–7 Character Classification and Transliteration in `libc`

Library Routine	Description
`isalnum()`	Is character alphabetic or digital?
`isalpha()`	Is character alphabetic?
`isascii()`	Is character an ASCII character?
`iscntrl()`	Is character a control character?
`isdigit()`	Is character a digit?
`isenglish()`	Is wide character in English alphabet from a supplementary code set?
`isgraph()`	Is character a visible character?
`isideogram()`	Is wide character an ideogram?
`islower()`	Is character lowercase?
`isnumber()`	Is wide character a digit from a supplementary code set?
`isphonogram()`	Is wide character a phonogram?
`isprint()`	Is character printable?
`ispunct()`	Is character a punctuation mark?
`isspace()`	Is character a space?
`isspecial()`	Is special wide character from a supplementary code set?
`isupper()`	Is character uppercase?
`iswalnum()`	Is wide character an alphabetic character or digit?
`iswalpha()`	Is wide character alphabetic?
`iswascii()`	Is wide character an ASCII character?
`iswcntrl()`	Is wide character a control character?
`iswdigit()`	Is wide-character a digit?
`iswgraph()`	Is wide character a visible character?

TABLE 2–7 Character Classification and Transliteration in libc *(Continued)*

Library Routine	Description
iswlower()	Is wide character lowercase?
iswprint()	Is wide character a printable character?
iswpunct()	Is wide character a punctuation mark?
iswspace()	Is wide character a white space?
iswupper()	Is wide character uppercase?
iswxdigit()	Is wide character a hex digit?
isxdigit()	Is character a hex digit?
tolower()	Convert an uppercase character to lowercase.
toupper()	Convert a lowercase character to uppercase.
towctrans()	Wide character mapping.
towlower()	Convert an uppercase wide character to lowercase.
towupper()	Convert a lowercase wide character to uppercase.

The following table describes the character collation function APIs in `libc`.

TABLE 2–8 Character Collation in libc

Library Routine	Description
strcoll()	Collate character strings
strxfrm()	Transform character strings for comparison
wcscoll()	Collate wide-character strings
wcsxfrm()	Transform wide-character strings for comparison

The following table describes the monetary handling function APIs in `libc`.

TABLE 2–9 Monetary Formatting in libc

Library Routine	Description
localeconv()	Get monetary formatting information for the current locale
strfmon()	Convert monetary value to string representation

The following table describes the date and time formatting in `libc`.

TABLE 2–10 Date and Time Formatting in `libc`

Library Routine	Description
`getdate()`	Convert user format date and time.
`strftime()`	Convert date and time to string representation. The `%u` conversion function conforms to the X/Open CAE Specification, System Interfaces and Headers, Issue 4, Version 2. This function represents a weekday as a decimal number [1,7], with 1 now representing Monday.
`strptime()`	Date and time conversion.

The following table describes the multibyte handling function APIs in `libc`.

TABLE 2–11 Multibyte Handling in `libc`

Library Routine	Description
`btowc()`	Single-byte to wide-character conversion
`mblen()`	Get number of bytes in a character
`mbrlen()`	Get number of bytes in character (restartable)
`mbrtowc()`	Convert a character to a wide-character code (restartable)
`mbsinit()`	Determine conversion object status
`mbsrtowcs()`	Convert a character string to a wide-character string (restartable)
`mbstowcs()`	Convert a character string to a wide-character string
`mbtowc()`	Convert a character to a wide-character code

The following table describes the wide character and string handling in `libc`.

TABLE 2–12 Wide Character and String Handling in `libc`

Library Routine	Description
`wcrtomb()`	Convert a wide-character code to a character (restartable)
`wcscat()`	Concatenate wide-character strings
`wcschr()`	Find character in wide-character string
`wcscmp()`	Compare wide-character strings
`wcscpy()`	Copy wide-character strings

TABLE 2–12 Wide Character and String Handling in libc *(Continued)*

Library Routine	Description
`wcscspn()`	Return span of one wide-character string not in another
`wcslen()`	Get length of wide-character string
`wcsncat()`	Concatenate wide-character strings to length *n*
`wcsncmp()`	Compare wide-character strings to length *n*
`wcsncpy()`	Copy wide-character strings to length *n*
`wcspbrk()`	Return pointer to one wide-character string in another
`wcsrchr()`	Find character in wide-character string from right
`wcsrtombs()`	Convert a wide-character string to a character string (restartable)
`wcsspn()`	Return span of one wide-character string in another
`wcstod()`	Convert wide-character string to double precision
`wcstok()`	Move token through wide-character string
`wcstol()`	Convert wide-character string to long integer
`wcstombs()`	Convert wide-character string to multibyte string
`wcstoul()`	Convert wide-character string to unsigned long integer
`wscwcs()`	Find string in wide-character string
`wcswidth()`	Determine number of column positions of a wide-character string
`wctob()`	Wide character to single byte conversion
`wctomb()`	Convert wide-character to multibyte character
`wcwidth()`	Determine number of column positions of a wide character
`wscol()`	Return display width of wide-character string
`wsdup()`	Duplicate wide-character string

The following table describes the formatted wide-character input and output in `libc`.

TABLE 2–13 Formatted Wide-character Input and Output in libc

Library Routine	Description
`fwprintf()`	Print formatted wide-character output
`fwscanf()`	Convert formatted wide-character input
`swprintf()`	Print formatted wide-character output

TABLE 2–13 Formatted Wide-character Input and Output in `libc` *(Continued)*

Library Routine	Description
`swscanf()`	Convert formatted wide-character input
`vfwprintf()`	Wide-character formatted output of a `stdarg` argument list
`vswprintf()`	Wide-character formatted output of a `stdarg` argument list
`wprintf()`	Print formatted wide-character output
`wscanf()`	Convert formatted wide-character input
`wsprintf()`	Generate wide-character string according to format
`wsscanf()`	Formatted input conversion

This table describes the wide strings function APIs in `libc`.

TABLE 2–14 Wide Strings`libc`

Library Routine	Description
`wcsstr()`	Find a wide-character substring
`wmemchr()`	Find a wide character in memory
`wmemcmp()`	Compare wide characters in memory
`wmemcpy()`	Copy wide characters in memory
`wmemmove()`	Copy wide characters in memory with overlapping areas
`wmemset()`	Set wide characters in memory
`wscasecmp()`	Compare wide-character strings, ignore case differences
`wsncasecmp()`	Process code-string operations

The following table describes the wide-character input and output in `libc`.

TABLE 2–15 Wide-Character Input and Output in `libc`

Library Routine	Description
`fgetwc()`	Get multibyte character from stream, convert to wide character
`fgetws()`	Get multibyte string from stream, convert to wide character
`fputwc()`	Convert wide character to multibyte character, puts to stream
`fputws()`	Convert wide character to multibyte string, puts to stream
`fwide()`	Set stream orientation

TABLE 2–15 Wide-Character Input and Output in libc *(Continued)*

Library Routine	Description
`getwchar()`	Get multibyte character from `stdin`, convert to wide character
`getws()`	Get multibyte string from `stdin`, convert to wide character
`putwchar()`	Convert wide character to multibyte character, puts to `stdin`
`putws()`	Convert wide character to multibyte string, puts to `stdin`
`ungetwc()`	Push a wide character back into input stream

genmsg Utility

The new `genmsg` utility can be used with the `catgets()` family of functions to create internationalized source message catalogs. The utility examines a source program file for calls to functions in `catgets` and builds a source message catalog from the information it finds. For example:

```
% cat example.c
    ...
    /* NOTE: %s is a file name */
    printf(catgets(catd, 5, 1, "%s cannot be opened."));
    /* NOTE: "Read" is a past participle, not a present
             tense verb */
    printf(catgets(catd, 5, 1, "Read"));
    ...
% genmsg -c NOTE example.c
The following file(s) have been created.
            new msg file = "example.c.msg"
% cat example.c.msg
$quote "
$set 5
1            "%s cannot be opened"
    /* NOTE: %s is a file name */
2            "Read"
    /* NOTE: "Read" is a past participle, not a present
            tense verb */
```

In the above example, `genmsg` is run on the source file `example.c`, which produces a source message catalog named `example.c.msg`. The `-c` option with the argument `NOTE` causes `genmsg` to include comments in the catalog. If a comment in the source program contains the string specified, the comment appears in the message catalog after the next string extracted from a call to `catgets`.

You can use `genmsg` to number the messages in a message set automatically.

For more information, see the `genmsg`(1) man page.

To generate a formatted message catalog file, use the `gencat`(1) utility.

For information on the message extraction utility for portable message files (`.po` files) and also on how to generate message object files (`.mo` files) from the `.po` files.

User-Defined and User-Extensible Code Conversions

You can create user-defined codeset converters using the `geniconvtbl` utility.

This utility enables user-defined and user-customizable codeset conversions with a standard system utility and interface like `iconv`(1) and `iconv`(3C). This feature enhances the ability of an application to deal with incompatible data types, particularly data generated from proprietary or legacy applications. Modification to existing Oracle Solaris codeset conversions is also supported.

Sample input source files for the utility are available in the `/usr/lib/iconv/geniconvtbl/srcs/` directory.

Once the user-defined code conversions are prepared and placed properly, users can use the code conversions from the `iconv`(1) utility and the `iconv`(3C) functions of both 32-bit and 64-bit Oracle Solaris operating system.

Internationalized Domain Name (IDN) Support

Internationalized Domain Name (IDN) enables the use of non-English native language names as host and domain names. To use non-English host and domain names, convert these names into ASCII Compatible Encoding (ACE) encoded names before sending the names to resolver routines as specified in RFC 3490. System administrators are also required to use ACE names in system files and applications where the system administration applications do not support the IDNs.

See RFC 3490 Internationalizing Domain Names in Applications (IDNA).

The APIs for the Internationalized Domain Name in `libidnkit`(3EXT) provide convenient conversions between UTF-8 or the application locale's codeset and ACE. If `idn_decodename2`(3EXT) is used, you can also specify an arbitrary codeset name as the codeset of the input argument.

FIGURE 2–1 IDN to ACE Conversion

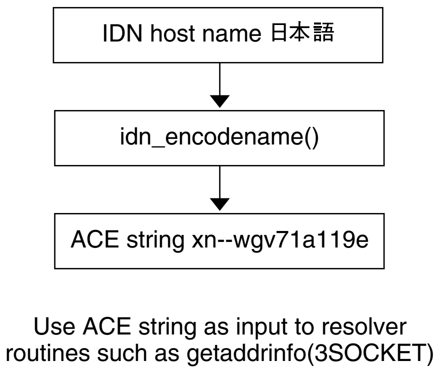
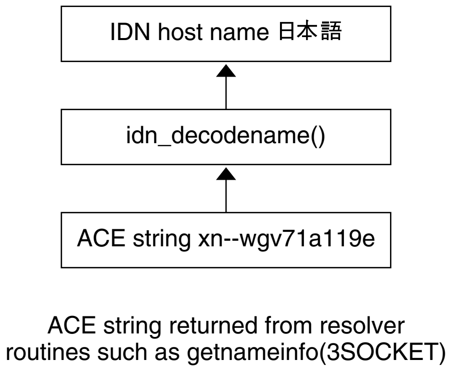

IDN host name 日本語
idn_encodename()
ACE string xn--wgv71a119e
Use ACE string as input to resolver routines such as getaddrinfo(3SOCKET)

FIGURE 2–2 ACE to IDN Conversion


IDN host name 日本語
idn_decodename()
ACE string xn--wgv71a119e
ACE string returned from resolver routines such as getnameinfo(3SOCKET)

The following table shows bilateral `iconv` code conversions that you can use.

TABLE 2–16 `iconv` Code Conversions

From Code	To Code
ACE	UTF-8
ACE-ALLOW-UNASSIGNED	UTF-8
UTF-8	ACE
UTF-8	ACE-ALLOW-UNASSIGNED

The ACE and the ACE-ALLOW-UNASSIGNED `iconv` code conversion names have the following meanings:

- ACE.

  ACE is a `fromcode` or `tocode` name that can be used in `iconv` code conversions to refer to the ASCII Compatible Encoding defined in RFC 3490. This conversion uses STD3 ASCII rules. Unassigned characters are not allowed. ACE is typically used for storing or giving host or domain names to machines.
- ACE-ALLOW-UNASSIGNED.

  ACE-ALLOW-UNASSIGNED performs the same operations as ACE except that ACE-ALLOW-UNASSIGNED allows unassigned characters. ACE-ALLOW-UNASSIGNED is typically used for query purpose.

The following example shows a conversion from ACE to UTF-8 with input from the `hostnames.txt` file. Output goes to standard output.

```
system% iconv -f ACE -t UTF-8 hostnames.txt
```

The dedicated IDN conversion utility idnconv(1) provides IDN conversions with various options. The options control the conversion details.

For information about IDN, the conversion routines, and `iconv` code conversions, see `libidnkit`(3LIB), `idn_decodename`(3EXT), `idn_decodename2`(3EXT), `idn_encodename`(3EXT), and `iconv_en_US.UTF-8`(5) man pages.

CHAPTER 3

Localization in the Oracle Solaris Environment

This chapter discusses the localization features in the current Oracle Solaris environment. The chapter covers the following topics.

- "Software Support for Localization" on page 53
- "Supported Locales" on page 55
- "Multiple Key Compose Sequences for Locales" on page 64
- "Keyboard Support in the Oracle Solaris Environment" on page 65
- "New Oracle Solaris Keyboard Software Support" on page 80

Software Support for Localization

This section contains information about the Oracle Solaris locale packages, DVD disk, localization functions, and script enabling.

Summary of the Oracle Solaris Locale Packages

All current Oracle Solaris locale packages are classified into either full locales or partial locales.

Partial locales are the enablers of the locales. With partial locales installed on the system, users can input, display, print text, and run applications on the target locales, while the OS/GUI messages in the Oracle Solaris operating system are English. All partial locale packages are available on the Oracle Solaris DVD. Japanese and Asian partial locales are packaged according to the language. Partial locales are packaged according to the geographic region.

Full locale packages include translations of software messages, online help files, optional fonts, and language-specific features. Full locale packages provide the full set of language features for many languages. All locales based on the following languages are full locales:

- German
- French

- Spanish
- Brazilian Portuguese
- Italian
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

Full locale packages are packaged according to the language and are available on the Oracle Solaris DVD.

Note – Partial locale packages (locale enablers) must be installed in order for the full locales to be functional.

During the Oracle Solaris installation process, you are prompted to choose which geographic regions' support you require. The locale support that is available after the installation is completed depends on the choices made at this stage. Note that the English locale ("C") is installed in any case.

Adding Additional Locales After Installation

You can configure additional locales by using the Locale administrator tool. Using the `localeadm`(1M) tool, you can display information about locale packages that are installed on the system or that reside on a particular device or directory. You can add and remove locales on the current system on a per-region basis. For example, you can add all locales in the Eastern European region to the current system.

The locale administrator now enables you to automatically add and remove locale packages on a per-region basis on the machine, once the system is installed. Because of this capability, you need not work with individual packages, and thus errors are reduced.

The locale administrator is a supplement to the locale selection logic in the Oracle Solaris installer. The installer is still considered as the primary application for the correct installation of Oracle Solaris locales.

Supported Locales

The following tables list all the locales supported in the Oracle Solaris environment. The locale names conform to international naming standards.

TABLE 3–1 Asia Locales

Locale	User Interface	Territory	Code Set	Language Support
`bn_IN.UTF-8`	English (*)	India	UTF-8	Bengali (UTF-8) Unicode4.0
`en_IN.UTF-8`	English	India	UTF-8	English (UTF-8) Unicode4.0
`en_SG.UTF-8`	English	Singapore	UTF-8	English (UTF-8) Unicode4.0
`gu_IN.UTF-8`	English (*)	India	UTF-8	Gujarati (UTF-8) Unicode4.0
`hi_IN.UTF-8`	English	India	UTF-8	Hindi (UTF-8) Unicode4.0
`id_ID.UTF-8`	English	Indonesia	UTF-8	Indonesian (UTF-8) Unicode4.0
`ja`	Japanese	Japan	eucJP[1]	Japanese (EUC)
				JIS X 0201-1976
				JIS X 0208-1990
				JIS X 0212-1990
`ja_JP.eucJP`	Japanese	Japan	eucJP	Japanese (EUC)
				JIS X 0201-1976
				JIS X 0208-1990
				JIS X 0212-1990
`ja_JP.PCK`	Japanese	Japan	PCK[2]	Japanese (PC Kanji)
				JIS X 0201-1976
				JIS X 0208-1990
`ja_JP.UTF-8`	Japanese	Japan	UTF-8	Japanese (UTF-8) Unicode4.0
`kn_IN.UTF-8`	English	India	UTF-8	Kannada (UTF-8) Unicode4.0
`ko_KR.EUC`	Korean	Korea	1001	Korean (EUC) KS X 1001
`ko_KR.UTF-8`	Korean	Korea	UTF-8	Korean (UTF-8) Unicode4.0
`mr_IN.UTF-8`	English	India	UTF-8	Marathi (UTF-8) Unicode4.0

[1] eucJP signifies the Japanese EUC code set. Specification of `ja_JP.eucJP` locale conforms to UI_OSF Japanese Environment Implementation Agreement Version 1.1 and `ja` locale conforms to the traditional specification from the past Solaris releases.

[2] PCK is also known as Shift_JIS (SJIS).

TABLE 3–1 Asia Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
ms_MY.UTF-8	English	Malaysia	UTF-8	Malay (UTF-8) Unicode4.0
ta_IN.UTF-8	English	India	UTF-8	Tamil (UTF-8) Unicode4.0
te_IN.UTF-8	English	India	UTF-8	Telugu (UTF-8) Unicode4.0
th_TH.ISO8859-11	English (*)	Thailand	ISO8859-11	Thai (ISO8859-11)
th_TH.UTF-8	English	Thailand	UTF-8	Thai (UTF-8) Unicode4.0
th_TH.TIS620	English	Thailand	TIS620.2533	Thai TIS620.2533
zh_CN.EUC	Simplified Chinese	PRC	gb2312[3]	Simplified Chinese (EUC) GB2312-1980
zh_CN.GBK	Simplified Chinese	PRC	GBK [4]	Simplified Chinese (GBK)
zh_CN.GB18030	Simplified Chinese	PRC	GB18030–2000	Simplified Chinese (GB18030–2000) GB18030–2000
zh_CN.UTF-8	Simplified Chinese	PRC	UTF-8	Simplified Chinese (UTF-8) Unicode4.0
zh_HK.BIG5HK	Traditional Chinese	Hong Kong	Big5+HKSCS	Traditional Chinese (BIG5+HKSCS)
zh_HK.UTF-8	Traditional Chinese	Hong Kong	UTF-8	Traditional Chinese (UTF-8) Unicode4.0
zh_SG.UTF-8	English	Singapore	UTF-8	Chinese (UTF-8) Unicode4.0
zh_TW.EUC	Traditional Chinese	Taiwan	cns11643	Traditional Chinese (EUC) CNS 11643-1992
zh_TW.BIG5	Traditional Chinese	Taiwan	BIG5	Traditional Chinese (BIG5)
zh_TW.UTF-8	Traditional Chinese	Taiwan	UTF-8	Traditional Chinese (UTF-8) Unicode4.0

[3] gb2312 signifies Simplified Chinese EUC code set, which contains GB 1988–80 and GB 2312–80.

[4] GBK signifies GB extensions. These extensions include all GB 2312–80 characters and all Unified Han characters of ISO/IEC 10646–1, as well as Japanese Hiragana and Katakana characters. GBK also includes many characters of Chinese, Japanese, and Korean character sets and of ISO/IEC 10646–1.

(*) Partially translated

TABLE 3–2 Australia Locales

Locale	User Interface	Territory	Code Set	Language Support
en_AU.ISO8859-1	English	Australia	ISO8859-1	English (Australia)
en_AU.UTF-8	English	Australia	UTF-8	English (UTF-8) Unicode4.0
en_NZ.ISO8859-1	English	New Zealand	ISO8859-1	English (New Zealand)
en_NZ.UTF-8	English	New Zealand	UTF-8	English (UTF-8) Unicode4.0

TABLE 3–3 Central America Locales

Locale	User Interface	Territory	Code Set	Language Support
es_CR.ISO8859-1	Spanish	Costa Rica	ISO8859-1	Spanish (Costa Rica)
es_CR.UTF-8	Spanish	Costa Rica	UTF-8	Spanish (UTF-8) Unicode4.0
es_GT.ISO8859-1	Spanish	Guatemala	ISO8859-1	Spanish (Guatemala)
es_GT.UTF-8	Spanish	Guatemala	UTF-8	Spanish (UTF-8) Unicode4.0
es_NI.ISO8859-1	Spanish	Nicaragua	ISO8859-1	Spanish (Nicaragua)
es_NI.UTF-8	Spanish	Nicaragua	UTF-8	Spanish (UTF-8) Unicode4.0
es_PA.ISO8859-1	Spanish	Panama	ISO8859-1	Spanish (Panama)
es_PA.UTF-8	Spanish	Panama	UTF-8	Spanish (UTF-8) Unicode4.0
es_SV.ISO8859-1	Spanish	El Salvador	ISO8859-1	Spanish (El Salvador)
es_SV.UTF-8	Spanish	El Salvador	UTF-8	Spanish (UTF-8) Unicode4.0

TABLE 3–4 Central Europe Locales

Locale	User Interface	Territory	Code Set	Language Support
cs_CZ.ISO8859-2	English	Czech Republic	ISO8859-2	Czech (Czech Republic)
cs_CZ.UTF-8@euro	Czech	Czech Republic	UTF-8	Czech (UTF-8) Unicode4.0
cs_CZ.UTF-8	Czech	Czech Republic	UTF-8	Czech (UTF-8) Unicode4.0
de_AT.ISO8859-1	German	Austria	ISO8859-1	German (Austria)
de_AT.ISO8859-15	German	Austria	ISO8859-15	German (Austria, ISO8859-15 - Euro)
de_AT.UTF-8	German	Austria	UTF-8	German (UTF-8) Unicode4.0

TABLE 3–4 Central Europe Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
`de_CH.ISO8859-1`	German	Switzerland	ISO8859-1	German (Switzerland)
`de_CH.UTF-8`	German	Switzerland	UTF-8	German (UTF-8) Unicode4.0
`de_DE.UTF-8`	German	Germany	UTF-8	German (Germany, Unicode4.0)
`de_DE.ISO8859-1`	German	Germany	ISO8859-1	German (Germany)
`de_DE.ISO8859-15`	German	Germany	ISO8859-15	German (Germany, ISO8859-15 - Euro)
`de_LU.UTF-8`	German	Luxembourg	UTF-8	German (UTF-8) Unicode4.0
`fr_CH.ISO8859-1`	French	Switzerland	ISO8859-1	French (Switzerland)
`fr_CH.UTF-8`	French	Switzerland	UTF-8	German (UTF-8) Unicode4.0
`hu_HU.ISO8859-2`	English	Hungary	ISO8859-2	Hungarian (Hungary)
`hu_HU.UTF-8`	Hungary	Hungarian	UTF-8	Hungarian (UTF-8) Unicode4.0
`pl_PL.ISO8859-2`	English	Poland	ISO8859-2	Polish (Poland)
`pl_PL.UTF-8`	English	Poland	UTF-8	Polish (Poland, Unicode4.0)
`sk_SK.ISO8859-2`	English	Slovakia	ISO8859-2	Slovak (Slovakia)
`sk_SK.UTF-8`	English (*)	Slovakia	UTF-8	Slovak (UTF-8) Unicode4.0

(*) Partially translated

TABLE 3–5 Eastern Europe Locales

Locale	User Interface	Territory	Code Set	Language Support
`bg_BG.ISO8859-5`	English	Bulgaria	ISO8859-5	Bulgarian (Bulgaria)
`bg_BG.UTF-8`	English (*)	Bulgaria	UTF-8	Bulgarian (UTF-8) Unicode4.0
`et_EE.ISO8859-15`	English	Estonia	ISO8859-15	Estonian (Estonia)
`et_EE.UTF-8`	Estonian (*)	Estonia	UTF-8	Estonian (UTF-8) Unicode4.0
`hr_HR.ISO8859-2`	English	Croatia	ISO8859-2	Croatian (Croatia)
`hr_HR.UTF-8`	Croatian (*)	Croatia	UTF-8	Croatian (UTF-8) Unicode4.0

TABLE 3–5 Eastern Europe Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
`lt_LT.ISO8859-13`	English	Lithuania	ISO8859-13	Lithuanian (Lithuania)
`lt_LT.UTF-8`	Lithuanian (*)	Lithuania	UTF-8	Lithuanian (UTF-8) Unicode4.0
`lv_LV.ISO8859-13`	English	Latvia	ISO8859-13	Latvian (Latvia)
`lv_LV.UTF-8`	Latvian (*)	Latvia	UTF-8	Latvian (UTF-8) Unicode4.0
`kk_KZ.UTF-8`	English	Kazakhstan	UTF-8	Kazakh (UTF-8) Unicode4.0
`mk_MK.ISO8859-5`	English	Macedonia	ISO8859-5	Macedonian (Macedonia)
`mk_MK.UTF-8`	Macedonian (*)	Macedonia	UTF-8	Macedonian (UTF-8) Unicode4.0
`ro_RO.ISO8859-2`	English	Romania	ISO8859-2	Romanian (Romania)
`ro_RO.UTF-8`	Romanian (*)	Romania	UTF-8	Romanian (UTF-8) Unicode4.0
`ru_RU.KOI8-R`	English	Russia	KOI8-R	Russian (Russia, KOI8-R)
`ru_RU.ANSI1251`	English	Russia	ansi-1251	Russian (Russia, ANSI 1251)
`ru_RU.ISO8859-5`	English	Russia	ISO8859-5	Russian (Russia)
`ru_RU.UTF-8`	English	Russia	UTF-8	Russian (Russia, Unicode4.0)
`sh_BA.ISO8859-2@bosnia`	English	Bosnia	ISO8859-2	Bosnian (Bosnia)
`sh_BA.UTF-8`	Serbo-Croatian (*)	Bosnia	UTF-8	Bosnian (UTF-8) Unicode4.0
`sl_SI.ISO8859-2`	English	Slovenia	ISO8859-2	Slovenian (Slovenia)
`sl_SI.UTF-8`	Slovenian (*)	Slovenia	UTF-8	Slovenian (UTF-8) Unicode4.0
`sq_AL.ISO8859-2`	English	Albania	ISO8859-2	Albanian (Albania)
`sq_AL.UTF-8`	Albanian (*)	Albania	UTF-8	Albanian (UTF-8) Unicode4.0
`sr_CS.UTF-8`	English (**)	Serbian	UTF-8	Serbia and Montenegro (UTF-8) Unicode4.0
`sr_YU.ISO8859-5`	English	Serbia	ISO8859-5	Serbian (Serbia)

TABLE 3–5 Eastern Europe Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
`tr_TR.ISO8859-9`	English	Turkey	ISO8859-9	Turkish (Turkey)
`tr_TR.UTF-8`	English	Turkey	UTF-8	Turkish (Turkey, Unicode4.0
`uk_UA.UTF-8`	English	Ukraine	UTF-8	Ukrainian (UTF-8) Unicode4.0

(*) Partially translated

(**) Will be obsolete in the next release

TABLE 3–6 Middle East Locale

Locale	User Interface	Territory	Code Set	Language Support
`he`	English	Israel	ISO8859-8	Hebrew (Israel)
`he_IL.UTF-8`	English (*)	Israel	UTF-8	(UTF-8) Unicode4.0

(*) Partially translated

TABLE 3–7 North Africa Locales

Locale	User Interface	Territory	Code Set	Language Support
`ar`	English	Egypt	ISO8859-6	Arabic (Egypt)
`ar_EG.UTF-8`	English	Egypt	UTF-8	Arabic (Egypt)
`ar_SA.UTF-8`	Arabic (*)	Saudi Arabia	UTF-8	Arabic (UTF-8) Unicode4.0

(*) Partially translated

TABLE 3–8 North America Locales

Locale	User Interface	Territory	Code Set	Language Support
`en_CA.ISO8859-1`	English	Canada	ISO8859-1	English (Canada)
`en_CA.UTF-8`	English	Canada	UTF-8	English (UTF-8) Unicode4.0
`en_US.ISO8859-1`	English	USA	ISO8859-1	English (U.S.A.)
`en_US.ISO8859-15`	English	USA	ISO8859-15	English (U.S.A., ISO8859-15 - Euro)
`en_US.UTF-8`	English	USA	UTF-8	English (U.S.A., Unicode4.0)

TABLE 3–8 North America Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
es_MX.ISO8859–1	Spanish	Mexico	ISO8859–1	Spanish (Mexico)
es_MX.ISO8859-1	Spanish	Mexico	ISO8859-1	Spanish (ISO8859-1)
es_MX.UTF-8	Spanish	Mexico	UTF-8	Spanish (UTF-8) Unicode4.0
fr_CA.ISO8859-1	French	Canada	ISO8859-1	French (Canada)
fr_CA.UTF-8	French	Canada	UTF-8	French (UTF-8) Unicode4.0

TABLE 3–9 Northern Europe Locales

Locale	User Interface	Territory	Code Set	Language Support
da_DK.ISO8859–1	English	Denmark	ISO8859–1	Danish (Denmark)
da_DK.UTF-8	English (*)	Denmark	UTF-8	Danish (UTF-8) Unicode4.0
da_DK.ISO8859–15	English	Denmark	ISO8859–15	Danish (Denmark, ISO8859–15–Euro)
fi_FI.ISO8859–1	English	Finland	ISO8859–1	Finnish, Unicode4.0
fi_FI.ISO8859–15	English	Finland	ISO8859–15	Finnish (Finland, ISO8859–15–Euro)
fi_FI.UTF-8	English	Finland	UTF-8	Finnish (Finland)
is_IS.ISO8859–1	English	Iceland	ISO8859–1	Icelandic (Iceland)
is_IS.UTF-8	English (*)	Iceland	UTF-8	Icelandic (UTF-8) Unicode4.0
nb_NO.ISO8859-1@bokmal	English (*)	Norway	ISO8859-1	Norwegian Bokmel (ISO8859-1)
nb_NO.UTF-8	English (*)	Norway	UTF-8	Norwegian (UTF-8) Unicode4.0
nn_NO.ISO8859-1@nynorsk	English (*)	Norway	ISO8859-1	Norwegian (ISO8859-1)
nn_NO.UTF-8	English (*)	Norway	UTF-8	Norwegian Nynorsk (UTF-8) Unicode4.0
no_NO.ISO8859–1@bokmal	English	Norway	ISO8859–1	Norwegian (Norway-Bokmal)
no_NO.ISO8859-1@nynorsk	English	Norway	ISO8859–1	Norwegian (Norway-Nynorsk)
sv_SE.ISO8859–1	Swedish	Sweden	ISO8859–1	Swedish (Sweden)

TABLE 3–9 Northern Europe Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
sv_SE.ISO8859–15	Swedish	Sweden	ISO8859–15	Swedish (Sweden, ISO8859–15–Euro)
sv_SE.UTF-8	Swedish	Sweden	UTF-8	Swedish (Sweden, Unicode4.0)

(*) Partially translated

TABLE 3–10 South America Locales

Locale	User Interface	Territory	Code Set	Language Support
es_AR.ISO8859-1	Spanish	Argentina	ISO8859-1	Spanish (Argentina)
es_AR.UTF-8	Spanish	Argentina	UTF-8	Spanish (UTF-8) Unicode4.0
es_BO.ISO8859-1	Spanish	Bolivia	ISO8859-1	Spanish (Bolivia)
es_BO.UTF-8	Spanish	Bolivia	UTF-8	Spanish (UTF-8) Unicode4.0
es_CL.ISO8859-1	Spanish	Chile	ISO8859-1	Spanish (Chile)
es_CL.UTF-8	Spanish	Chile	UTF-8	Spanish (UTF-8) Unicode4.0
es_CO.ISO8859-1	Spanish	Colombia	ISO8859-1	Spanish (Colombia)
es_CO.UTF-8	Spanish	Colombia	UTF-8	Spanish (UTF-8) Unicode4.0
es_EC.ISO8859-1	Spanish	Ecuador	ISO8859-1	Spanish (Ecuador)
es_EC.UTF-8	Spanish	Ecuador	UTF-8	Spanish (UTF-8) Unicode4.0
es_PE.ISO8859-1	Spanish	Peru	ISO8859-1	Spanish (Peru)
es_PE.UTF-8	Spanish	Peru	UTF-8	Spanish (UTF-8) Unicode4.0
es_PY.ISO8859-1	Spanish	Paraguay	ISO8859-1	Spanish (Paraguay)
es_PY.UTF-8	Spanish	Paraguay	UTF-8	Spanish (UTF-8) Unicode4.0
es_UY.ISO8859-1	Spanish	Uruguay	ISO8859-1	Spanish (Uruguay)
es_UY.UTF-8	Spanish	Uruguay	UTF-8	Spanish (UTF-8) Unicode4.0
es_VE.ISO8859-1	Spanish	Venezuela	ISO8859-1	Spanish (Venezuela)
es_VE.UTF-8	Spanish	Venezuela	UTF-8	Spanish (UTF-8) Unicode4.0
pt_BR.ISO8859-1	English	Brazil	ISO8859-1	Portuguese (Brazil)
pt_BR.UTF-8	English	Brazil	UTF-8	Portuguese (Brazil, Unicode4.0)

TABLE 3–11 Southern Europe Locales

Locale	User Interface	Territory	Code Set	Language Support
`ca_ES.ISO8859-1`	English	Spain	ISO8859-1	Catalan (Spain)
`ca_ES.ISO8859-15`	English	Spain	ISO8859-15	Catalan (Spain, ISO8859-15 - Euro)
`ca_ES.UTF-8`	English (*)	Spain	UTF-8	Catalan (UTF-8) Unicode4.0
`el_CY.UTF-8`	English (*)	Cyprus	UTF-8	Greek (UTF-8) Unicode4.0
`el_GR.ISO8859-7`	English	Greece	ISO8859-7	Greek (Greece)
`el_GR.UTF-8`	English (*)	Greece	UTF-8	Greek (UTF-8) Unicode4.0
`en_MT.UTF-8`	English	Malta	UTF-8	English (UTF-8) Unicode4.0
`es_ES.ISO8859-1`	Spanish	Spain	ISO8859-1	Spanish (Spain)
`es_ES.ISO8859-15`	Spanish	Spain	ISO8859-15	Spanish (Spain, ISO8859-15 - Euro)
`es_ES.UTF-8`	Spanish	Spain	UTF-8	Spanish (Spain, Unicode4.0)
`it_IT.ISO8859-1`	Italian	Italy	ISO8859-1	Italian (Italy)
`it_IT.ISO8859-15`	Italian	Italy	ISO8859-15	Italian (Italy, ISO8859-15 - Euro)
`it_IT.UTF-8`	Italian	Italy	UTF-8	Italian (Italy, Unicode4.0)
`pt_PT.ISO8859-1`	English	Portugal	ISO8859-1	Portuguese (Portugal)
`pt_PT.ISO8859-15`	English	Portugal	ISO8859-15	Portuguese (Portugal, ISO8859-15 - Euro)
`pt_PT.UTF-8`	English (*)	Portugal	UTF-8	Portuguese (UTF-8) Unicode4.0

(*) Partially translated

TABLE 3–12 Western Europe Locales

Locale	User Interface	Territory	Code Set	Language Support
`en_GB.ISO8859-1`	English	Great Britain	ISO8859-1	English (Great Britain)
`en_GB.UTF-8`	English	United Kingdom	UTF-8	English (UTF-8) Unicode4.0
`en_IE.ISO8859-1`	English	Ireland	ISO8859-1	English (Ireland)
`en_IE.ISO8859-15`	English	Ireland	ISO8859-15	English (ISO8859-15

TABLE 3–12 Western Europe Locales *(Continued)*

Locale	User Interface	Territory	Code Set	Language Support
`en_IE.UTF-8`	English	Ireland	UTF-8	English (UTF-8) Unicode4.0
`fr_BE.ISO8859-1`	French	Belgium-Walloon	ISO8859-1	French (Belgium-Walloon, Unicode4.0)
`fr_BE.ISO8859-15`	French	Belgium	ISO8859-15	French (ISO8859-15)
`fr_BE.UTF-8`	French	Belgium-Walloon	UTF-8	French (Belgium-Walloon, Unicode4.0)
`fr_FR.ISO8859-1`	French	France	ISO8859-1	French (France)
`fr_FR.UTF-8`	French	France	UTF-8	French (France, Unicode4.0)
`fr_LU.UTF-8`	French	Luxembourg	UTF-8	French (UTF-8) Unicode4.0
`nl_BE.ISO8859-1`	English	Belgium-Flemish	ISO8859-1	Dutch (Belgium-Flemish)
`nl_BE.ISO8859-15`	Belgian (*)	Belgium	ISO8859-15	Dutch (ISO8859-15)
`nl_BE.UTF-8`	Belgian (*)	Belgium	UTF-8	Dutch (UTF-8) Unicode4.0
`nl_NL.ISO8859-1`	English	Netherlands	ISO8859-1	Dutch (Netherlands)
`nl_NL.ISO8859-15`	Belgian (*)	Netherlands	ISO8859-15	Dutch (ISO8859-15)
`nl_NL.UTF-8`	Belgian (*)	Netherlands	UTF-8	Dutch (UTF-8) Unicode4.0

(*) Partially translated

TABLE 3–13 Southern Africa Locales

Locale	User Interface	Territory	Code Set	Language Support
`af_ZA.UTF-8`	English	South Africa	UTF-8	Afrikaans (UTF-8) Unicode4.0

Multiple Key Compose Sequences for Locales

Many of the Oracle Solaris locales, especially the European and Unicode locales, allow input of various characters by using the Dead key sequences and also by using Compose key sequences.

Dead keys are included in some specific keyboard layouts where it is necessary to generate characters composed by two or more letters and symbols. For example, in the French Keyboard Layout, character egrave (è) can be generated by typing dead_grave (‘) and (e).

The Compose key sequence are used to input characters with diacritical marks and other characters that are not shown on the keyboard key caps.

The following table shows a few examples of Compose key sequences. For more complete information about the Compose key sequences, see "How to Use Compose and Dead Key Input" on page 171.

TABLE 3–14 Diacritical Characters Created With Compose Key

Mark	Compose Key Combination	Example
Dieresis	"	Compose A " —> A with diaeresis
Caron	v	Compose Z v —> Z with caron
Breve	u	Compose G u —> G with breve
Ogonek	a	Compose A a —> A with Ogonek
Cedilla	,	Compose K , —> K with cedilla
Registered Sign	R O	Compose R O —> Registered sign
Inverted Exclamation Mark	! !	Compose ! ! —> Inverted Exclamation Mark

Note – A compose key sequence cannot produce a character unless the character is a part of the code set in the current locale. For example, because no Z with a caron is in the ISO8859–1 codeset, you cannot input a Z with a caron in the `en_US.ISO8859–1` locale.

Keyboard Support in the Oracle Solaris Environment

Keyboards with different layouts for specific regions are supported for SPARC and Intel Architecture (IA) platforms. The Oracle Solaris operating system supports the regional keyboards listed in the following table.

TABLE 3–15 Support for Regional Keyboards

Region	Country	Sun Keyboard (Type 4/5/5c)	Sun Keyboard (Type 6)	PC Keyboard
Asia	Japan	X	X	X
	Korea	X	X	X
	Taiwan	X	X	X
Europe	Belgium	X	X	X
	Czech Republic	X		X
	Denmark	X	X	X
	Finland		X	

TABLE 3–15 Support for Regional Keyboards *(Continued)*

Region	Country	Sun Keyboard (Type 4/5/5c)	Sun Keyboard (Type 6)	PC Keyboard
	France	X	X	X
	Germany	X	X	X
	Great Britain	X	X	X
	Greece	X		X
	Hungary	X		X
	Italy	X	X	X
	Latvia	X		X
	Lithuania	X		X
	The Netherlands	X	X	X
	Norway	X	X	X
	Poland	X		X
	Portugal	X	X	X
	Russia	X	X	X
	Spain	X	X	X
	Sweden	X	X	X
	Switzerland (French)	X	X	X
	Switzerland (German)	X	X	X
	Turkey	X	X	X
America	Canada (French)	X	X	X
	Latin America (Spanish)	X		
	U.S.A.	X	X	X
Middle East	Arabic	X	X	

Additionally for the Xorg server, the Oracle Solaris operating system supports the following regional keyboards:

- U.S. English
- Arabic
- Albania
- Armenia
- Azerbaijan

- Belarus
- Belgium
- Bangladesh
- India
- Bosnia and Herzegovina
- Brazil
- Bulgaria
- Myanmar
- Canada
- Croatia
- Czechia
- Denmark
- Netherlands
- Bhutan
- Estonia
- Iran
- Faroe Islands
- Finland
- France
- Georgia
- Germany
- Greece
- Hungary
- Iceland
- Israel
- Italy
- Japan
- Kyrgyzstan
- Kazakhstan
- Laos
- Latin American
- Lithuania
- Latvia
- Maori
- Macedonian
- Malta
- Mongolia
- Norway
- Poland
- Portugal
- Romania
- Russia
- Serbian
- Slovenia

- Slovakia
- Spain
- Sweden
- Swiss French
- Swiss German
- Syria
- Tajikistan
- Sri Lanka
- Thailand
- Turkish
- Turkish (F)
- Ukraine
- United Kingdom
- Uzbekistan
- Vietnam
- Ireland
- Pakistan
- South Africa

For regions with keyboard layouts that conform to the international standard such as China, use the keyboard layout support provided for U.S.A. to input the locale's characters. The underlying keyboard mappings are identical. Some countries, like Japan, Turkey, and Switzerland, have multiple keyboards, because multiple languages are being used, or because multiple keyboard layouts exist.

Sun Type 4, 5, and 5c keyboards use Sun I/O interfaces through a Mini DIN 8–pin connection. Sun Type 6 keyboards have two versions of interfaces:

- Sun I/O through a Mini DIN 8–pin connection
- USB

Sun keyboard types are printed on the back of each Sun keyboard.

PC keyboards use various interfaces, such as PS/2 or USB, for example.

Changing Between Keyboards on SPARC Systems

You can change keyboard layouts on a Oracle Solaris system by using the DIP switch settings under most Sun Type 4, 5 and 5c keyboards. A list of keyboard type, names and corresponding layout IDs that can be used for the DIP switch settings is in the `/usr/openwin/share/etc/keytables/keytable.map` file.

Note – You cannot change the layout of Type 6 keyboards because the back of the keyboard has no DIP switch. Some Type 5 and 5c keyboards, for example, U.S.A., U.S.A./UNIX, and Japanese keyboards have jumpers instead of DIP switches. Other than the `xmodmap` utility or the `kbd -s` command, the SPARC platform does not offer utilities or tools that you can use to switch keyboards.

The following is a table of the layout ID values for Type 4, 5, and 5c keyboards (1 = switch up, 0 = switch down).

TABLE 3–16 Layouts for Type 4, 5, and 5c Keyboards

DIP Switch	Keyboard (Keytable File)	Setting in Binary
0	U.S.A. (`US4.kt`)	000000
1	U.S.A. (`US4.kt`)	000001
2	Belgium (`FranceBelg4.kt`)	000010
3	Canada (`Canada4.kt`)	000011
4	Denmark (`Denmark4.kt`)	000100
5	Germany (`Germany4.kt`)	000101
6	Italy (`Italy4.kt`)	000110
7	The Netherlands (`Netherland4.kt`)	000111
8	Norway (`Norway4.kt`)	001000
9	Portugal (`Portugal4.kt`)	001001
10 (0x0a)	Latin America/Spanish (`SpainLatAm4.kt`)	001010
11 (ox0b)	Sweden (`SwedenFin4.kt`)	001011
12 (0x0c)	Switzerland/French (`Switzer_Fr4.kt`)	001100
13 (0x0d)	Switzerland/German (`Switzer_Ge4.kt`)	001101
14 (0x0e)	Great Britain (`UK4.kt`)	001110
16 (0x10)	Korea (`Korea4.kt`)	010000
17 (0x11)	Taiwan (`Taiwan4.kt`)	010001
23	Russian	100001
33 (0x21)	U.S.A. (`US5.kt`)	100111
34 (0x22)	U.S.A./UNIX (`US_UNIX5.kt`)	100010

TABLE 3–16 Layouts for Type 4, 5, and 5c Keyboards *(Continued)*

DIP Switch	Keyboard (Keytable File)	Setting in Binary
35 (0x23)	France (`France5.kt`)	100011
36 (0x24)	Denmark (`Denmark5.kt`)	100100
37 (0x25)	Germany (`Germany5.kt`)	100101
38 (0x26)	Italy (`Italy5.kt`)	100110
39 (0x27)	The Netherlands (`Netherland5.kt`)	100111
40 (0x28)	Norway (`Norway5.kt`)	101000
41 (0x29)	Portugal (`Portugal5.kt`)	101001
42 (0x2a)	Spain (`Spain5.kt`)	101010
43 (0x2b)	Sweden (`Sweden5.kt`)	101011
44 (0x2c)	Switzerland/French (`Switzer_Fr5.kt`)	101101
45 (0x2d)	Switzerland/German (`Switzer_Ge5.kt`)	101110
46 (0x2e)	Great Britain (`UK5.kt`)	101111
47 (0x2f)	Korea (`Korea5.kt`)	101111
48 (0x30)	Taiwan (`Taiwan5.kt`)	110000
49 (0x31)	Japan (`Japan5.kt`)	110001
50 (0x32), see also 63 (0x3f)	Canada/French (`Canada_Fr5.kt`)	110010
51 0(x33)	Hungary (`Hungary5.kt`)	110011
52 (0x34	Poland (`Poland5.kt`)	110100
53 (0x35)	Czech (`Czech5.kt`)	110101
54 (0x36)	Russia (`Russia5.kt`)	110110
55 (0x37)	Latvia (`Latvia5.kt`)	110111
56 (0x38) see also 62 (0x3e)	Turkey-Q5 (`TurkeyQ5.kt`)	111000
57 (0x39)	Greece (`Greece5.kt`)	111001
58 (0x3a)	Arabic (`Arabic5.kt`)	111011
59 (0x3b)	Lithuania (`Lithuania5.kt`)	111010
60 (0x3c)	Belgium (`Belgian5.kt`)	111100

TABLE 3–16 Layouts for Type 4, 5, and 5c Keyboards *(Continued)*

DIP Switch	Keyboard (Keytable File)	Setting in Binary
62 (0x3e)	Canada/French (`Canada_Fr5_TBITS5.kt`) French Canadian Polish Programmer Estonian	111111

Keytable file names with 4 are for a Type 4 keyboard. Keytable file names with 5 are for a Type 5 keyboard.

▼ How to Change the Keyboard Layout to the Czech Layout in the Xsun Server

1. **Determine the correct DIP switch ID (or layout ID) either from the table or from the `/usr/openwin/share/etc/keytables/keytable.mp` file. The layout ID value in the `keytable.mp` file is a decimal value.**

   For Czech, the layout ID is 53 in decimal (0x35 in hexadecimal).

2. **Convert the layout ID to binary, or use a proper Setting in Binary value from the column in the above table. For base conversion, calculator utilities such as `dtcalc`(1) may be used.**

   For example, the correct binary value for the Czech keyboard is 110101.

3. **Shut down and power off the system.**

4. **Change the DIP switch settings at the back of the keyboard by using the binary value in step 2.**

   The first DIP switch is on your left. Move the switch up for 1 and down for 0.

   The Czech keyboard binary value 110101, corresponds to: Up Up Down Up Down Up

5. **Power on and boot the system for use.**

Note – Unlike Type 4 keyboards, Type 5 and 5c keyboards have only five DIP switches. For the Type 5 and 5c keyboards, disregard the first binary digit. For the Czech Type 5c keyboard, for example, the correct DIP switch settings are Up Down Up Down Up, using only the last five digits from 10101.

Changing Between Keyboards on Intel Systems

On Intel Architecture systems, a keyboard is selected during the `kdmconfig(1M)` part of the installation. To change this setting after installation, exit your GUI desktop environment to the command-line mode. As superuser, type `kdmconfig` to run the program. Follow the instructions to get the desired keyboard layout.

Additionally you can use the `setxkbmap` feature to change the keyboard layout simultaneously.

How to Change the Keyboard Layout to the Czech Layout in the Xorg server

The `setxkbmap` allows switching the keyboard layout simultaneously when using the Xorg server. This command maps the keyboard. It uses the different command line options. For more information, see `setxkbmap` man pages.

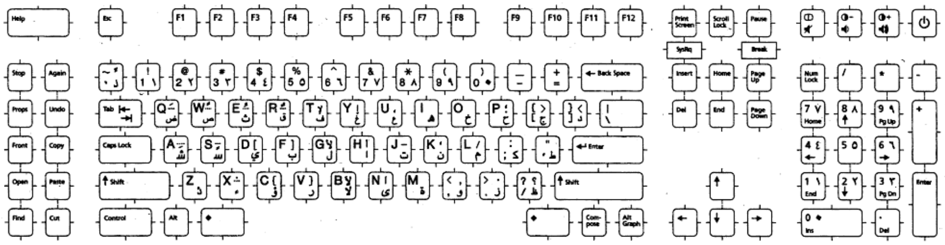
Open a terminal and type the following command:

```
$ /usr/X11/bin/setxkbmap cz
```

Keyboard Layout Illustrations

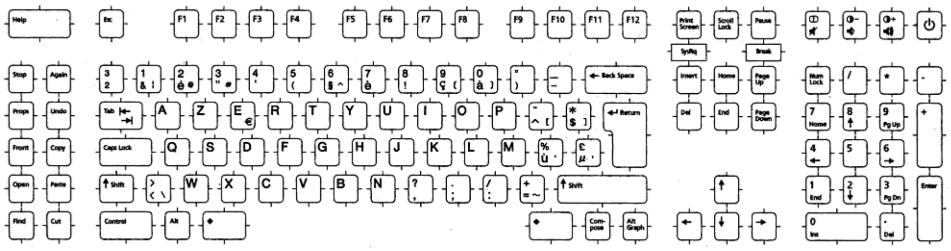
The following figure shows the Arabic keyboard.

FIGURE 3–1 Arabic Keyboard



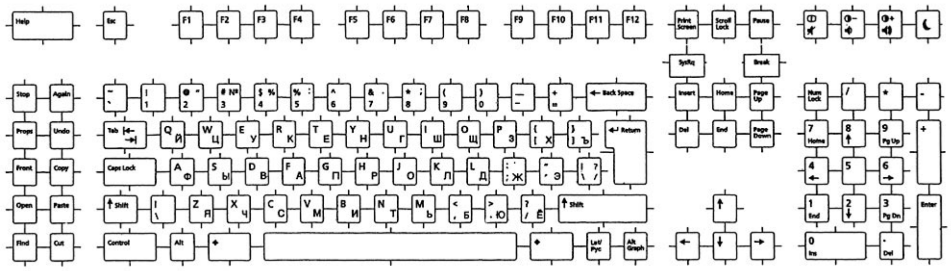
The following figure shows the Belgian keyboard.

FIGURE 3–2 Belgian Keyboard



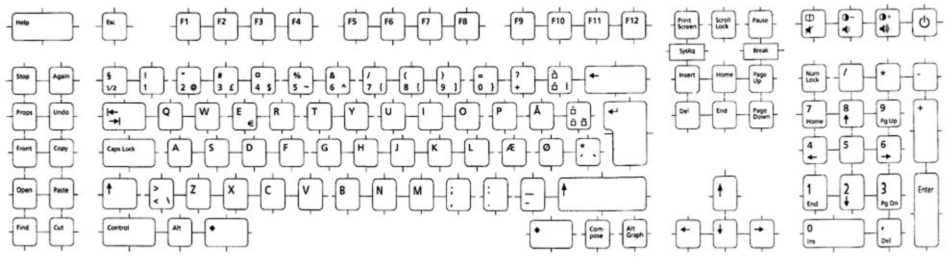
The following figure shows the Cyrillic keyboard.

FIGURE 3–3 Cyrillic (Russian) Keyboard



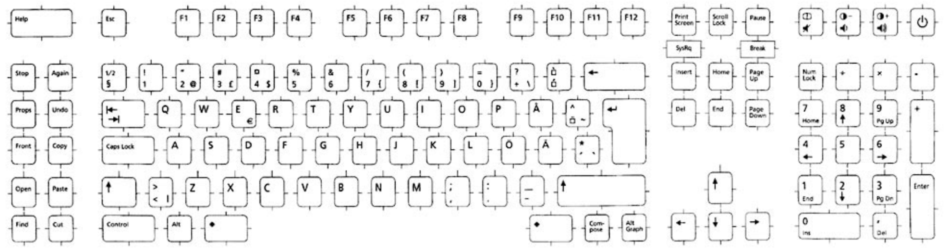
The following figure shows the Danish keyboard.

FIGURE 3–4 Danish Keyboard



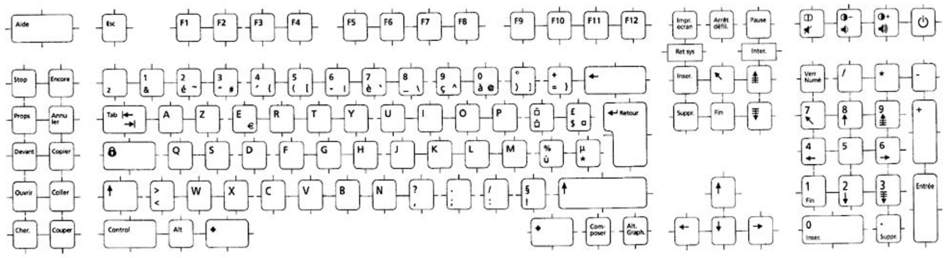
The following figure shows the Finnish keyboard.

FIGURE 3–5 Finnish Keyboard



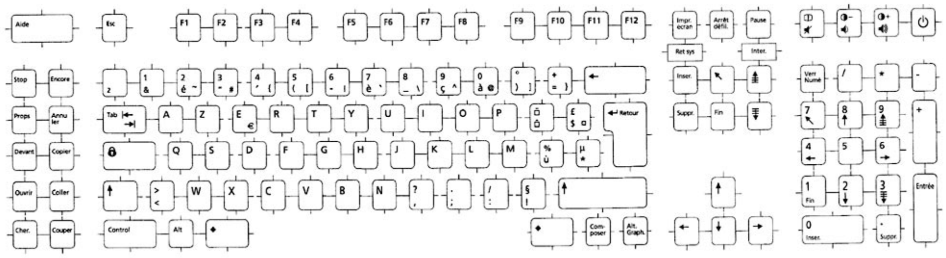
The following figure shows the French keyboard.

FIGURE 3–6 French Keyboard



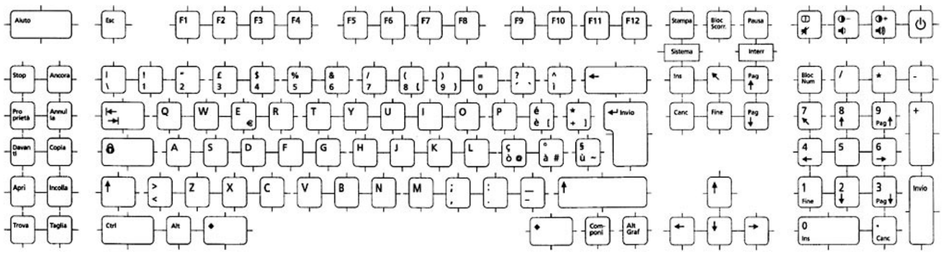
The following figure shows the German keyboard.

FIGURE 3–7 German Keyboard



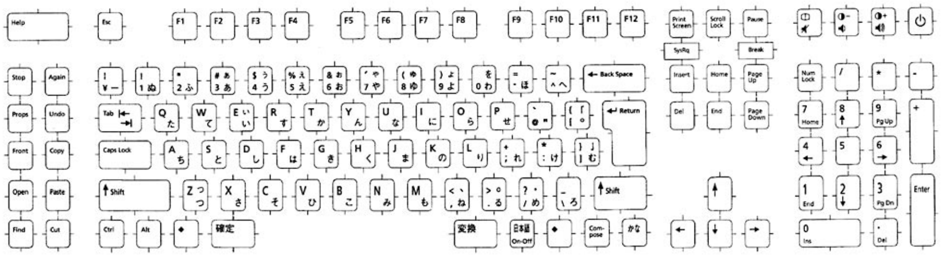
The following figure shows the Italian keyboard.

FIGURE 3–8 Italian Keyboard



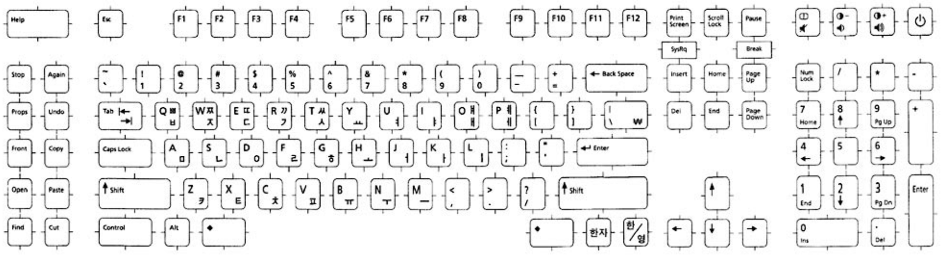
The following figure shows the Japanese keyboard,

FIGURE 3–9 Japanese Keyboard



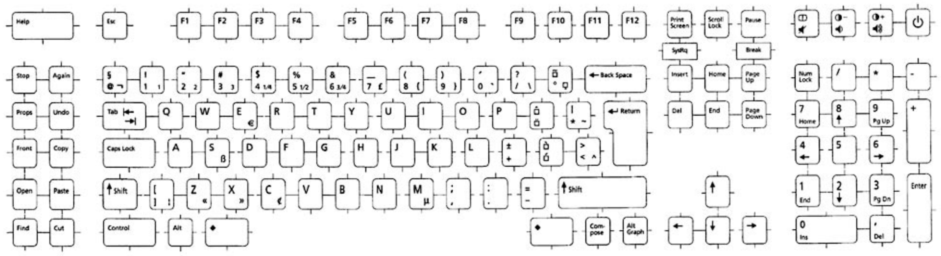
The following shows the Korean keyboard,

FIGURE 3–10 Korean Keyboard



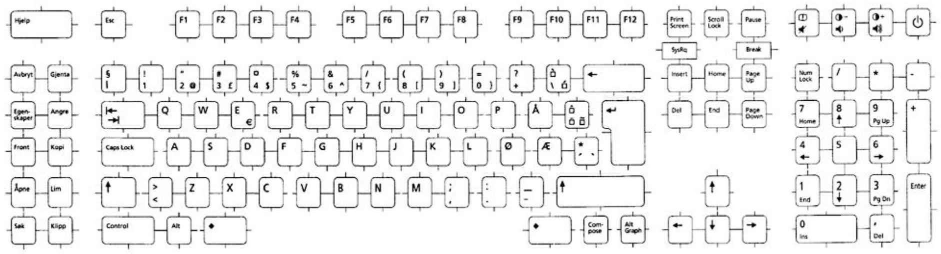
The following shows the Netherlands (Dutch) keyboard,

FIGURE 3–11 Netherlands (Dutch) Keyboard



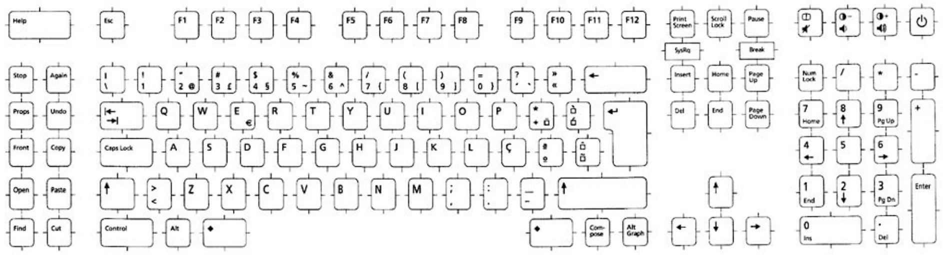
The following figure shows the Norwegian keyboard.

FIGURE 3–12 Norwegian Keyboard



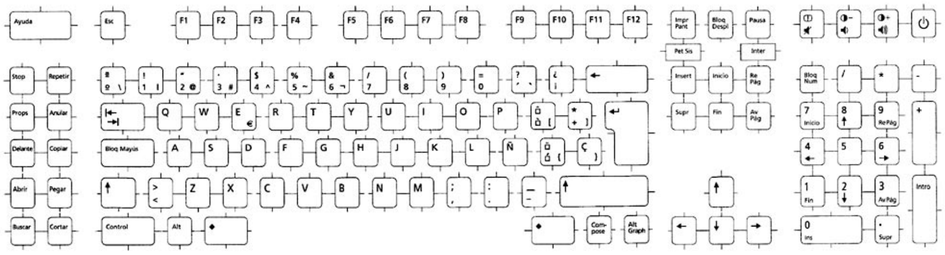
The following figure shows the Portuguese keyboard.

FIGURE 3–13 Portuguese Keyboard



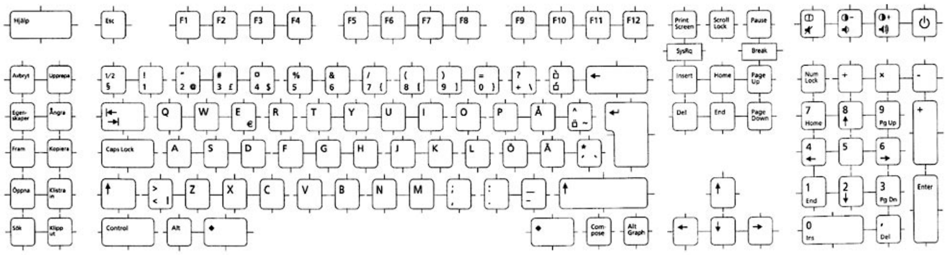
The following figure shows the Spanish keyboard.

FIGURE 3–14 Spanish Keyboard



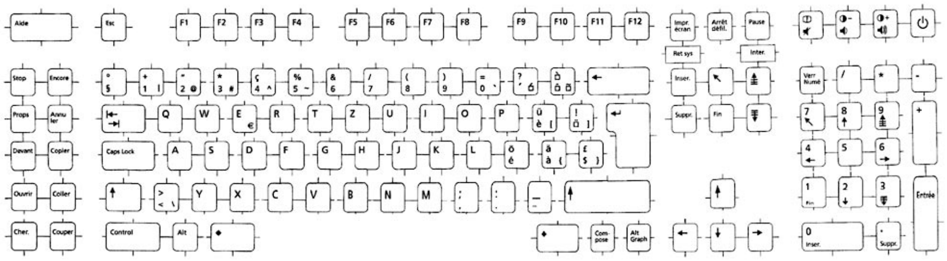
The following figure shows the Swedish keyboard.

FIGURE 3–15 Swedish Keyboard



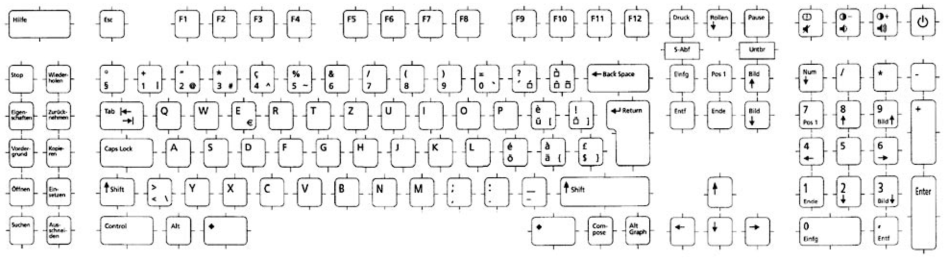
The following figure shows Swiss (French) keyboard.

FIGURE 3–16 Swiss (French) Keyboard



The following figure shows the Swiss (German) keyboard.

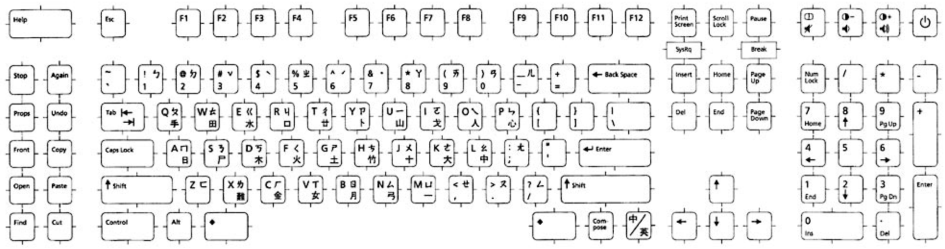
FIGURE 3–17 Swiss (German) Keyboard



The following figure shows the Thai Pattachote keyboard.

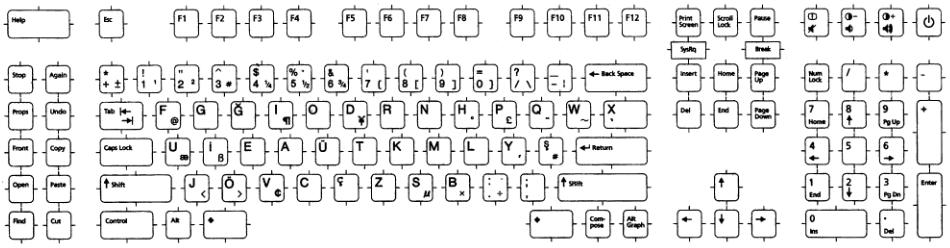
The following figure shows the Traditional Chinese keyboard.

FIGURE 3–18 Traditional Chinese Keyboard



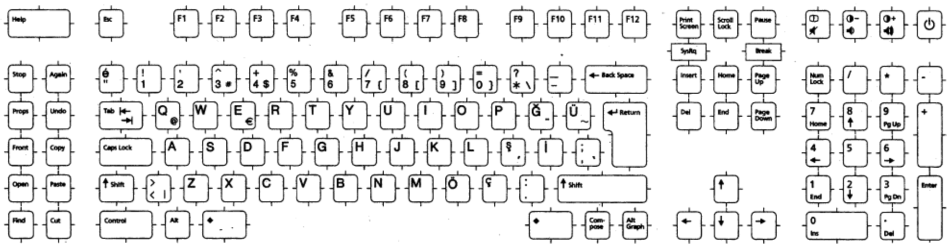
The following figure shows the Turkish F keyboard.

FIGURE 3–19 Turkish F Keyboard



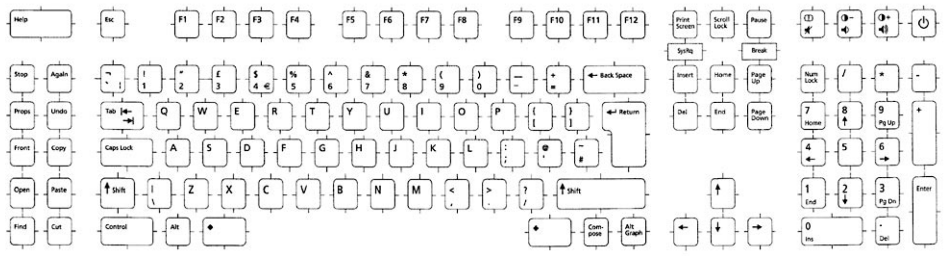
The following figure shows the Turkish Q keyboard.

FIGURE 3–20 Turkish Q Keyboard



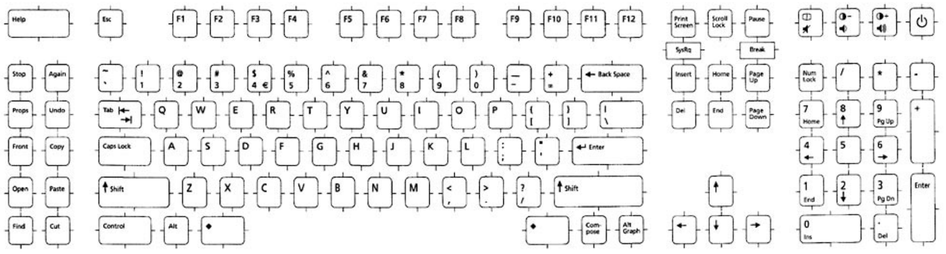
The following figure shows the United Kingdom keyboard.

FIGURE 3–21 United Kingdom Keyboard



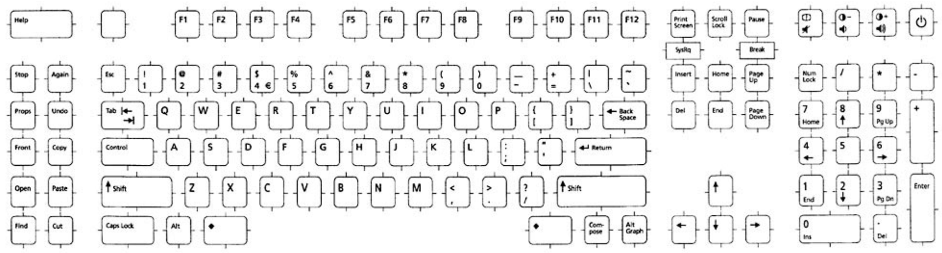
The following figure shows the United States keyboard.

FIGURE 3–22 United States Keyboard



The following figure shows the U.S.A./UNIX keyboard.

FIGURE 3–23 U.S.A./UNIX Keyboard



New Oracle Solaris Keyboard Software Support

Software support for the following additional keyboards is available in this release:

- Russian Type 6 USB keyboard
- Estonian Type 6 USB keyboard
- French Canadian Type 6 USB keyboard
- Polish programmer's Type 5 keyboard

The software enables users in Russia, Canada, Estonia, and Poland to modify the standard U.S. keyboard layouts to meet individual language needs. Currently, no hardware is available for the additional keyboard types. To take advantage of this new keyboard software, follow the steps in the procedures in this section.

▼ How to Access Estonian Type 6 USB Keyboard Support

1 **Change the `US6.kt` entry to `Estonia6.kt` in the `/usr/openwin/share/etc/keytables/keytable.map` file.**

The modified entry should appear as follows:

```
6               0       Estonia6.kt
```

2 **Add one of the following entries to the `/usr/openwin/share/lib/locale/iso_8859_15/Compose` file.**

The modified entry should appear as follows:

```
<scaron>     : "/xa8"   scaron
<scaron>     : "/xa6"   scaron
<scaron>     : "/270"   scaron
<scaron>     : "/264"   scaron
```

3 **Reboot the system to implement the changes.**

▼ How to Access French Canadian Type 6 USB Keyboard Support

1 **Change the `US6.kt` entry to `Canada6.kt` in the `/usr/openwin/share/etc/keytables/keytable.map` file.**

The modified entry should appear as follows:

```
6               0       Canada6.kt
```

2 **Reboot the system to implement the changes.**

▼ How to Access Polish Programmers Type 5 Keyboard Support

1 **Change the `Poland5.kt` entry to `Poland5_pr.kt` in the `/usr/openwin/share/etc/keytables/keytable.map` file.**

The modified entry should appear as follows:

```
6               0       Poland5_pr.kt
```

2 **Reboot the system to implement the changes.**

CHAPTER 4

Supported Asian Locales

This chapter provides information on localization related information for the Japanese, Indic, and Thai languages. The sections in this chapter are:

- "Japanese Localization" on page 83
- "Indic Localization" on page 86
- "Thai Localization" on page 114

For Korean and Chinese locale support, see the following documents.

- *Oracle Solaris 10 User's Guide - Korean*
- *Oracle Solaris 10 User's Guide - Simplified Chinese*
- *Oracle Solaris 10 User's Guide - Traditional Chinese*

Japanese Localization

This section describes Japanese locale-specific information. For more information, see the documents in Oracle Solaris 10 User Collection - Japanese (written in Japanese).

Japanese Locales

Four Japanese locales, which support different character encodings, are available in the current Oracle Solaris environment. The `ja` and `ja_JP.eucJP` locales are based on the Japanese EUC. The `ja_JP.eucJP` locale conforms to the UI-OSF Japanese Environment Implementation Agreement Version 1.1 and the `ja` locale conforms to the traditional specification from earlier Oracle Solaris releases. The `ja_JP.PCK` locale is based on PC-Kanji code (known as Shift_JIS) and the `ja_JP.UTF-8` is based on `UTF-8`.

See the `eucJP`(5) man page for a map showing Japanese EUC and the character set. See the `PCK`(5) man page for the map showing PC-Kanji code and the character set.

Japanese Character Sets

The supported Japanese character sets include:

- JIS X 0201–1976
- JIS X 0208–1990
- JIS X 0212–1990
- JIS X 0213:2004 (only characters defined in Unicode4.0)

JIS X 0212–1990 is not supported in the `ja_JP.PCK` locale. JIS X 0213:2004 is supported in the `ja_JP.UTF-8` locale only. Not all characters defined in the JIS X 0213:2004 are available. Only those characters defined in the Unicode 4.0 character set are available.

Vendor-defined characters (VDC) and user-defined characters (UDC) are also supported. VDCs occupy unused (reserved) code points of JIS X 0208–1990 or JIS X 0212–1990. UDCs occupy the same code points as VDCs, except those code points allocated for VDCs.

Japanese Fonts

Three Japanese font formats are supported: bitmap, TrueType, and Type1. The Japanese Type1 font includes only JIS X 0212 for printing. The Type1 font is also used by UDC.

Japanese bitmap fonts are described in the following table.

TABLE 4–1 Japanese Bitmap Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
`sun gothic`	R, B	PCF(12,14,16,20,24)		JIS X 0208–1983, JIS X 0201–1976
`sun minchou`	R	PCF(12,14,16,20,24)		JIS X 0208–1983, JIS X 0201–1976
`ricoh hg gothic b`	R	PCF(10,12,14,16,18,20,24)	RICOH	JIS X 0208–1983, JIS X 0201–1976
`ricoh hg mincho l`	R	PCF(10,12,14,16,18,20,24)	RICOH	JIS X 0208–1983, JIS X 0201–1976
`ricoh gothic`	R	PCF(10,12,14,16,18,20,24)	RICOH	JIS X 0212–1990, JIS X 0213:2004
`ricoh mincho`	R	PCF(10,12,14,16,18,20,24)	RICOH	JIS X 0212–1990, JIS X 0213:2004
`ricoh heiseimin`	R	PCF(12,14,16,18,20,24)	RICOH	JIS X 0212–1990

Japanese TrueType fonts are described in the following table.

TABLE 4–2 Japanese TrueType Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
`ricoh hg gothic b`	Fixed	TrueType	RICOH	JIS X 0208–1983, JIS X 0201–1976
`ricoh hg mincho l`	Fixed	TrueType	RICOH	JIS X 0208–.1983, JIS X 0201–1976
`ricoh hg gothicb sun`	Fixed, Proportional	TrueType	RICOH	JIS X 0201–176, JIS X 0208–1983, JIS X 0213–2004
`ricoh hg minchol sun`	Fixed, Proportional	TrueType	RICOH	JIS X 0201–1976, JIS X 0208–1983, JIS X 0213–2004
`ricoh heiseimin`	Fixed	TrueType	RICOH	JIS X 0212–1990

Japanese Input Systems

ATOK for Solaris (equivalent to ATOK17) is the default Japanese input system. Wnn6 is also available. To switch ATOK for Solaris to Wnn6, see Japanese Environment User's Guide (written in Japanese).The `kkcv` Japanese input system is available for Japanese Solaris 1.x BCP support.

Terminal Setting for Japanese Terminals

To use Japanese locales on a character-based terminal (TTY) you must use terminal settings to make line editing work correctly.

- If your terminal is a CDE Terminal emulator (`dtterm`), use `stty`(1) with the argument `-defeucw` in any Japanese locale (`ja`, `ja_JP.PCK`, or `ja_JP.UTF-8`). For example, in the `ja` locale you would type:

  ```
  % setenv LANG ja
  % stty defeucw
  ```
- If your terminal is not a CDE Terminal emulator but the code set of your terminal is the same as that of the current locale, use `stty`(1) with the argument `-defeucw`.
- If your terminal's code set doesn't match that of the current locale, use `setterm`(1) to enable code conversion. For example, if you are in the `ja` locale but your terminal requires PCK (Shift_JIS code), you would type:

  ```
  % setenv LANG ja
  % setterm -x PCK
  ```

  See the `setterm`(3CURSES) man page for details.

Japanese iconv Module

Several Japanese code set conversions are supported with iconv(1) and iconv(3C). See the iconv_ja(5) man page for details.

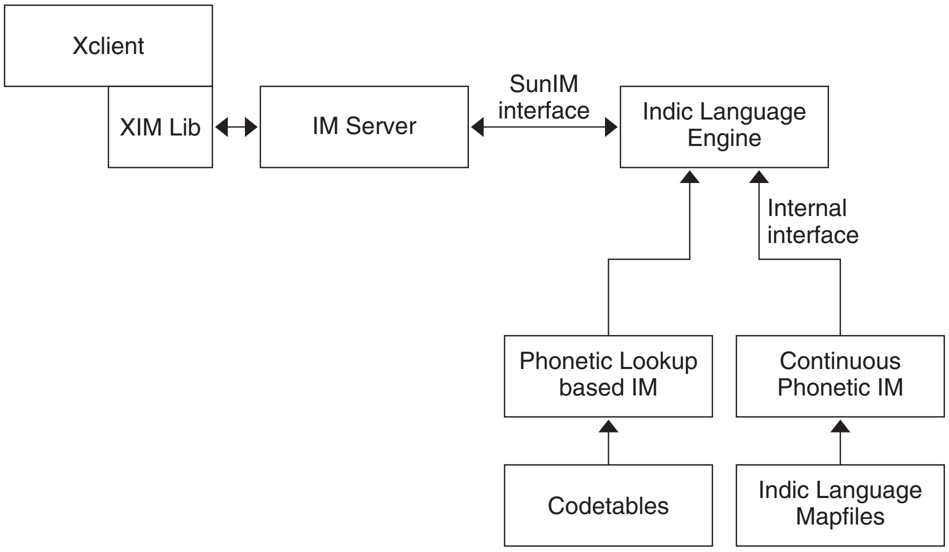
User-Defined Character Support

The user-defined character utility sdtudctool handles both outline (Type1) and bitmap (PCF) fonts. Some utilities are also available to migrate the UDC fonts that were created by old utilities in prior releases, such as fontedit, type3creator, and fontmanager.

Indic Localization

Phonetic lookup based input method (Shabdalipi) and continuous phonetic input method are available for all Indic languages which are supported in the UTF-8 locale. The input methods and virtual keyboards allow you to enter Indic text in all of the CDE applications.

The following data flow illustrates the workings of the Indic input process.



▼ How to Use the Indic Input Methods

1. **Click the input status area to display the input method selection menu.**
2. **Select an input method from the menu.**

   Alternatively, you can press the F6 key to select from among the available input methods.

   You can also type the Compose-hi key sequence to select the input method that you used previously.
3. **Press the F5 key to select the Indic script you want to use.**

   a. **For the keyboard-based (indic INSCRIPT keyboard) input method, use the keyboard images shown in "Indic Keyboards" on page 87.**

   b. **For the phonetic lookup-based input method, type the first English phonetic equivalent character corresponding to the character in the target script.**

      Select from a list of choices displayed in the lookup window.

   c. **For the continuous phonetic input method, type in English phonetic equivalents continuously.**

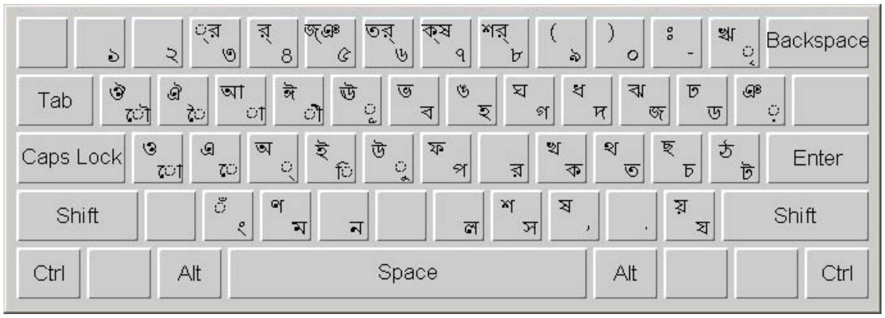
      The corresponding characters in the target script are displayed in the preedit and will be committed when subsequent input makes the preedit text unambiguous or by an explicit commit. Refer to figures given in "Mapping for the Continuous Phonetic Based Input Method" on page 91 for illustrations of the mapping from the English tokens to the UTF-8 codepoints of the target script for the continuous phonetic input method.
4. **Press Control-spacebar to switch back to English/European input mode.**

   Alternatively, click in the status area to select the English/European input mode from the input mode selection window.

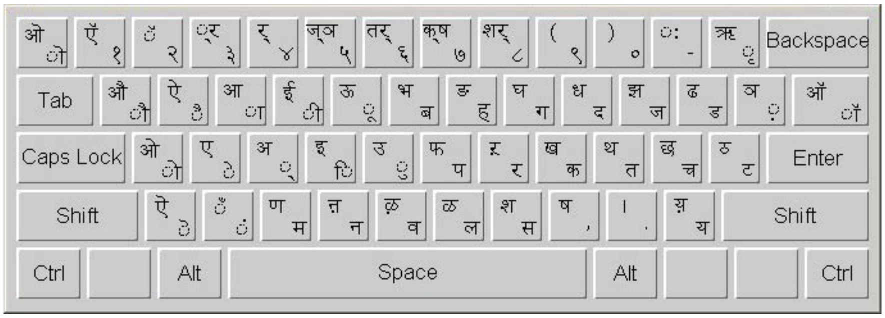
Indic Keyboards

The following figures show the keyboard layouts that are available for the Indic input method.

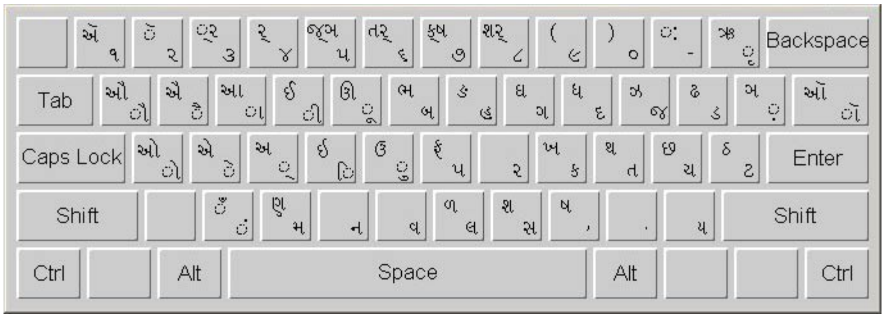
The following figure shows the layout of the Bengali keyboard.



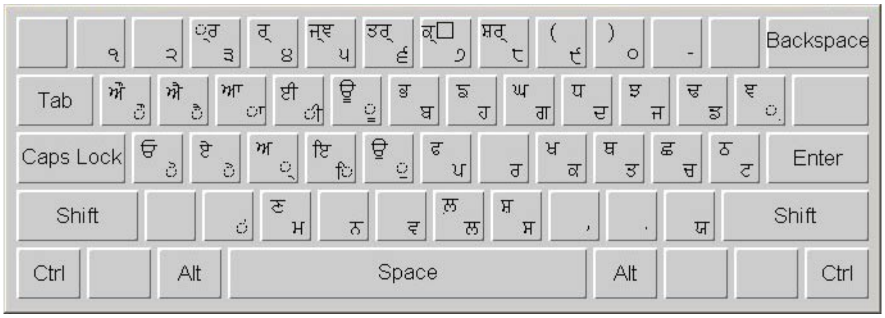
The following figure shows the layout of the Devanagari keyboard.



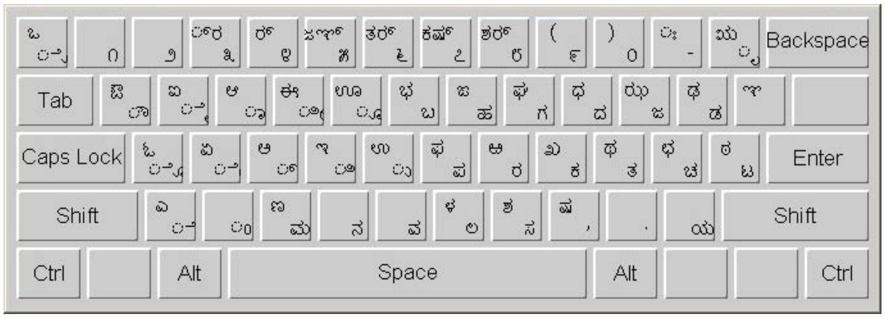
The following figure shows the layout of the Gujarati keyboard.



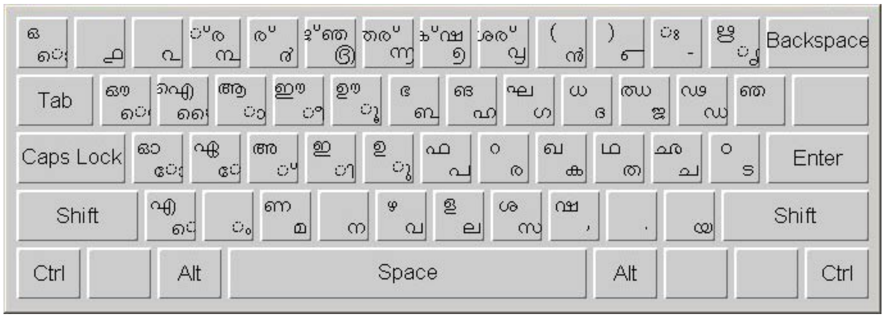
The following figure shows the layout of the Gurmukhi keyboard.



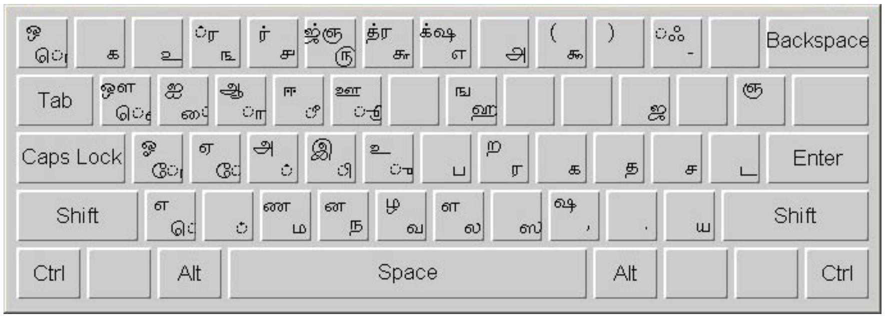
The following figure shows the layout of the Kannada keyboard.



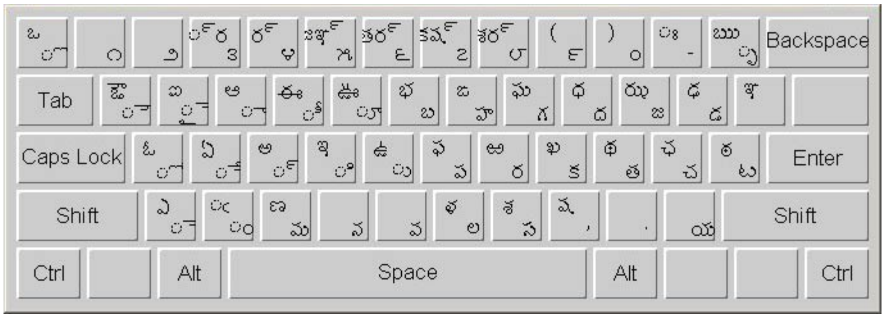
The following figure shows the layout of the Malayalam keyboard.



The following figure shows the layout of the Tamil keyboard.



The following figure shows the layout of the Teluga keyboard.



Understanding the Mappings

The images in “Mapping for the Continuous Phonetic Based Input Method” on page 91 show the mappings between English tokens and their equivalent codepoints in each of the target scripts supported. The CONSONANT category means the mapping is between the English tokens and consonants of the script. The VOWEL category means that mapping from English tokens and vowels of the script. The OTHER category includes mapping of characters that do not exhibit the properties of consonants and vowels (whose form does not change depending on the surrounding character).

The keywords CONSONANT, VOWEL and OTHER also mean that these characters are part of Unicode standard. The section SPECIAL CONSONANT, SPECIAL VOWEL or SPECIAL OTHER means that though in principle these characters display the properties of consonants, vowels or others they are not officially part of the Unicode standard and are font dependent. They are assigned codepoint values in Unicode Private User Area. They are supported in Oracle Solaris UTF-8 locales and the mapping may not work in a different platform.

These mapfiles are not the same as the ones in your system, but slightly edited ones for removing unneeded keywords for the context of this discussion.

In the VOWELS and SPECIAL VOWELS section, an independent form and a dependent form is displayed for the same English token depending on the context. See "How the Continuous Phonetic Input Method Works" on page 113.

The Malayalam script contains a special 'CHILLU' section, that is actually the SPECIAL OTHER category.

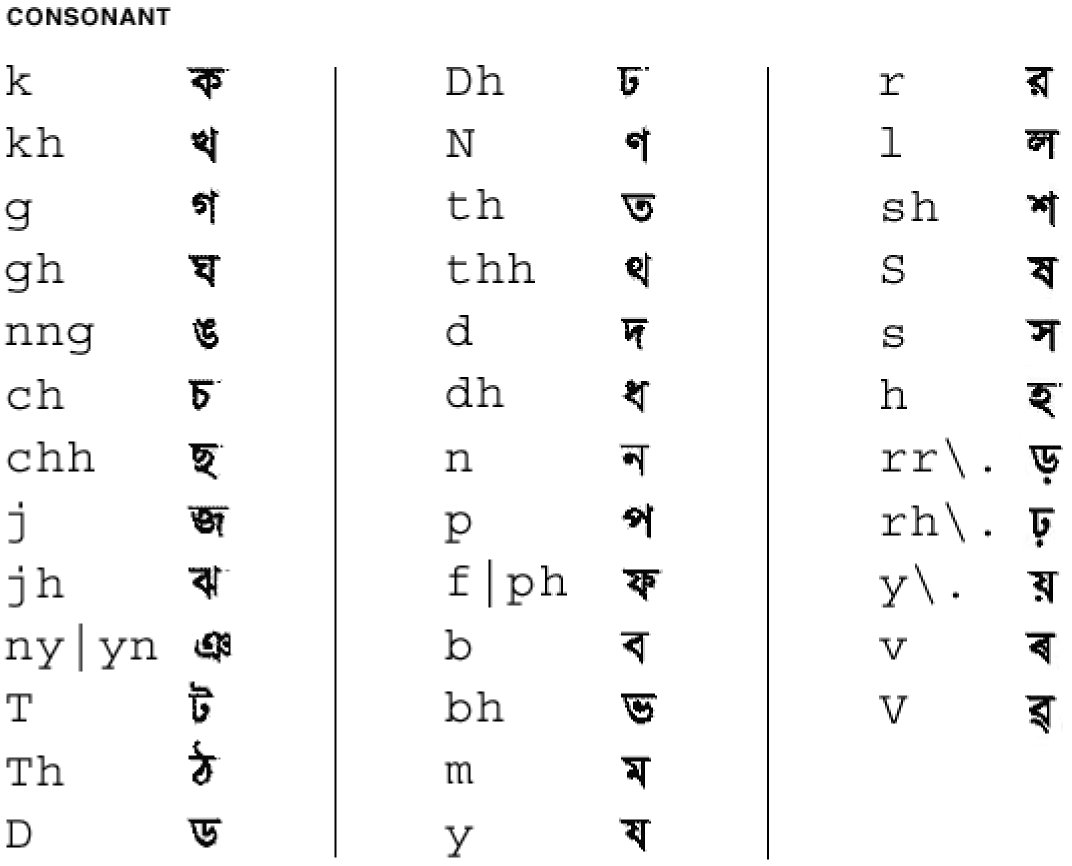
Mapping for the Continuous Phonetic Based Input Method

The following figures show the existing mappings from English to the phonetic equivalent characters in the target Indic scripts. Use these illustrations as a reference until you know all the mappings for the script that you use. Mappings given here are intuitive, so you should be able to input most of the characters without looking up the illustration.

Note – In these mappings, special characters such as '.' and '|' included as part of the mapping are escaped with a '\' character. If not escaped, the '|' character acts as a separator when more than one token represents the same UTF-8 character.

Figure 4–1, Figure 4–2, and Figure 4–3 show the English to Bengali mappings for consonants, vowels, and others.

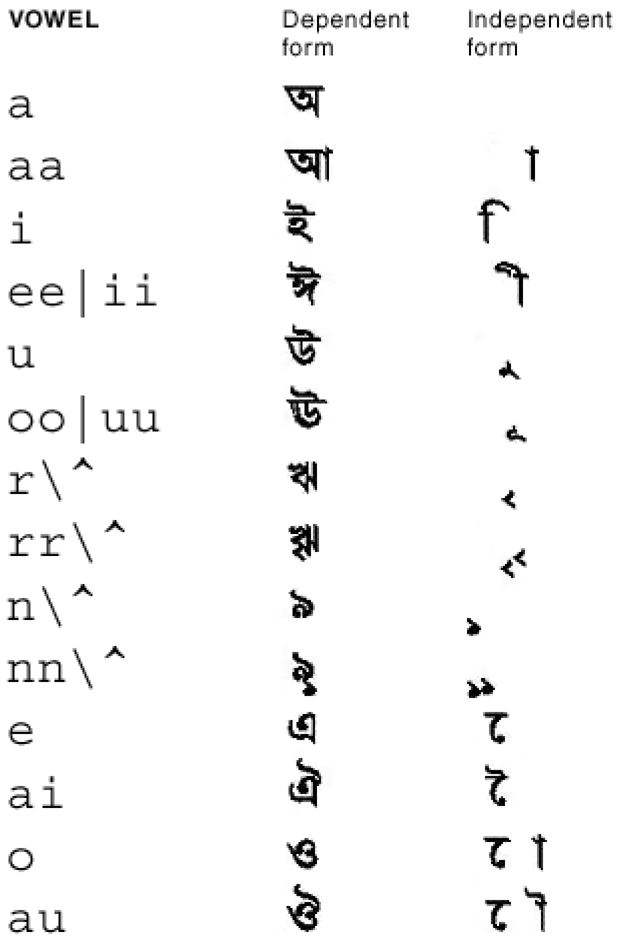
FIGURE 4–1 Map for Bengali Consonants



CONSONANT

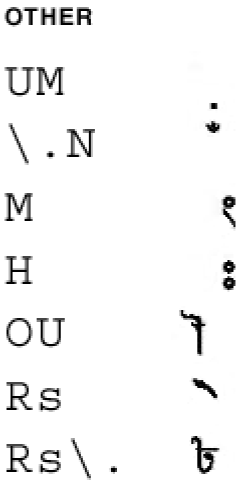
k	ক	Dh	ঢ	r	র
kh	খ	N	ণ	l	ল
g	গ	th	ত	sh	শ
gh	ঘ	thh	থ	S	ষ
nng	ঙ	d	দ	s	স
ch	চ	dh	ধ	h	হ
chh	ছ	n	ন	rr\.	ড়
j	জ	p	প	rh\.	ঢ়
jh	ঝ	f|ph	ফ	y\.	য়
ny|yn	ঞ	b	ব	v	ব
T	ট	bh	ভ	V	ব্
Th	ঠ	m	ম		
D	ড	y	য		

FIGURE 4–2 Map for Bengali Vowels



VOWEL	Dependent form	Independent form
a	অ	
aa	আ	া
i	ই	ি
ee|ii	ঈ	ী
u	উ	ু
oo|uu	ঊ	ূ
r\\^	ঋ	ৃ
rr\\^	ৠ	ৄ
n\\^	ঌ	ৢ
nn\\^	ৡ	ৣ
e	এ	ে
ai	ঐ	ৈ
o	ও	ে া
au	ঔ	ে ৗ

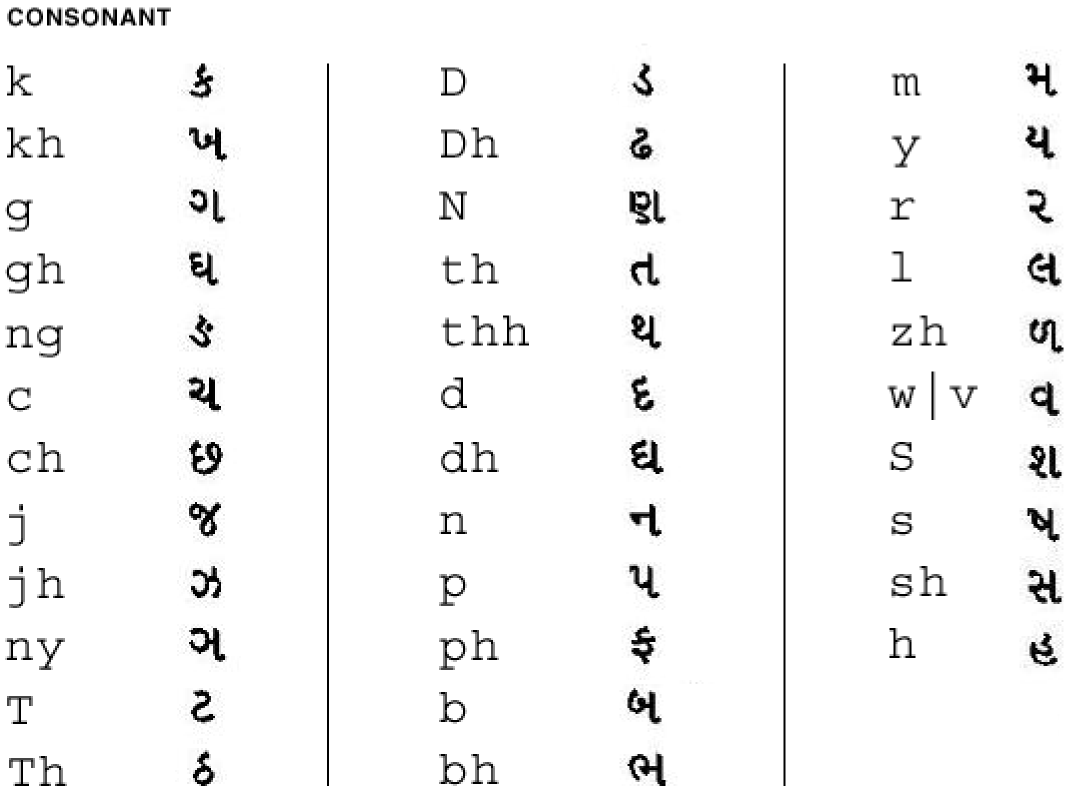
FIGURE 4–3 Map for Bengali Others



OTHER	
UM	
\ .N	ঁ
M	ং
H	ঃ
OU	ৗ
Rs	্
Rs\ .	৳

Figure 4–4, Figure 4–5, and Figure 4–6 show the English to Gujarati mappings for consonants, vowels, and others.

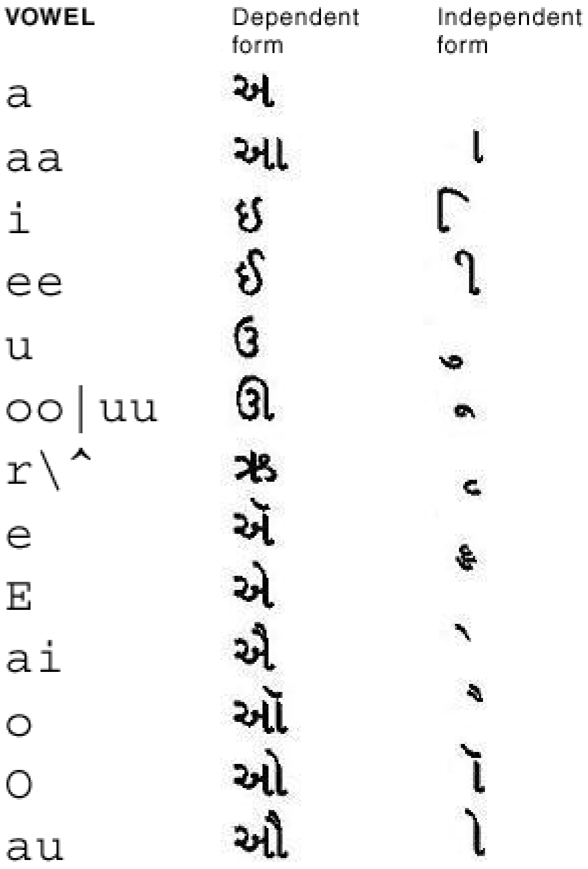
FIGURE 4–4 Map for Gujarati Consonants



CONSONANT

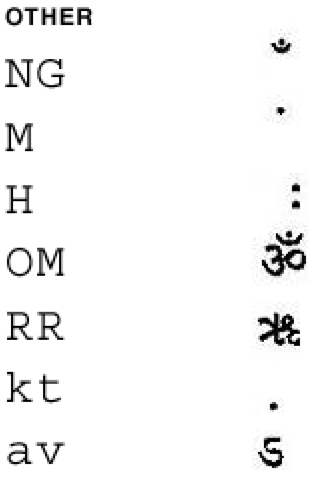
k	ક	D	ડ	m	મ
kh	ખ	Dh	ઢ	y	ય
g	ગ	N	ણ	r	ર
gh	ઘ	th	ત	l	લ
ng	ઙ	thh	થ	zh	ળ
c	ચ	d	દ	w|v	વ
ch	છ	dh	ધ	S	શ
j	જ	n	ન	s	ષ
jh	ઝ	p	પ	sh	સ
ny	ઞ	ph	ફ	h	હ
T	ટ	b	બ		
Th	ઠ	bh	ભ		

FIGURE 4–5 Map for Gujarati Vowels



VOWEL	Dependent form	Independent form
a	અ	
aa	આ	ા
i	ઇ	િ
ee	ઈ	ી
u	ઉ	ુ
oo|uu	ઊ	ૂ
r\^	ઋ	ૃ
e	ઍ	ૅ
E	એ	ે
ai	ઐ	ૈ
o	ઑ	
O	ઓ	ો
au	ઔ	ૌ

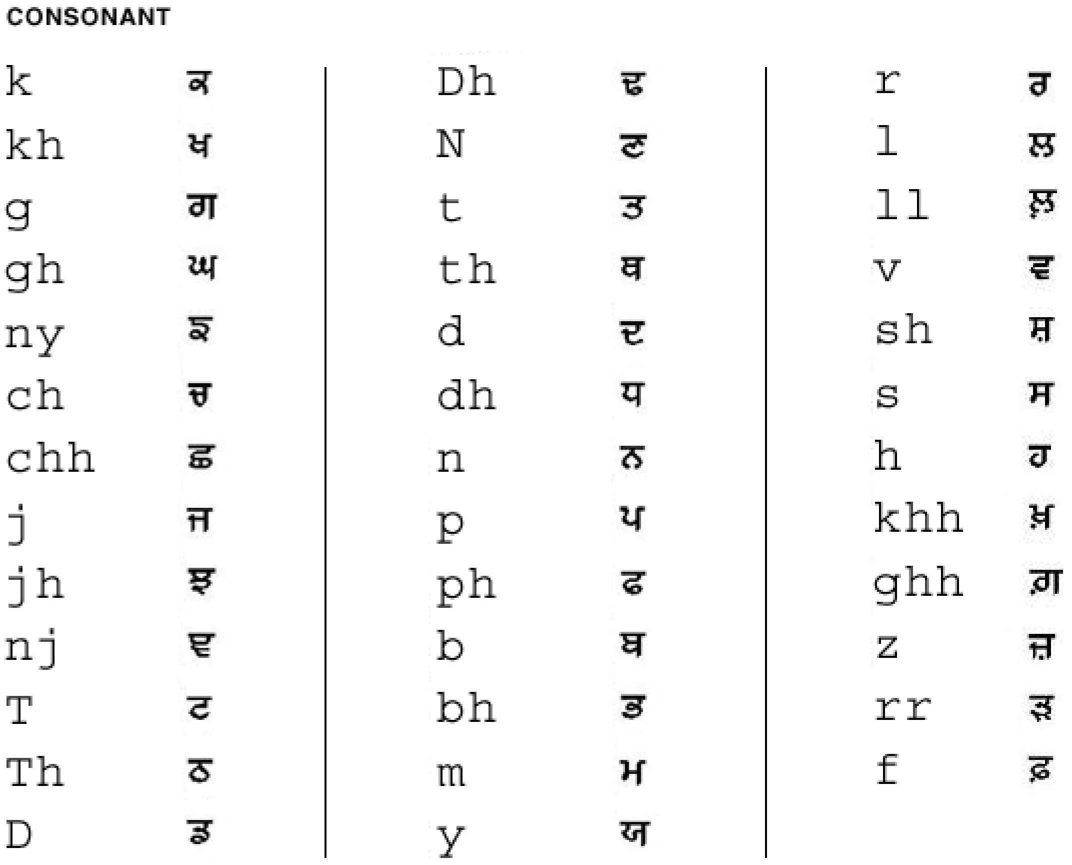
FIGURE 4–6 Map for Gujarati Others



OTHER	
NG	ઁ
M	ં
H	ઃ
OM	ૐ
RR	ૠ
kt	઼
av	ઽ

Figure 4–7, Figure 4–8, and Figure 4–9 show the English to Gurmukhi mappings for consonants, vowels, and others.

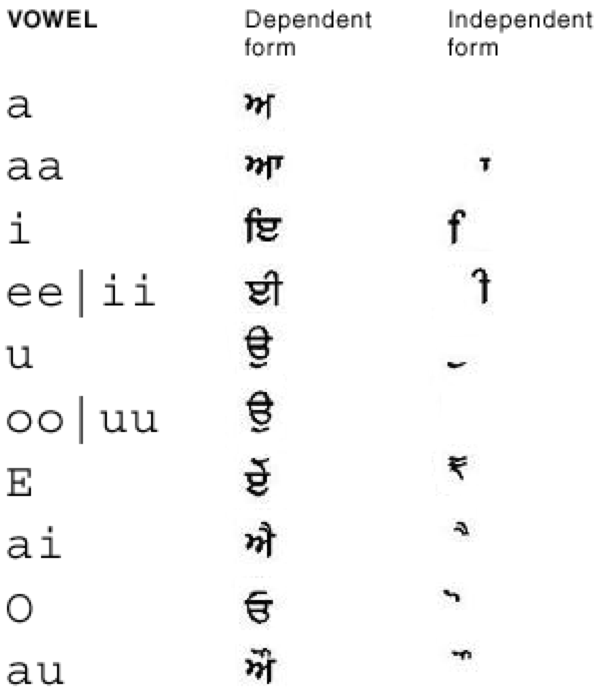
FIGURE 4–7 Map for Gurmukhi Consonants



CONSONANT

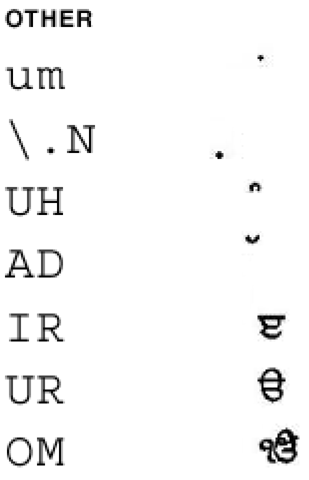
k	ਕ	Dh	ਢ	r	ਰ
kh	ਖ	N	ਣ	l	ਲ
g	ਗ	t	ਤ	ll	ਲ਼
gh	ਘ	th	ਥ	v	ਵ
ny	ਙ	d	ਦ	sh	ਸ਼
ch	ਚ	dh	ਧ	s	ਸ
chh	ਛ	n	ਨ	h	ਹ
j	ਜ	p	ਪ	khh	ਖ਼
jh	ਝ	ph	ਫ	ghh	ਗ਼
nj	ਞ	b	ਬ	z	ਜ਼
T	ਟ	bh	ਭ	rr	ੜ
Th	ਠ	m	ਮ	f	ਫ਼
D	ਡ	y	ਯ		

FIGURE 4–8 Map for Gurmukhi Vowels



VOWEL	Dependent form	Independent form
a	ਅ	
aa	ਆ	ਾ
i	ਇ	ਿ
ee | ii	ਈ	ੀ
u	ਉ	ੁ
oo | uu	ਊ	
E	ਏ	ੇ
ai	ਐ	ੈ
O	ਓ	ੋ
au	ਔ	ੌ

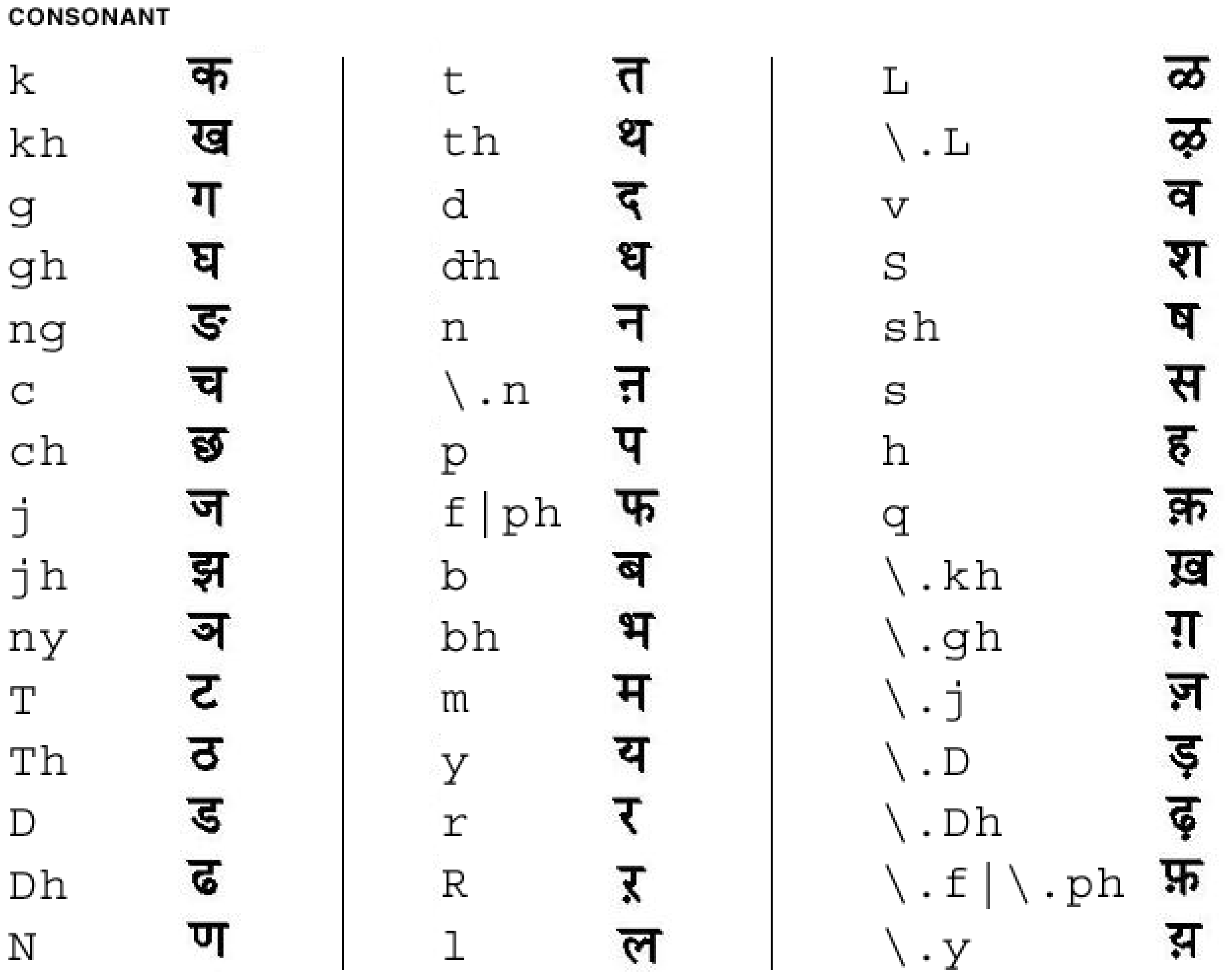
FIGURE 4–9 Map for Gurmukhi Others



OTHER	
um	ਂ
\\.N	.
UH	ੰ
AD	ੱ
IR	ੲ
UR	ੳ
OM	ੴ

Figure 4–10, Figure 4–11, and Figure 4–12 show the English to Hindi mappings for consonants, vowels, and others.

FIGURE 4–10 Map for Hindi Consonants



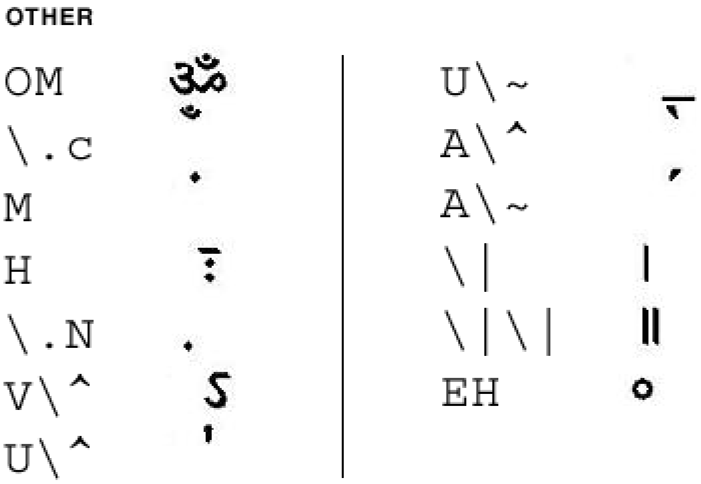
CONSONANT

k	क	t	त	L	ळ
kh	ख	th	थ	\\.L	ऴ
g	ग	d	द	v	व
gh	घ	dh	ध	S	श
ng	ङ	n	न	sh	ष
c	च	\\.n	ऩ	s	स
ch	छ	p	प	h	ह
j	ज	f|ph	फ	q	क़
jh	झ	b	ब	\\.kh	ख़
ny	ञ	bh	भ	\\.gh	ग़
T	ट	m	म	\\.j	ज़
Th	ठ	y	य	\\.D	ड़
D	ड	r	र	\\.Dh	ढ़
Dh	ढ	R	ऱ	\\.f|\\.ph	फ़
N	ण	l	ल	\\.y	य़

FIGURE 4–11 Map for Hindi Vowels

VOWEL	Dependent form	Independent form
a	अ	
aa	आ	ा
i	इ	ि
ee	ई	ी
u	उ	ु
oo	ऊ	ू
r\^	ऋ	ृ
rr\^	ॠ	ॄ
l\^	ऌ	ॢ
ll\^	ॡ	ॣ
EE	ऍ	ॅ
E	ऎ	ॆ
e	ए	े
ai	ऐ	ै
OO	ऑ	ॉ
O	ऒ	ॊ
o	ओ	ो
au	औ	ौ

FIGURE 4–12 Map for Hindi Others

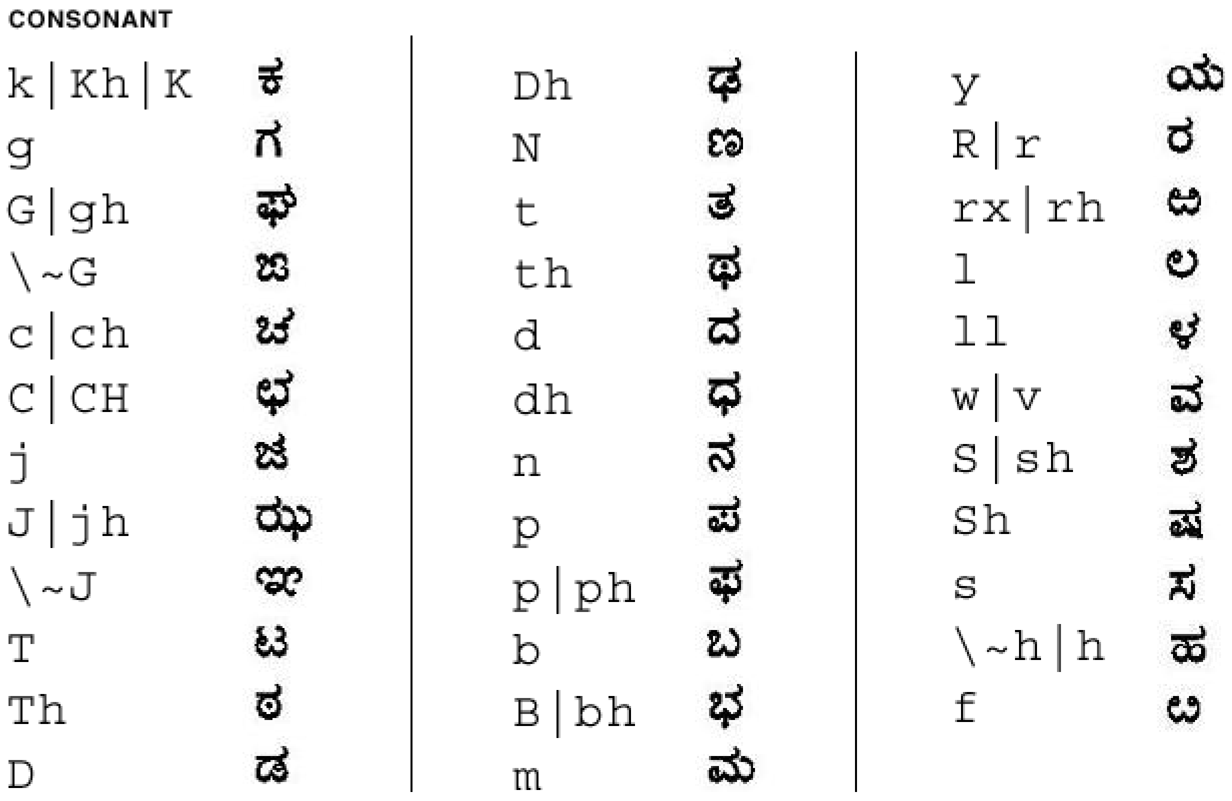


OTHER

OM	ॐ	U\~	॒
\.c	ँ	A\^	॓
M	ं	A\~	॔
H	ः	|	।
\.N	़	||	॥
V\^	ऽ	EH	॰
U\^	॑		

Figure 4–13, Figure 4–14, and Figure 4–15 show the English to Kannada mappings for consonants, vowels, and others.

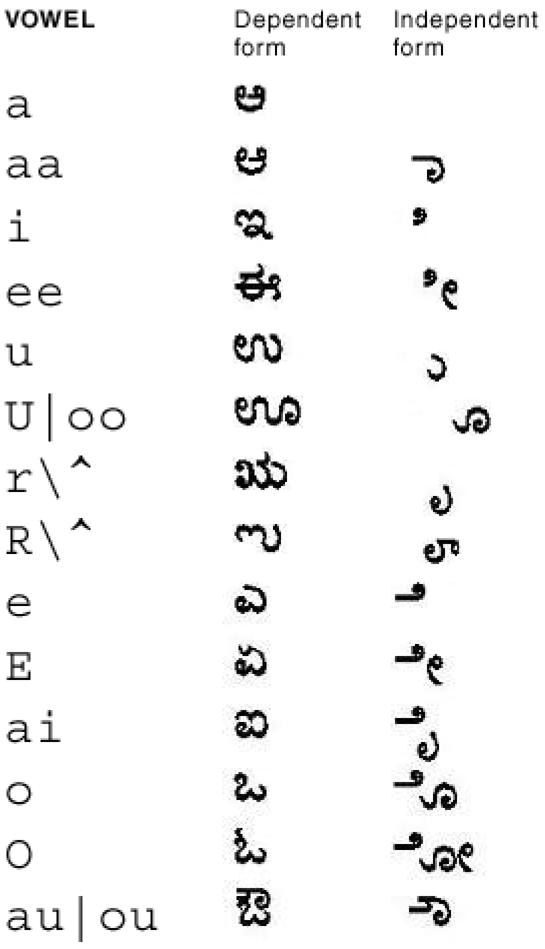
FIGURE 4–13 Map for Kannada Consonants



CONSONANT

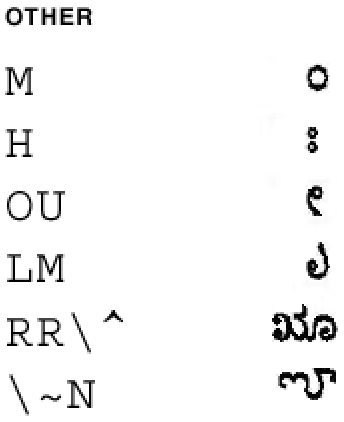
k | Kh | K	ಕ	Dh	ಢ	y	ಯ
g	ಗ	N	ಣ	R | r	ರ
G | gh	ಘ	t	ತ	rx | rh	ಱ
\~G	ಙ	th	ಥ	l	ಲ
c | ch	ಚ	d	ದ	ll	ಳ
C | CH	ಛ	dh	ಧ	w | v	ವ
j	ಜ	n	ನ	S | sh	ಶ
J | jh	ಝ	p	ಪ	Sh	ಷ
\~J	ಞ	p | ph	ಫ	s	ಸ
T	ಟ	b	ಬ	\~h | h	ಹ
Th	ಠ	B | bh	ಭ	f	ೞ
D	ಡ	m	ಮ		

FIGURE 4–14 Map for Kannada Vowels



VOWEL	Dependent form	Independent form
a	ಅ	
aa	ಆ	ಾ
i	ಇ	ಿ
ee	ಈ	ೀ
u	ಉ	ು
U|oo	ಊ	ೂ
r\^	ಋ	ೃ
R\^	ೠ	ೄ
e	ಎ	ೆ
E	ಏ	ೇ
ai	ಐ	ೈ
o	ಒ	ೊ
O	ಓ	ೋ
au|ou	ಔ	ೌ

FIGURE 4–15 Map for Kannada Others



OTHER	
M	ಂ
H	ಃ
OU	ೌ
LM	ಌ
RR\^	ೠ
\~N	ಞ

Figure 4–16, Figure 4–17, and Figure 4–18 show the English to Malayalam mappings for consonants, vowels, and others.

FIGURE 4–16 Map for Malayalam Consonants

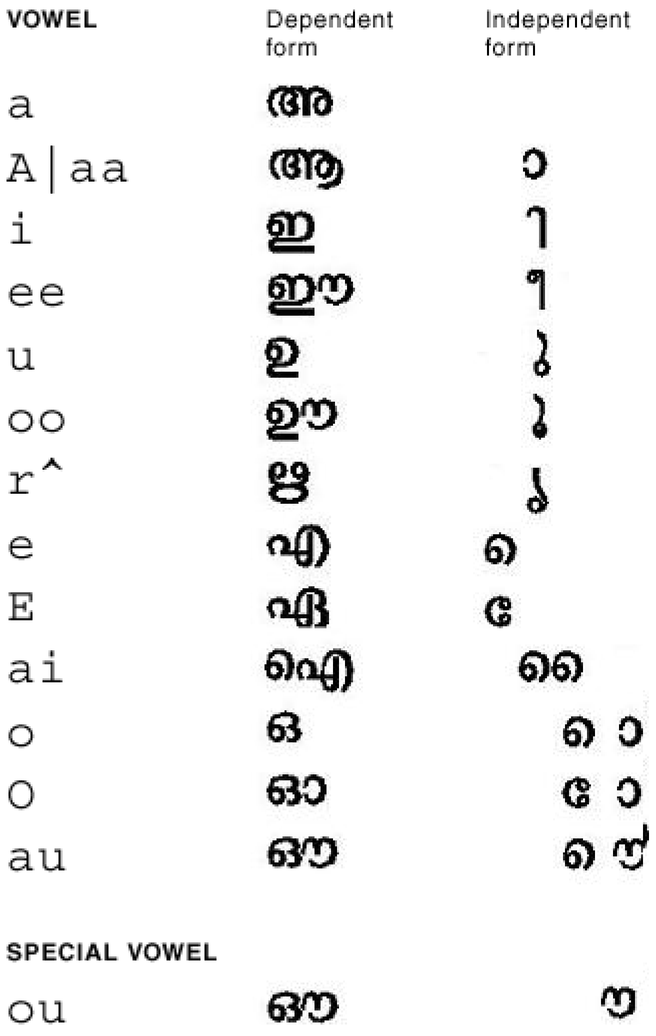
CONSONANT

k	ക	th	ത	zh	ഴ
kh	ഖ	thh	ഥ	w | v	വ
g	ഗ	d	ദ	S	ശ
gh	ഘ	dh	ധ	sh	ഷ
ng	ങ	n	ന	s	സ
ch	ച	p	പ	h	ഹ
chh	ഛ	f | ph	ഫ		
j	ജ	b	ബ		
jh	ഝ	bh	ഭ		
nj	ഞ	m	മ		
T	ട	y	യ		
Th	ഠ	r	ര		
D	ഡ	R	റ		
Dh	ഢ	l	ല		
N	ണ	L	ള		

SPECIAL CONSONANT

nt	ന്റ
nth	ന്ത
nnj	ഞ്ഞ
nk	ങ്ക
nng	ങ്ങ
t	റ്റ

FIGURE 4–17 Map for Malayalam Vowels

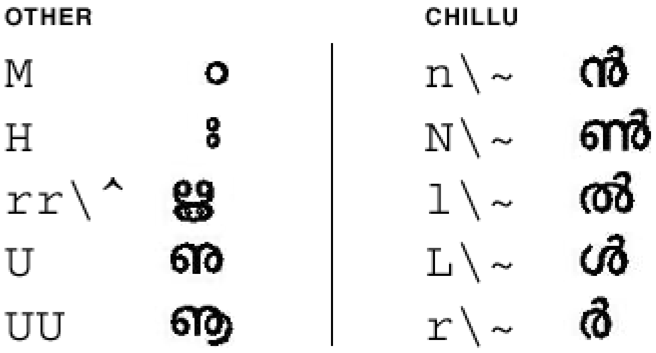


VOWEL	Dependent form	Independent form
a	അ	
A|aa	ആ	ാ
i	ഇ	ി
ee	ഈ	ീ
u	ഉ	ു
oo	ഊ	ൂ
r^	ഋ	ൃ
e	എ	െ
E	ഏ	േ
ai	ഐ	ൈ
o	ഒ	െ ാ
O	ഓ	േ ാ
au	ഔ	െ ൗ

SPECIAL VOWEL

ou	ഔ	ൗ

FIGURE 4–18 Map for Malayalam Others



OTHER		CHILLU	
M	ം	n\~	ൻ
H	ഃ	N\~	ൺ
rr\^	ൠ	l\~	ൽ
U	ഞ	L\~	ൾ
UU	ഞു	r\~	ർ

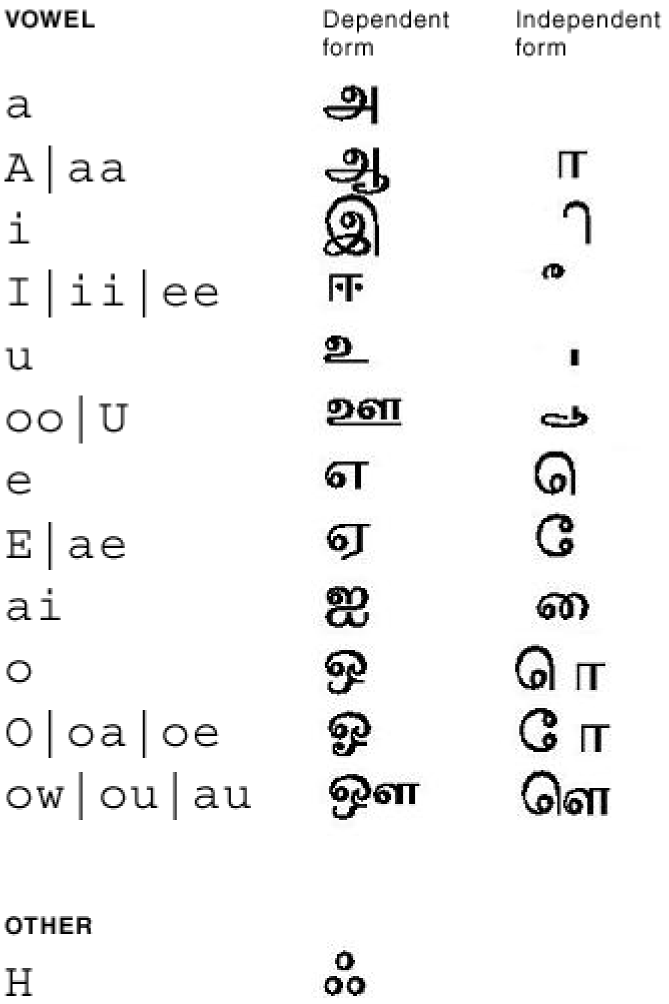
Figure 4–19 and Figure 4–20 show the English to Tamil mappings for consonants and vowels.

FIGURE 4–19 Map for Tamil Consonants

CONSONANT

k | g | K | G	க	R	ற
nG | ng	ங	TR	ற்ற
ch | CH	ச	DR	ற
j | J	ஜ	l	ல
gn | Gn	ஞ	L	ள
t | d	ட	zh | ZH	ழ
N	ண	w | v | W | V	வ
th | dh | TH | DH	த	S	ஷ
n	ந	s	ஸ
n\^	ன	h | H	ஹ
p | b | P | B	ப	ndh	ந்த
m | M	ம	nth	ந்த
y | Y	ய	nj | NJ	ஞ்ச
r	ர	f | ph | F | PH	ஃப

FIGURE 4–20 Map for Tamil Vowels



VOWEL	Dependent form	Independent form
a	அ	
A | aa	ஆ	ா
i	இ	ி
I | ii | ee	ஈ	ீ
u	உ	ு
oo | U	ஊ	ூ
e	எ	ெ
E | ae	ஏ	ே
ai	ஐ	ை
o	ஒ	ெ ா
O | oa | oe	ஓ	ே ா
ow | ou | au	ஔ	ௌ

OTHER		
H	ஃ	

Figure 4–21,Figure 4–22, and Figure 4–23 show the English to Telugu mappings for consonants, vowels, and others.

FIGURE 4–21 Map for Telugu Consonants

CONSONANT

k	క	D	డ	m	మ
K|Kk|kh	ఖ	Dh	ఢ	y	య
g	గ	N|nh	ణ	r	ర
G|Gh|gh	ఘ	t	త	rr	ఱ
\~m	ఙ	th	థ	l	ల
ch|c	చ	d	ద	L	ళ
C|Ch	ఛ	dh	ధ	w|W|V|v	వ
j	జ	n	న	S	శ
J|Jh|jh	ఝ	p	ప	sh	ష
\~n	ఞ	F|f|P|Ph|ph	ఫ	s	స
T	ట	b	బ	h	హ
Th	ఠ	Bh|B|bh	భ		

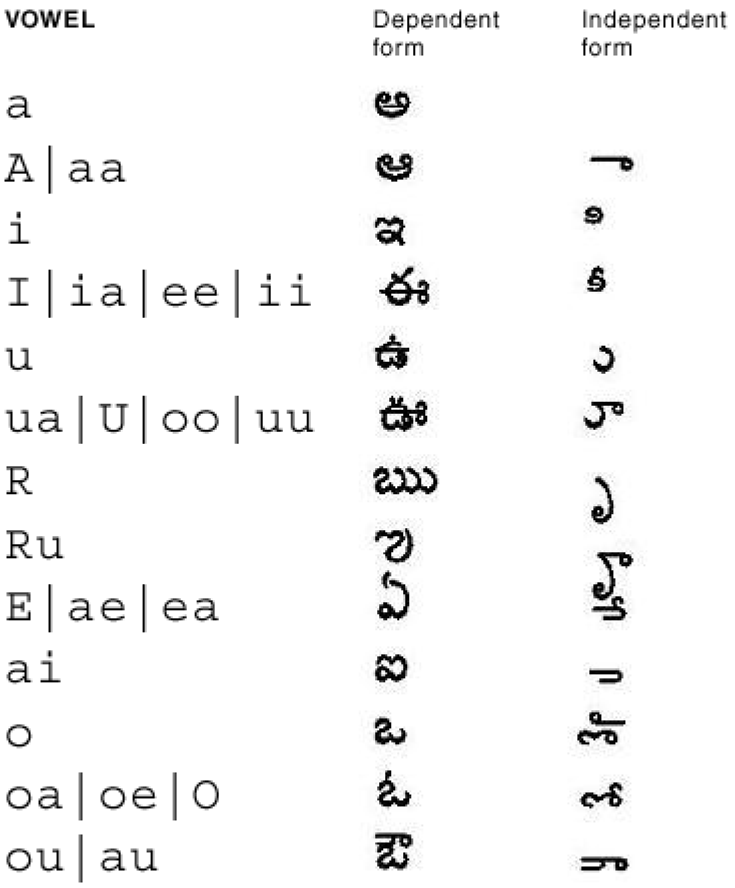


FIGURE 4–22 Map for Telugu Vowels

VOWEL	Dependent form	Independent form
a	అ	
A|aa	ఆ	ా
i	ఇ	ి
I|ia|ee|ii	ఈ	ీ
u	ఉ	ు
ua|U|oo|uu	ఊ	ూ
R	ఋ	ృ
Ru	ౠ	ౄ
E|ae|ea	ఏ	
ai	ఐ	ై
o	ఒ	ొ
oa|oe|O	ఓ	ో
ou|au	ఔ	ౌ

FIGURE 4–23 Map for Telugu Others

OTHER	
\\.N	ఁ
M	ం
H	ః
OU	ౕ
LM	ౖ
RR	ౠ
Nu	ౡ

How the Continuous Phonetic Input Method Works

For each Indic script, a 'virama' or equivalent sign combined with a consonant gives the half form (or ready to combine form) of the consonant. Whenever a multiple key combination corresponding to a consonant is typed, the consonant + virama form is output, symbolizing that the characters are ready to combine.

Consonants, at initial input, will assume their half form and will be a full syllable or their variation when followed by a vowel.

Two consecutive consonants remain as the ready to combine half forms. Half forms can be converted by the layout engine as a single combined character or can remain as those independent forms that are also syntactically valid for every language.

Any vowel that forms the beginning of a word or is followed by another vowel appears in independent form. A vowel that immediately follows a consonant assumes dependent forms.

Characters that do not change shapes in any context are called others. These characters are neither consonants nor vowels.

Digits and other punctuation marks that do not form a part of a character are mapped one to one.

Using these principles, a parser is written that will parse the input into these different categories and output the language-specific Unicode codepoints. The continuous phonetic input method engine does not deal with layout or rendering, which will be done by other modules in the system.

Thai Localization

The current Oracle Solaris environment supports three Thai input levels and four Thai keyboard layouts.

Thai Input Methods

The following Thai input methods are supported in this release. These input methods are specified in the Thai IT Standard for character sequence checking.

1. Passthrough level, no input check
2. Basic input check level
3. Strict input check level

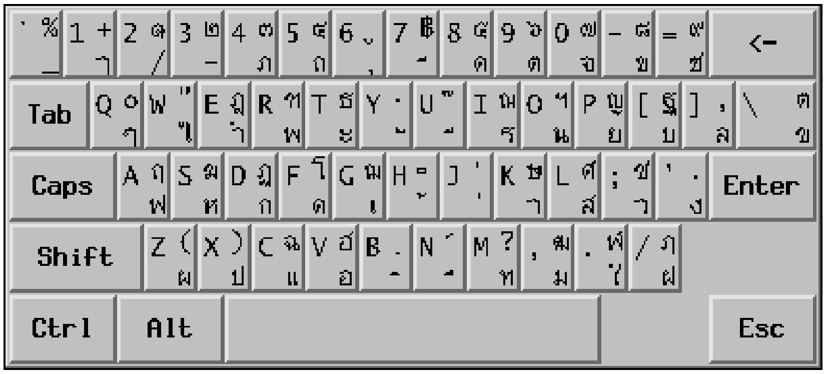
The passthrough level, with no sequence check, is the default in this release as it was in previous Oracle Solaris releases.

You can use the F2 function key to switch from one input level to the next.

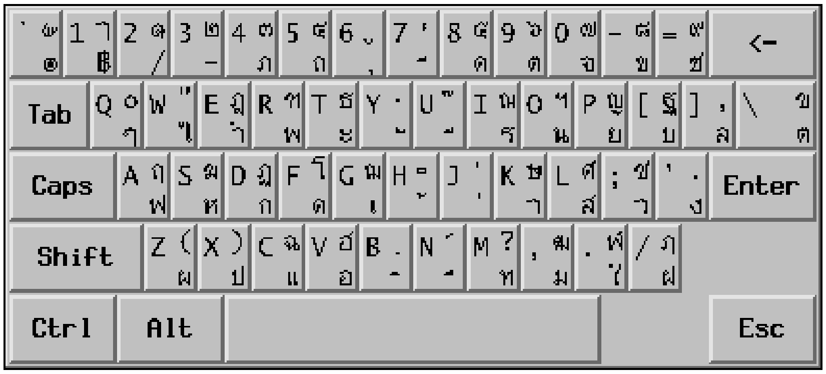
Thai Keyboard Layouts

Four different keyboard layouts are supported for the Thai input method.

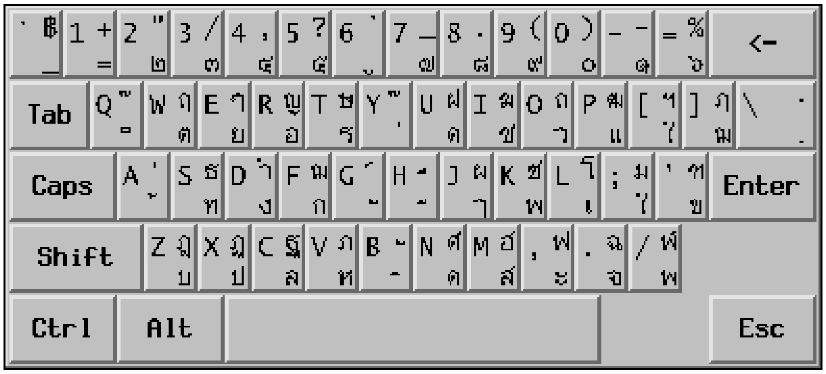
- Kedmanee (TIS820-2531) keyboard layout. The Kedmanee layout was designed for the typewriter, not the computer keyboard. The limited number of keys on the typewriter keyboard meant that some of the Thai special characters were not available in the layout. TIS820-2531 has adopted the Kedmanee layout for use with a computer keyboard.



- TIS820-2538 keyboard layout. This enhanced Kedmanee layout is an updated version of the TIS820-2531 layout that includes some of the Thai special characters that were unavailable in the original Kedmanee layout. Currently, TIS820-2538 is the only Thai keyboard layout standard that is issued by Thai Industrial Standard Institute.



- Pattajoti keyboard layout. The Pattajoti layout was also designed for the typewriter, but with better finger-load distribution.



- Configurable keyboard layout. User-defined keyboard layout for the Thai input method.

Thai Input Method Auxiliary Window

The Thai input method auxiliary window supports the following functions and utilities:

- Input level switching. You can click the input level button on the auxiliary palette to choose the passthrough, basic, or strict as your input level.

- Thai virtual keyboards. You can click the keyboard button to display the Thai virtual keyboard to use to enter Thai characters.

CHAPTER 5

Overview of UTF-8 Locale Support

This chapter provides an overview of UTF-8 locale support. The chapter covers the following topics:

Unicode Overview

Unicode is the universal character encoding standard used for representation of text for computer processing. Unicode is fully compatible with the international standards ISO/IEC 10646-1:2000 and ISO/IEC 10646–2:2001, and contains all the same characters and encoding points as ISO/IEC 10646. The Unicode Standard provides additional information about the characters and their use. Any implementation that conforms to Unicode also conforms to ISO/IEC 10646.

Unicode provides a consistent way of encoding multilingual plain text and facilitates exchanging international text files. Computer users who deal with multilingual text, business people, linguists, researchers, scientists, and others find that the Unicode Standard greatly simplifies their work. Mathematicians and technicians who regularly use mathematical symbols and other technical characters also find the Unicode Standard valuable.

The maximum possible number of code points Unicode can support is 1,114,112 through seventeen 16-bit planes. Each plane can support 65,536 different code points.

Among the more than one million code points that Unicode can support, version 4.0 curently defines 96,382 characters at plane 0, 1, 2, and 14. Planes 15 and 16 are for private use characters, also known as user-defined characters. Planes 15 and 16 together can support total 131,068 user-defined characters.

Unicode can be encoded using any of the following character encoding schemes:

- UTF-8
- UTF-16
- UTF-32

UTF-8 is a variable-length encoding form of Unicode that preserves ASCII character code values transparently. This form is used as file code in Oracle Solaris Unicode locales.

UTF-16 is a 16-bit encoding form of Unicode. In UTF-16, characters up to 65,535 are encoded as single 16-bit values. Characters mapped above 65,535 to 1,114,111 are encoded as pairs of 16-bit values (surrogates).

UTF-32 is a fixed-length, 21-bit encoding form of Unicode usually represented in a 32-bit container or data type. This form is used as the process code (wide-character code) in Oracle Solaris Unicode locales.

For more details on the Unicode Standard and ISO/IEC 10646 and their various representative forms, refer to the following sources:

- *The Unicode Standard, Version 4.0* from the Unicode Consortium
- ISO/IEC 10646-1:2000, Information Technology-Universal Multiple-Octet Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane
- ISO/IEC 10646-2: Information Technology-Universal Multiple-Octet Character Set (UCS) - Part 2: Secondary Multilingual Plane for Scripts and Symbols, Supplementary Plane for CJK Ideographs, Special Purpose Plane
- The Unicode Consortium web site at http://www.unicode.org/.

Unicode Locale: `en_US.UTF-8` Support

The Unicode/UTF-8 locales support Unicode 4.0. The `en_US.UTF-8` locale provides multiscript processing support by using `UTF-8` as its codeset. This locale handles processing of input and output text in multiple scripts, and was the first locale with this capability in the Oracle Solaris operating system. The capabilities of other UTF-8 locales are similar to those of `en_us.UTF-8`. The discussion of `en_US.UTF-8` that follows applies equally to these locales.

Note – `UTF-8` is a file-system safe Universal Character Set Transformation Format of Unicode/ISO/IEC 10646-1 formulated by X/Open-Uniforum Joint Internationalization Working Group (XoJIG) in 1992 and approved by ISO and IEC, as Amendment 2 to ISO/IEC 10646-1:1993 in 1996. This standard has been adopted by the Unicode Consortium, the International Standards Organization, and the International Electrotechnical Commission as a part of Unicode 4.0 and ISO/IEC 10646-1.

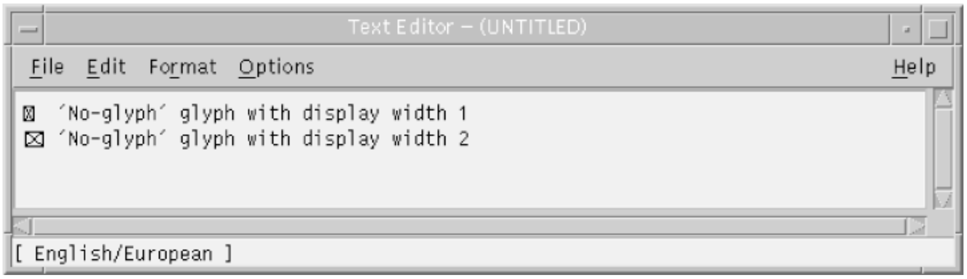
Unicode locales in the Oracle Solaris environment support the processing of every code point value that is defined in Unicode 4.0 and ISO/IEC 10646-1 and 10646-2. Supported scripts include pan-European and Asian scripts and also complex text layout scripts for the Arabic, Hebrew, Indic, and Thai languages.

Note – Some Unicode locales, notably the Asian locales, include more Kanji or Hanzi glyphs.

Due to limited font resources, the current Oracle Solaris Unicode locales include character glyphs from the following character sets.

- ISO 8859-1 (most Western European languages, such as English, French, Spanish, and German)
- ISO 8859-2 (most Central European languages, such as Czech, Polish, and Hungarian)
- ISO 8859-4 (Scandinavian and Baltic languages)
- ISO 8859-5 (Russian)
- ISO 8859-6 (Arabic, including many more presentation-form character glyphs)
- ISO 8859–7 (Greek)
- ISO 8859–8 (Hebrew)
- ISO 8859-9 (Turkish)
- TIS 620.2533 (Thai, including many more presentation-form character glyphs)
- ISO 8859–15 (most Western European languages with euro sign)
- GB 2312–1980 (Simplified Chinese)
- JIS X 0201–1976, JIS X 0208–1990 (Japanese)
- KSC 5601–1992 Annex 3 (Korean)
- GB 18030 (Simplified Chinese)
- HKSCS (Traditional Chinese, Hong Kong)
- Big5 (Traditional Chinese, Taiwan)
- IS 13194.1991, also known as ISCII (Hindi, including many more presentation-form character glyphs)

If you try to view characters for which the en_US.UTF-8 locale does not have corresponding glyphs, the locale displays a no-glyph glyph instead, as shown in the following illustration:



```
⊠ 'No-glyph' glyph with display width 1
⊠ 'No-glyph' glyph with display width 2
```

[English/European]

The locale is selectable at installation time and may be designated as the system default locale.

The same level of en_US.UTF-8 locale support is provided for both 64-bit and 32-bit Oracle Solaris systems.

Note – Motif and CDE desktop applications and libraries support the en_US.UTF-8 locale. However, XView™ and OLIT libraries do *not* support the en_US.UTF-8 locale.

About Desktop Input Methods

CDE provides the ability to enter localized input for an internationalized application using Xm Toolkit. The XmText[Field] widgets are enabled to interface with input methods from each locale. Input methods are internationalized because some language environments write their text from right-to-left, top-to-bottom, and so forth. Within the same application, you can use different input methods that apply several fonts.

The preedit area displays the string that is being pre-edited. Writing text can be done in four modes:

- OffTheSpot
- OverTheSpot (default)
- Root
- None

In OffTheSpot mode, the location is just below the main window area at the right of the status area. In OverTheSpot mode, the pre-edit area is at the cursor point. In Root mode, the preedit and status areas are separate from the client's window.

For more details, refer to the XmNpreeditType resource description in the VendorShell(3X) man page.

Input Method Support on the Oracle Solaris OS

Oracle Solaris has been adopting Internet Intranet Input Method Framework (IIIMF) to support multiple language input or scripts. The IIIM server starts per user in all the UTF-8 locales and Asian locales. It serves both IIIM and XIM (X input method) clients.

In European UTF-8 locales, Compose key input or dead key input is also available. For more information, see Appendix A, "Compose and Dead Key Input."

Available Input Method Engines

Various IMEs (Input Method Engine) are available such as Chinese, Japanese, Korean, Thai, Indic, Unicode (HEX/OCTAL). IIIMF also supports various EMEA keyboard layout emulations such as French, Russian or Arabic. You can find the existing IMEs through Input Method Preference Editor (`iiim-properties`).

Note – Asian IMEs that includes Chinese, Japanese, Korean, Thai, and Indic are available only when the corresponding locale support is installed.

Note – The English/European IME (Latin input) mode enables input of some Latin characters with diacritical marks, for example, á, è, î, õ and ü, without using Compose key. For example, " + A generates Ä.

Note – Table Lookup IME to input characters from Unicode character tables has been removed. Use the Character Map application (`charmap`) instead.

Basic Usage of Input Method

To activate and deactivate the input method, press the IM trigger key (for example, Shift_L + Alt_L). The current selected IME is activated. The Default IM trigger key is determined depending on the locale in which you log in the desktop for the first time. You can confirm the current IM trigger key by checking the Trigger Keys tab of `iiiim-properties`.

The IM status window shows the current input mode and selected IME. By default the IM status window is located at the bottom left corner of each application in the CDE environment. In the JDS environment, the IM status is shown on the Input Method Switcher application, `iiim-panel` resides in the Notification area on the Gnome panel.

To switch the IME, click the left mouse button on IM status window or `iiim-panel`. The language selection menu appears. Select the appropriate language that you want to switch.

Note – When the IM status window is used, the language selection menu is not available for gnome (GTK based) applications. You can switch to the IME through non-GTK application if the option, The language is applied to all applications, is enabled in `iiim-properties`. Otherwise use the `iiim-panel`.

For more information, see `iiim-properties` online help.

Customizing IIIM behaviors

Input Method Preference Editor, `iiim-properties`, customizes various IIIM behaviors. For example, you can change the IM trigger key, display the IM status and language selection menu, or add and delete IMEs.

You can start `iiim-properties` from the command line, `iiim-panel` or the desktop menu (Preferences menu in JDS and workspace menu->Tools in CDE).

For more customize options and detail information, see `iiim-properties` online help.

Note – IIIMF version has been upgraded from revision 10 to revision 12 since Oracle Solaris 10 6/06 release and each IME has also been upgraded correspondingly. This document explains about input methods based on IIIMF revision 12.

To upgrade IIIMF to revision 12 on earlier Oracle Solaris 10 system, apply the following patches.

```
120410-xx(SPARC) / 120411-xx(x86) - IIIMF rev.12 patch
121675-xx(SPARC) / 121676-xx(x86) - Japanese ATOK17 patch
121677-xx(SPARC) / 121678-xx(x86) - Japanese Wnn8 patch
120412-xx(SPARC) / 120413-xx(x86) - S-Chinese patch
120414-xx(SPARC) / 120415-xx(x86) - T-Chinese, Korean, Thai, Indic patch
```

IIIMF revision 10 is no longer supported in any of Oracle Solaris 10 releases.

System Environment

This section describes locale environment variables, TTY environment setup, 32–bit and 64–bit STREAMS modules, and terminal support.

Locale Environment Variable

Be sure you have the en_US.UTF-8 locale installed on your system. To check current locale settings in various categories, use the locale utility.

```
system% locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_ALL=
```

To use the en_US.UTF-8 locale desktop environment, choose the locale first. In a TTY environment, choose the locale first by setting the LANG environment variable to en_US.UTF-8, as in the following C-shell example:

```
system% setenv LANG en_US.UTF-8
```

Make sure that the LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_NUMERIC, LC_MONETARY, and LC_TIME categories are not set, or are set to en_US.UTF-8. If any of these categories is set, they override the lower-priority LANG environment variable. See the setlocale(3C) man page for more details about the hierarchy of environment variables.

You can also start the en_US.UTF-8 environment from the CDE desktop. At the CDE login screen's Options -> Language menu, choose en_US.UTF-8.

TTY Environment Setup

Depending on the terminal and terminal emulator that you are using, you might need to push certain code set-specific STREAMS modules onto your streams.

For more information on STREAMS modules and streams in general, see the *STREAMS Programming Guide*.

The following table lists the 64–bit STREAMS modules supported by the en_US.UTF-8 locale in the terminal environment. For more details, see the *Oracle Solaris 64–bit Developer's Guide*.

TABLE 5–1 STREAMS Modules Supported by en_US.UTF-8

32-bit STREAMS module	Description
/usr/kernel/strmod/sparcv9/u8lat1	Code conversion STREAMS module between UTF-8 and ISO8859-1 (Western European)
/usr/kernel/strmod/sparcv9/u8lat2	Code conversion STREAMS module between UTF-8 and ISO8859-2 (Eastern European)

TABLE 5–1 STREAMS Modules Supported by en_US.UTF-8 *(Continued)*

32-bit STREAMS module	Description
/usr/kernel/strmod/sparcv9/u8koi8	Code conversion STREAMS module between UTF-8 and KOI8-R (Cyrillic)

Note – Starting with the Oracle Solaris 10 release, the 32-bit kernel is no longer supported for the SPARC platform. Table 5–1 applies only to the 32-bit kernel for the x86 platform. For more details, refer to the Release Notes.

The following table lists the 64–bit STREAMS modules supported by en_US.UTF-8.

TABLE 5–2 64–bit STREAMS Modules Supported by en_US.UTF-8

64-bit STREAMS Module	Description
/usr/kernel/strmod/sparcv9/u8lat1	Code conversions STREAMS module between UTF-8 and ISO8859-1 (Western European)
/usr/kernel/strmod/sparcv9/u8lat2	Code conversions STREAMS module between UTF-8 and ISO8859-2 (Eastern European)
/usr/kernel/strmod/sparcv9/u8koi8	Code conversions STREAMS module between UTF-8 and KOI8-R (Cyrillic)

▼ How to Load a STREAMS Kernel Module

1. **As the root user, determine whether you are running a 64-bit Oracle Solaris or 32-bit Oracle Solaris system.**

   ```
   system# isainfo -v
   ```

   - A 64—bit Oracle Solaris system returns the following information:

     ```
     64-bit sparcv9 applications
     32-bit sparc applications
     ```

   - A 32—bit Oracle Solaris system returns the following information:

     ```
     32-bit sparc applications
     ```

   - A 32—bit x86 system returns the following information:

     ```
     32-bit i386 applications
     ```

2. **Determine whether your system has already loaded the STREAMS module.**

   ```
   system# modinfo | grep modulename
   ```

If the STREAMS module, such as `u8lat1`, is already installed, the output looks as follows:

```
system# modinfo | grep u8lat1
89 ff798000  4b13  18   1  u8lat1 (UTF-8 <--> ISO 8859-1 module)
```

3 **If the module has not already been loaded, load it using the `modload(1M)` command.**

- On a 32—bit system, you would type:

  ```
  system# modload /usr/kernel/strmod/u8lat1
  ```
- On a 64—bit system, you would type:

  ```
  system# modload /usr/kernel/strmod/sparcv9/u8lat1
  ```

  The appropriate `u8lat1` STREAMS module is loaded in the kernel. You can now push it onto a stream.

▼ How to Unload a STREAMS Kernel Module

1 **As root, verify that the kernel module is loaded.**

For example, to verify the `u8lat1` is loaded, you would type:

```
system# modinfo | grep u8lat1
89 ff798000  4b13  18   1  u8lat1 (UTF-8 <--> ISO 8859-1 module)
```

2 **Use the `modunload(1M)` command to unload the kernel.**

For example, to unload the `u8lat1` module, you would type:

```
system# modunload -i 89
```

▼ How to Setup a Latin-2 Terminal and STREAMS Module

1 **Use the `strchg`(1M), as shown in the second command line**

```
system% cat > tmp/mystreams
ttcompat
ldterm
u8lat1
ptem
^D
system% strchg -f /tmp/mystreams
```

Be sure that you are either root or the owner of the device when you use `strchg(1)`.

2 **Run the strconf command to examine the current configuration.**

```
system% strconf
ttcompat
ldterm
u8lat1
ptem
pts
system%
```

3 Run the strchg command to reset the original configuration.

```
system% cat > /tmp/orgstreams
ttcompat
ldterm
ptem
^D
system% strchg -f /tmp/orgstreams
```

`dtterm`, `xterm` and Terminals Capable of Input and Output of UTF-8 Characters

Unlike the older releases of the Oracle Solaris operating system, the `dtterm` and `xterm` terminal emulators and any other terminals that support input and output of the `UTF-8` code set, do not need to have any additional STREAMS modules in their streams. The `ldterm` module is now codeset independent and supports Unicode/`UTF-8` if you set up the terminal environment with the `stty`(1) utility.

To set up the proper terminal environment for the Unicode locales, use the `stty(1)` utility.

```
system% /bin/stty defeucw
```

To query the current settings, use the `-a` option of the `stty` utility, as shown below:

```
system% /bin/stty -a
```

Note – Because `/usr/ucb/stty` is not internationalized, use `/bin/stty` instead.

Terminal Support for Latin-1, Latin-2, or KOI8-R

For terminals that support only Latin-1 (ISO8859-1), Latin-2 (ISO8859-2), or KOI8-R, you should have the following STREAMS configuration:

```
head <-> ttcompat <->  ldterm <->  u8lat1 <-> TTY
```

This configuration is only for terminals that support Latin-1. For Latin-2 terminals, replace the STREAMS module `u8lat1` with `u8lat2`. For KOI8-R terminals, replace the module with `u8koi8`.

Make sure you already have the STREAMS module loaded into the kernel.

Saving the Settings in `~/.cshrc`

Assuming the necessary STREAMS modules are already loaded with the kernel, you can save the following lines in your `.cshrc` file (C shell example) for convenience:

```
setenv LANG en_US.UTF-8
if ($?USER != 0 && $?prompt != 0) then
     cat >! /tmp/mystreams$$ << _EOF
     ttcompat
     ldtterm
     u8lat1
     ptem
_EOF
     /bin/strchg -f /tmp/mystreams$$
     /bin/rm -f /tmp/mystreams$$
     /bin/stty cs8 -istrip defeucw
endif
```

With these lines in your .cshrc file, you do not have to type all of the commands each time you use the STREAMS module. Note that the second _EOF should start from the first column of the file.

Code Conversions

Unicode locale support adds various code conversions among major code sets of many countries through iconv and sdtconvtool utilities.

In the current Oracle Solaris environment, the utility geniconvtbl enables user-defined code conversions. The user-defined code conversions created with the geniconvtbl utility can be used with both iconv(1) and iconv(3). For more details about this utility, refer to the geniconvtbl(1) and geniconvtbl(4) man pages.

The available fromcode and tocode names that can be applied to iconv, iconv_open, and sdtconvtool are described by the following man pages:

- iconv_1250(5)
- iconv_1251(5)
- iconv_646(5)
- iconv_852(5)
- iconv_8859-1(5)
- iconv_8859-2(5)
- iconv_8859-5(5)
- iconv_dhn(5)
- iconv_en_US.UTF-8(5)
- iconv_ja(5)
- iconv_ko(5)
- iconv_koi8-r(5)
- iconv_mac_cyr(5)
- iconv_maz(5)
- iconv_pc_cyr(5)
- iconv_unicode(5)
- iconv_zh(5)
- iconv_zh_TW(5)

For more details about iconv code conversion, see the iconv(1), and sdtconvtool(1) man pages. For more information about the available code conversions, see the iconv(5) man page.

Note – UCS-2, UCS-4, UTF-16 and UTF-32 are all Unicode/ ISO/IEC 10646 representation forms that recognize Byte Order Mark (BOM) characters defined in the Unicode 4.0 and ISO/IEC 10646-1:2000 standards if the character appears at the beginning of the character stream. Other forms, like UCS-2BE, UCS-4BE, UTF-16BE, and UTF-32BE, are fixed-width Unicode/ISO/IEC 10646 representation forms that do not recognize the BOM character and also assume big endian byte ordering. Representation forms like UCS-2LE, UCS-4LE, UTF-16LE, and UTF-32LE, on the other hand, assume little endian byte ordering. These forms also do not recognize the BOM character.

For associated scripts and languages of ISO8859–* and KO18–*, see `http://czyborra.com/charsets/iso8869.html`.

Configuring Fonts

Oracle Solaris desktop environment uses `fontconfig` for font configuration. For more information about how to configure fonts in Oracle Solaris, refer to Chapter 4, "Configuring Fonts," in *Java Desktop System Release 3 Administration Guide*.

DtMail Support

As a result of increased coverage in scripts, Oracle Solaris DtMail running in the `en_US.UTF-8` locale supports the following character sets, indicated by the following MIME names:

- US-ASCII (7-bit US ASCII)
- UTF-8 (UCS Transmission Format 8 bit)
- UTF-7 (UCS Transmission Format 7 bit)
- ISO-8859-1 (Latin-1)
- ISO-8859-2 (Latin-2)
- ISO-8859-3 (Latin-3)
- ISO-8859-4 (Latin-4)
- ISO-8859-5 (Latin/Cyrillic)
- ISO-8859-6 (Latin/Arabic)
- ISO-8859-7 (Latin/Greek)
- ISO-8859-8 (Latin/Hebrew)
- ISO-8859-9 (Latin-5)
- ISO-8859-10 (Latin-6)
- ISO-8859-13 (Latin-7/Baltic)
- ISO-8859-14 (Latin-8/Celtic)
- ISO-8859-15 (Latin-9)
- ISO-8859-16 (Latin-10)
- KOI8-R (Cyrillic)

- ISO-2022-JP and EUC-JP (Japanese)
- ISO-2022-KR and EUC-KR (Korean)
- ISO-2022-CN (Simplified Chinese)
- ISO-8859–13 (Latin-7/Baltic)
- ISO-8859–14 (Latin-8/Celtic)
- KOI8–U (Cyrillic/Ukrainian)
- Shift_JIS (Japanese in Shift JIS)
- GB2312 (Simplified Chinese in EUC)
- TIS-620 (Thai)
- UTF-16 (UCS Transmission Format 16 bit)
- UTF-16BE (UTF-16 Big-Endian)
- UTF-16LE (UTF-16 Little-Endian)
- Windows-1250
- Windows-1251
- Windows-1252
- Windows-1253
- Windows-1254
- Windows-1255
- Windows-1256
- Windows-1257
- Windows-1258
- Big5 (Traditional Chinese)
- UTF-32 (UCS Transmission Format 32 bit)
- UTF-32BE (UTF-32 Big-Endian)
- UTF-32LE (UTF-32 Little-Endian)

This support enables users to view virtually any kind of email encoded in various character sets from any region of the world in a single instance of DtMail. DtMail decodes received email by looking at the MIME charset and content transfer encoding provided with the email. Windows-125x MIME charsets are supported.

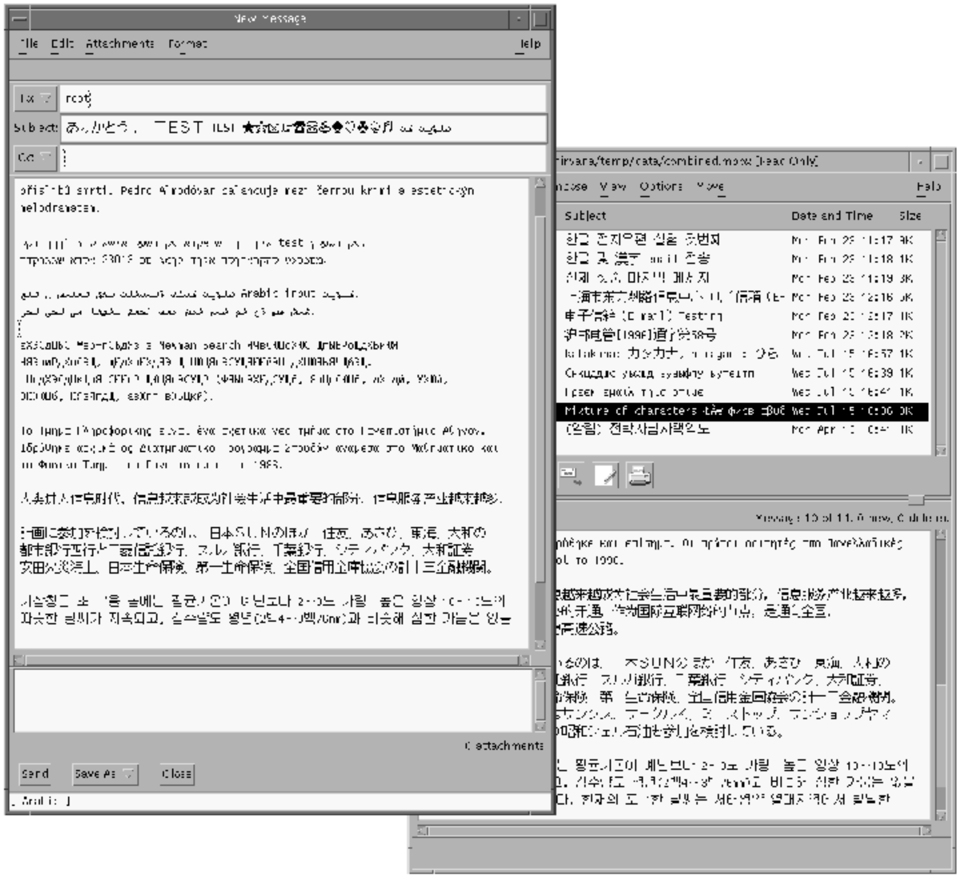
For sending email, you need to specify a MIME charset that is understood by the recipient mail user agent (mail client), or you can use the default MIME charset provided by the `en_US.UTF-8` locale. You can switch the character set of outgoing email, in the New Message window, press Control Y, or click the Format menu button and then click the Change Char Set button. The next available character set name displays in the bottom left corner at the top of the Send button.

If your email message header or message body contains characters that cannot be represented by the MIME charset specified, the system automatically switches the charset to `UTF-8` which can represent any character.

If your message contains characters from the 7-bit US-ASCII character set only, the default MIME charset of your email is `US-ASCII`. Any mail user agent can interpret such email messages without loss of characters or information.

If your message contains characters from a mixture of scripts, the default MIME charset is `UTF-8`. Any 8-bit characters of `UTF-8` are encoded with Quoted-Printable encoding. For more details on MIME, registered MIME charsets, and Quoted-Printable encoding, refer to RFCs 2045, 2046, 2047, 2048, 2049, 2279, 2152, 2237, 1922, 1557, 1555, and 1489.

FIGURE 5–1 DtMail New Message Window



Subject	Date and Time	Size
[illegible]	Mon Feb 23 11:17	9K
[illegible]	Mon Feb 23 11:18	1K
[illegible]	Mon Feb 23 11:19	3K
[illegible]	Mon Feb 23 12:16	5K
[illegible] Testing	Mon Feb 23 12:17	1K
[illegible]	Mon Feb 23 12:18	2K
[illegible] カタカナ [illegible]	Wed Jul 15 16:57	1K
[illegible]	Wed Jul 15 16:39	1K
[illegible]	Wed Jul 15 16:41	1K
Mixture of characters [illegible]	Wed Jul 15 10:06	3K
[illegible]	Mon Apr 10 [illegible]	1K

Message 10 of 11, 0 new, 0 deleted

Programming Environment

Internationalized applications should automatically enable the en_US.UTF-8 locale. However, proper FontSet/XmFontList definitions in the application's resource file are required.

For information on internationalized applications, see *Creating Worldwide Software: Oracle Solaris International Developer's Guide*, 2nd edition.

FontSet Used with X Applications

For information about the FontSet used with X applications, please see "Unicode Locale: en_US.UTF-8 Support" on page 118.

Each character set has an associated set of fonts in the Oracle Solaris desktop environment.

The following is a list of the Latin-1 fonts that are supported in the current Oracle Solaris environment:

```
-dt-interface system-medium-r-normal-xxs sans utf-10-100-72-72-p-59-iso8859-1
-dt-interface system-medium-r-normal-xs sans  utf-12-120-72-72-p-71-iso8859-1
-dt-interface system-medium-r-normal-s sans  utf-14-140-72-72-p-82-iso8859-1
-dt-interface system-medium-r-normal-m sans  utf-17-170-72-72-p-97-iso8859-1
-dt-interface system-medium-r-normal-l sans  utf-18-180-72-72-p-106-iso8859-1
-dt-interface system-medium-r-normal-xl sans utf-20-200-72-72-p-114-iso8859-1
-dt-interface system-medium-r-normal-xxl sans utf-24-240-72-72-p-137-iso8859-1
```

For information on CDE common font aliases, including -dt-interface user-* and-dt-application-* aliases, see *Common Desktop Environment: Internationalization Programmer's Guide*.

In the en_US.UTF-8 locale, utf is also included in the locale's common font aliases as an additional attribute in the style field of the X logical font description name. Therefore, to have a proper set of fonts, the additional style has to be included in the font set creation as in the following example:

```
fs = XCreateFontSet(display,
"-dt-interface system-medium-r-normal-s*utf*",
 &missing_ptr, &missing_count, &def_string);
```

FontList Definition in CDE/Motif Applications

As with FontSet definition, the XmFontList resource definition of an application should also include the additional style attribute supported by the locale.

```
*fontList:\
 -dt-interface system-medium-r-normal-s*utf*:
```

◆ ◆ ◆ CHAPTER 6

Complex Text Layout

Complex Text Layout (CTL) extensions enable the Motif APIs to support writing systems that require complex transformations between logical and physical text representations. Arabic, Hebrew, and Thai languages require such transformations. CTL Motif provides character shaping, such as ligatures, diacritics, and segment ordering. Support for the transformations of static and dynamic text widgets is also provided, along with bidirectional text capability and tabbing for dynamic text widgets. Because text rendering is handled through the rendition layer, other widget libraries can easily be extended to support CTL.

This chapter covers the following topics:

Overview of CTL Technology

To leverage the new features, users must have the Portable Layout Services (PLS) library and the appropriate language engine. CTL uses PLS as the interface to the language engine, and uses the language engine to transform text before the text is rendered. Applications that support CTL must include additional resources, as described in the CTL documentation.

Specifically, XomCTL supports the following complex language shaping and reordering features provided by underlying locale-dependent PLS module transformations:

- Positional variation
- Ligation (many-to-one) and character composition (one-to-many)
- Diacritics
- Bidirectionality

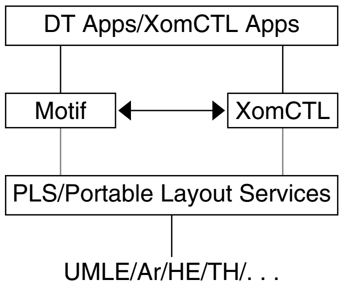
- Symmetrical swapping
- Numeral shaping
- String validation

Overview of CTL Architecture

The CTL architecture is organized as shown in Figure 6–1. `Dt Apps` at the top of the stack employs Motif CTL functionality for rendering text. Motif in turn interfaces with locale-specific language engines using PLS, and performs transformations to support positional variation, numeral shaping, and so on.

The CTL architecture supports new languages with a locale-specific engine. In other words, support for Thai and Vietnamese can be added without altering Motif or `Dt Apps`.

FIGURE 6–1 CTL Architecture



CTL Support for X Library Based Applications

XomCTL (Complex Text Layout support in X Library Output Module) enables all pure X Windows applications, such as an X-based terminal emulator, to have CTL support. XomCTL provides a full-featured Open Source XI18N implementation including X11 dumb font support.

XOC Resources

The following XOC resources are provided in the current Oracle Solaris environment:

`XNText`	Enables the user to set the text buffer on which CTL operation needs to be performed
`XNTextLayoutNumGlyphs`	Provides the number of glyphs for the text in the text buffer
`XNTextLayoutModifier`	Same as the XmNLayoutModifier of Motif

`XNTextLayoutProperty`	Same as the PLS Property, input-to-output and output-to-input
`XNTextLayoutMapInpToOut`	Same as the PLS Property, input-to-output and output-to-input
`XNTextLayoutMapOutToInp`	Same as the PLS Property, input-to-output and output-to-input

Descriptions of these resources may be obtained from the specification of X/Open or PLS Portable Layout Services.

Changes in Motif to Support CTL Technology

The following changes to Motif support the CTL technology:

`XmNlayoutDirection`	Controls object layout
`XmStringDirection`	Specifies the direction in which the system displays characters of a string
`XmRendition`	Adds new pseudo resources to `XmRendition`
`XmText` and `XmTextField`	Affects the layout behavior of the text associated with the `XmRendition`
`XmTextFieldGetLayoutModifier`	Returns the layout modifier string of a rendition layout object
`XmTextGetLayoutModifier`	Returns the value of the current layout object settings of the rendition associated with the widget
`XmTextFieldSetLayoutModifier`	Sets the layout modifier values for the layout object tied to its rendition
`XmTextSetLayoutModifier`	Modifies the layout object settings of a rendition associated with the widget
`XmStringDirectionCreate`	Creates a compound string

`XmNlayoutDirection` Resource

The `XmNlayoutDirection` resource controls object layout. This resource interacts with the orientation value of the `LayoutObject` in the manner described below.

See section 11.3 of the Motif *Programmer's Guide* (Release 2.1) for an overview of `XmNlayoutDirection`, and especially for a description of the interaction between `XmStringDirection` and `XmNlayoutDirection`.

Determining the Layout Direction

When the `XmNlayoutDirection` is specified as `XmDEFAULT_DIRECTION`, the layout direction of the widget is set at creation time from the governing pseudo-XOC. In the case of dynamic text (`XmText` and `XmTextField`), the governing pseudo-XOC is the one that is associated with the `XmRendition` used for the widget. In the case of static text (`XmList`, `XmLabel`, `XmLabelG`), the layout direction is set from the first compound string component that specifies a direction. This specification happens in one of two ways:

- The component is of type `XmSTRING_COMPONENT_LAYOUT_PUSH` or `XmSTRING_COMPONENT_DIRECTION.`
- The component is of type `XmSTRING_COMPONENT_LOCALE_TEXT`, `XmSTRING_COMPONENT_WIDECHAR_TEXT`, or `XmSTRING_COMPONENT_TEXT`, from the associated `XmRendition` and `LayoutObject`.

When `XmNlayoutDirection` is not specified as `XmDEFAULT_DIRECTION` and the `XmNlayoutModifier @ls orientation` value is not specified explicitly in the layout modifier string, then the `XmNlayoutDirection` value is passed through to the XOC and its `LayoutObject`.

If both `XmNlayoutDirection` and the `XmNlayoutModifier @ls orientation` value are explicitly specified, then the behavior is mixed. The `XmNlayoutDirection` controls widget object layout, and the `XmNlayoutModifier @ls orientation` value controls layout transformations.

See *CAE Specification: Portable Layout Services: Context-dependent and Directional Text* (The Open Group: Feb 1997; ISBN 1-85912-142-X; document number C616) for a description of portable functions for handling context-dependent and bidirectional text transformations as a logical extension to the existing POSIX locale model. The document is intended for system and application programmers who want to provide support for complex-text languages.

`XmStringDirection` Resource

`XmStringDirection` is the data type used to specify the direction in which the system displays characters of a string.

The `XmNlayoutDirection` resource sets a default rendering direction for any compound string (`XmString`) that does not have a component specifying the direction of that string. Therefore, to set the layout direction, you need to set the appropriate value for the `XmNlayoutDirection` resource. You do not need to create compound strings with specific direction components.

When the application renders an `XmString`, the application should check whether the string was created with an explicit direction (`XmStringDirection`). If the string does not provide a direction component, the application should check the value of the `XmNlayoutDirection` resource for the current widget and use that value as the default rendering direction for the `XmString`.

`XmRendition` Resource

CTL adds the new pseudo resources listed in the following table to `XmRendition`. Descriptions of the pseudo resources follow the table.

TABLE 6–1 New Resources in `XmRendition`

Name	Class/Type	Access	Default Value
`XmNfontType`	`XmCFontType/XmFontType`	CSG	`XmAS_IS`
`XmNlayoutAttrObject`	`XmClayoutAttrObject/String`	CG	`NULL`
`XmNlayoutModifier`	`XmClayoutModifier/String`	CSG	`NULL`

`XmNfontType`
: Specifies the type of the Rendition font object. For CTL, the value of this resource must be the `XmFONT_IS_XOC` value. If the value does not match, then the `XmNlayoutAttrObject` and `XmNlayoutModifier` resources are ignored.

  When the value of this resource is `XmFont_IS_XOC` and the `XmNfont` resource is not specified, then at create time the value of the `XmNfontName` resource is converted into an XOC object in either the locale specified by the `XmNlayoutAttrObject` resource or the current locale. Furthermore, the value of the `XmNlayoutModifier` resource is passed through to any layout object associated with the XOC.

`XmNlayoutAttrObject`
: Specifies the layout `AttrObject` argument. This resource is used to create the layout object associated with the XOC associated with this `XmRendition`. Refer to the layout services `m_create_layout()` specification for the syntax and semantics of this string. See the description of `XmNfontType` for an explanation of the interaction between the Layout Modifier Orientation output value and the `XmNlayoutDirection` widget resource.

`XmNlayoutModifier`
: Specifies the layout values to be passed through to the layout object used with the XOC for this `XmRendition`. For the syntax and semantics of this string, see *CAE Specification*.

  Setting this resource using `XmRendition{Retrieve,Update}` causes the string to be passed through to the layout object associated with the XOC associated with this rendition. This mechanism enables you to configure layout services dynamically. Unpredictable behavior can result if the Orientation, Context, TypeOfText, TextShaping, or ShapeCharset are changed.

Additional Layout Behavior

The XmNlayoutModifier affects the layout behavior of the text associated with the XmRendition. For example, if the layout default treatment of numerals is NUMERALS_NOMINAL, you change to NUMERALS_NATIONAL by setting XmNlayoutModifier to @ls numerals=nominal:national, or @ls numerals=:national.

The layout values can be classified into the following groups:

- Encoding description – TypeOfText, TextShaping, ShapeCharset (and locale codeset)

  TypeOfText is essentially segment ordering and can be illustrated with opaque blocks. Modifying these values dynamically through the rendition object is not usually meaningful, and is almost certain to result in unpredictable behavior.
- Layout behavior – Orientation, Context, ImplicitAlg, Swapping, and Numerals. Orientation and Context should not be modified dynamically. You can safely modify ImplicitAlg, Swapping, and Numerals.
- Editing behavior – CheckMode

XmText and XmTextField Resource

Xm CTL extends XmText and XmTextField by adding a parallel set of movement and deletion actions that operate visually, patterned after the Motif 2.0 CSText widget. The standard Motif 2.1 Text and TextField do not distinguish between logical and physical order: *next* and *forward* mean "to the right," while *previous* and *backward* mean "to the left." CSText, however, makes the proper distinction and defines a new set of actions with strictly physical names (for example, left-character(), delete-right-word(), and so on). These action routines are defined to be sensitive to the XmNlayoutDirection of the widget and to call the appropriate *next-* or *previous-* action.

The Xm CTL extensions are slightly more complex than the CSText extensions. The Xm CTL extensions are sensitive not to the global orientation of the widget, but to the specific directionality of the physical characters surrounding the cursor, as determined by the pseudo-XOC, including neutral stabilization.

The new resource name enables you to control selection policy, to provide a rendition tag, and to control alignment.

The set of new Xm CTL actions is roughly the cross product of {Move,Delete,Kill} by {Left,Right} by {Character,Word}. The action set is listed in the following table.

TABLE 6–2 New Resources in Xm CTL

Name	Class/Type	Access	Default Value
`XmNrenditionTag`	`XmCRenditionTag/XmRString`	CSG	`XmFONTLIST_DEFAULT_TAG`
`XmNalignment`	`XmCAlignment/XmRAlignment`	CSG	`XmALIGNMENT_BEGINNING`
`XmNeditPolicy`	`XmCEditPolicy/XmREditPolicy`	CSG	`XmEDIT_LOGICAL`

`XmNrenditionTag`
Specifies the rendition tag of the `XmRendition` that is in the `XmNrenderTable` resource, used for a widget.

`XmNalignment`
Specifies the text alignment used in the widget. Only `XmALIGNMENT_END` and `XmALIGNMENT_CENTER` are supported.

`XmNeditPolicy`
Specifies the editing policy used for the widget, either `XmEDIT_LOGICAL` or `XmEDIT_VISUAL`. In the case of `XmEDIT_VISUAL`, selection, cursor movement, and deletion are in a visual style. Setting this resource also changes the translations for the standard keyboard movement and deletion events either to the new “visual” actions list or to the existing logical actions.

Character Orientation Action Routines

The `forward-cell()` and `backward-cell()` actions query the orientation of the character in the direction specified. If the direction is left-to-right, these actions call the corresponding *next-/forward-* or *previous-/backward-* variants.

Character Orientation Additional Behavior

The actions determine the orientation of characters by using the Layout Services transformation `OutToInp` and `Property` buffers for the nesting level. The widget's behavior is therefore dependent on the locale-specific transformation. If the information in the `OutToInp` or, especially, `Property` buffers is inaccurate, the widget might behave unexpectedly. Moreover, as the locale-specific modules fall outside of the scope of this specification, bidirectional editing behavior can differ from platform to platform for the same text, application, resource values, and `LayoutObject` configuration.

The visual mode actions result in a display of cell-based behavior. The logical mode actions result in logical character-based behavior. For example, the `delete-right-character()` operation deletes the input buffer characters that correspond to the display cell. That is, one input buffer character whole `LayoutObject` transformation “property” byte “new cell indicator” is 1, and all succeeding characters whose “new cell indicator” is 0.

For more information on the `Property` buffer, see the specification for `m_transform_layout()` in *CAE Specification*.

Similarly, for `backward-character()`, the insertion point is moved backward one character in the input buffer, and the cursor is redrawn at the visual location corresponding to the associated output buffer character. Therefore, several keystrokes are required to move across a composite display cell. The cursor does not actually change display location as the insertion point moves across input buffer characters such as diacritics or ligature fragments whose “new cell indicator” is 0.

This behavior means that deletion operates either from the logical/input buffer side, or from the display cell level of the physical/output side. No mode exists for a strict, physical character-by-character deletion because no one-to-one correspondence exists between the input and output buffers. A given physical character can represent only a fragment of a logical character, for example.

`XmText` Action Routines

The following list describes the `XmText` action routines.

`left-character(extend)`

If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and it is called without arguments, the insertion cursor moves back logically by a character. If the insertion cursor is at the beginning of the line, the insertion cursor moves to the logical last character of the previous line, if one exists. Otherwise, the insertion cursor position doesn't change.

If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then the cursor moves to the left of the cursor position. If the insertion cursor is at the beginning of the line, then it moves to the end character of the previous line, if one exists.

If `left-character`() is called with an `extend` argument, the insertion cursor moves as in the case of no argument, and extends the current selection.

The `left-character`() action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with an `extend` argument, this action can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

`right-character(extend)`

If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and it is called without any arguments, the insertion cursor moves logically forward by a character. If the insertion cursor is at the logical end of the line, this action moves the insertion cursor to the logical start of the next line, if one exists.

If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then the cursor moves to the right of the cursor position. If the insertion cursor is at the end of the line, it moves the insertion cursor to the starting of the next line, if one exists.

If called with an argument of `extend`, `XmNeditPolicy` moves the insertion cursor as in the case of no argument, and extends the current selection.

The `right-character()` action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with `extend` argument, this action can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

`right-word(extend)`

If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and it is called without any arguments, the insertion cursor moves to the logical starting character of the logical succeeding word, if one exists. Otherwise, the cursor moves to the logical end of the current word. If the insertion cursor is at the logical end of the line or in the logical last word of the line, the cursor moves to the logical first word in the next line, if one exists. Otherwise, the cursor moves to the logical end of the current word.

If the `XmNeditPolicy` is `XmEDIT_VISUAL` and it is called without arguments, the insertion cursor moves to the first non whitespace character after the first white space character to the right or after the end of the line.

If called with an argument of `extend`, the insertion cursor moves as in the case of no argument and extends the current selection.

The `left-word()` action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with `extend` argument, this action can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

`delete-left-character()`

If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, it is equivalent to `delete-previous-char()`. If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then in normal mode, if the selection is non-null, it deletes the selection. Otherwise this action deletes the character to the left of the insertion cursor. In add mode, if the selection is non-null, the cursor is not disjointed from the selection, and `XmNpendingDelete` is set to True, this action deletes the selection. Otherwise, the action deletes the character to the left of the insertion cursor, which can affect the selection.

The `delete-left-character()` action produces calls to the `XmNmodifyVerifyCallback` procedures with the reason value `XmCR_MODIFYING_TEXT_VALUE` and the `XmNvalueChangedCallback` procedures with the reason value `XmCR_VALUE_CHANGED`.

`delete-right-character()`

If the `XmNeditPolicy` is `XmEDIT_VISUAL`, it is equivalent to `delete-next-character()`. If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then in normal mode, if the selection is a non-null, it deletes the selection. Otherwise, it deletes the character to the right of the insertion cursor. In add mode, if there is a non-null selection and the cursor is not disjointed from the selection, the `XmNpendingDelete` is set to True and the selection is deleted. Otherwise, the character to the right of the insertion cursor is deleted. This action can affect the selection.

The `delete-right-character()` action produces calls to the `XmNmodifyVerify-Callback` procedures with reason value `XmCR_MODIFYING_TEXT_VALUE`, and the `XmNvalue-ChangedCallback` procedures with reason value `XmCR_VALUE_CHANGED`.

A few cell-based routines are implemented to support character composition, ligatures, and diacritics. In other words, two or more characters might be represented by a single glyph occupying one presentation cell.

The `XmText` cell action routines are as described in the following list.

`backward-cell(extend)`
: Moves the insertion cursor back one cell. If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then the insertion cursor is moved to the start of the cell that precedes the current cell logically, if one exists. Otherwise, the cursor moves to the start of the current cell.

  If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then the cursor moves to the start of cell to the left of the cursor, if one exists. The `prev-cell()` action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with an `extend` argument, this action can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

`forward-cell(extend)`
: Moves the insertion cursor to the start of the logical next cell, if one exists. Otherwise this action moves the cursor to the end of the cell. If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then the cursor moves forward one cell.

  If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then the cursor moves to the start of the cell to the right of the cursor position, if one exists; otherwise, it moves to the end of the current cell. The `forward-cell()` action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with an `extend` argument, this action can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

`XmTextFieldGetLayoutModifier` Resource

`XmTextFieldGetLayoutModifier()` returns the layout modifier string that reflects the state of the layout object tied to its rendition.

The syntax for `XmTextFieldGetLayoutModifier()` is:

```
#include <Xm/TextF.h>
string XmTextFieldGetLayoutModifier(Widget widget)
```

`XmTextFieldGetLayoutModifier()` accesses the value of the current layout object settings of the rendition associated with the widget. When the layout object modifier values are changed using a convenience function, the `XmTextFieldGetLayoutModifier` function returns the complete state of the layout object, not the changed values only.

`XmTextFieldGetLayoutModifier()` returns the layout object modifier values in the form of a string value.

`XmTextGetLayoutModifier` Resource

`XmTextGetLayoutModifier()` returns the layout modifier string that reflects the state of the layout object tied to its rendition.

The syntax for `XmTextGetLayoutModifier()` is:

```
#include <Xm/Text.h>
String XmTextGetLayoutModifier(Widget widget)
```

`XmTextGetLayoutModifier` accesses the value of the current layout object settings of the rendition associated with the widget. When the layout object modifier values are changed using a convenience function, the `XmTextGetLayoutModifier` function returns the complete state of the layout object, not just the changed values.

`XmTextGetLayoutModifier` returns the layout object modifier values in the form of a string value.

`XmTextFieldSetLayoutModifier` Resource

`XmTextFieldSetLayoutModifier()` sets the layout modifier values, which changes the behavior of the layout object tied to its rendition.

The syntax for `XmTextFieldSetLayoutModifier()` is:

```
#include <Xm/TextF.h> \
void XmTextFieldSetLayoutModifier(Widget \
widgetstring layout_modifier)
```

`XmTextFieldSetLayoutModifier` modifies the layout object settings of a rendition associated with the widget. When the layout object modifier values are set using this convenience function, only the attributes specified in the input parameter are changed. The rest of the attributes remain untouched.

`XmTextSetLayoutModifier` Resource

`XmTextSetLayoutModifier()` sets the layout modifier values, which changes the behavior of the layout object tied to its rendition.

The syntax for `XmTextSetLayoutModifier()` is:

```
#include <Xm/Text.h>
void XmTextSetLayoutModifier(Widget widget,string layout_modifier)
```

XmTextSetLayoutModifier modifies the layout object settings of a rendition associated with the widget. When the layout object modifier values are set using this convenience function, only the attributes specified in the input parameter are changed; the rest of the attributes are left untouched.

XmStringDirectionCreate Resource

XmStringDirectionCreate creates a compound string.

The syntax for XmTextSetLayoutModifier() is:

```
#include <Xm/Xm.h>
XmString XmStringDirectionCreate(direction)
XmStringDirection direction
```

XmStringDirectionCreate creates a compound string with a single component, a direction with the given value. On the other hand, the XmNlayoutDirection resource sets a default rendering direction for any compound string (XmString) that does not have a component specifying the direction for that string. Therefore, to set the layout direction, you set the appropriate value for the XmNlayoutDirection resource. You need not create compound strings with specific direction components.

When the application renders an XmString, the application should check whether the string was created with an explicit direction (XmStringDirection). If the application was provided no direction component, the application should check the value of the XmNlayoutDirection resource for the current widget and use that value as the default rendering direction for the XmString.

UIL Arguments

The following table shows the UIL argument name and type.

TABLE 6–3 UIL

UIL Argument Name	Argument Type
XmNlayoutAttrObject	String
XmNlayoutModifier	String
XmNrenditionTag	String
XmNalignment	Integer

TABLE 6–3 UIL *(Continued)*

UIL Argument Name	Argument Type
`XmNeditPolicy`	Integer

Developing CTL Applications

The following sections explain how to develop CTL applications.

Controlling Layout Direction

The direction of a compound string is stored so that the data structure is equally useful for describing text in left-to-right languages such as English, Spanish, French, and German, or for text in right-to-left languages, such as Hebrew and Arabic. In Motif applications, you can set the layout direction using the `XmNlayoutDirection` resource from the VendorShell or MenuShell. The Manager and Primitive widget as well as Gadgets, also have an `XmNlayoutDirection` resource. The default value is inherited from the closest ancestor with the same resource.

In the case of an `XmText` widget, you must specify the vertical direction as well as the horizontal direction. Setting the `layoutDirection` to `XmRIGHT_TO_LEFT` results in the string direction from right to left, but the cursor moves vertically down. If the vertical direction is important and you require top-to-bottom alignment, be sure to specify `XmRIGHT_TO_LEFT_TOP_TO_BOTTOM`. This setting specifies that the components are laid out from right to left first and then top to bottom, and results in the desired behavior.

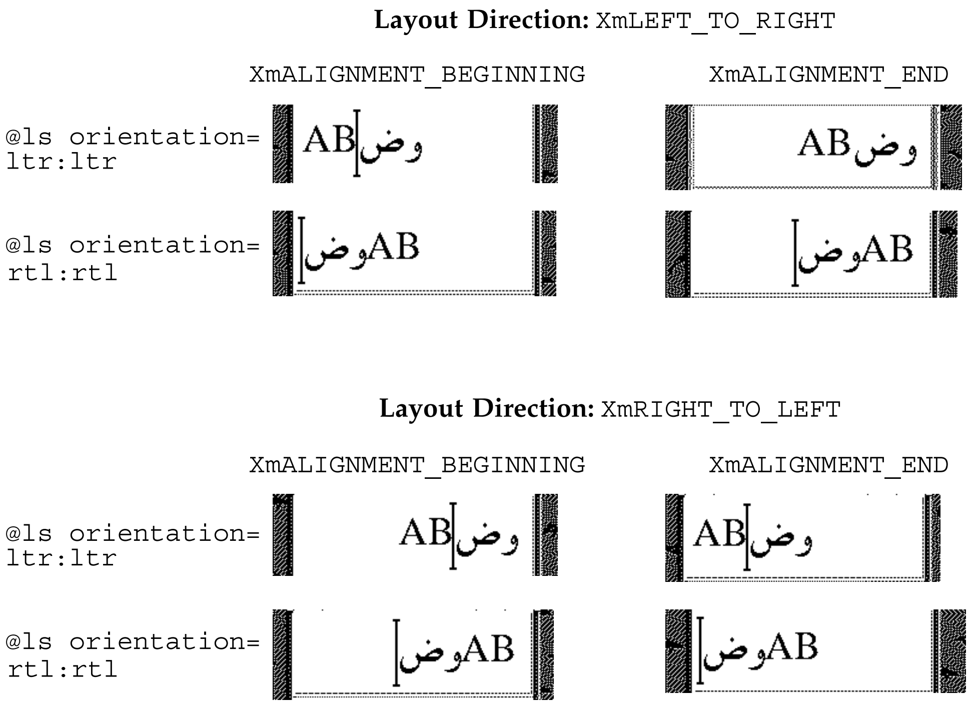
The behavior of the `XmText` and `TextField` widgets is also influenced by the `XmNalignment` and `XmNlayoutModifier` resources of the `XmRendition`. These resources, in addition to `XmNlayoutDirection`, control the layout behavior of the Text widget. This behavior is illustrated in Figure 6–2.

The input string used in the figure is:

A B و ض

The `XmNlayoutModifier` string `@ls orientation=` setting values for the following figure are shown in the left column.

FIGURE 6–2 Layout Direction



As the illustration shows, `XmNalignment` dictates whether the text is flush right or left in conjunction with the layout direction. `XmNlayoutModifier` breaks the text into segments and arranges them left-to-right or right-to-left, depending on the orientation value. In other words, if the `XmNlayoutDirection` is `XmRIGHT_TO_LEFT`, and the `XmNAlignment` value is `XmALIGNMENT_BEGINNING`, the string is flush right.

EXAMPLE 6–1 Creating a Rendition

The following code creates an `XmLabel` whose `XmNlabelString` is of the type `XmCHARSET_TEXT`, using the Rendition whose tag is "ArabicShaped." The Rendition is created with an `XmNlayoutAttrObject` of "ar" (corresponding to the locale name for the Arabic locale) and a layout modifier string that specifies for the output buffer a `Numerals` value of `NUMERALS_CONTEXTUAL` and a `ShapeCharset` value of "iso8859–6".

The locale-specific layout module transforms its input text into an output buffer of physical characters encoded using the 16-bit Unicode codeset. Because an explicit layout locale has been specified, this text is rendered properly independent of the runtime locale setting. In this example, the input is encoded in ISO 8859–6.

```
int n;
Arg args[10];
```

EXAMPLE 6–1 Creating a Rendition *(Continued)*

```
Widget w;
XmString labelString;
XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;
      /* alef lam baa noon taa - iso8859-6 */
labelString = XmStringGenerate("\307\344\310\346\312\", NULL
                                      XmCHARSET_TEXT, "ArabicShaped");
w = XtVaCreateManagedWidget("a label", xmLabelWidgetClass, parent,
                             XmNlabelString, labelString,
                                  XmNlabelType, XmSTRING,
                             NULL);
n = 0;
XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*");
      n++;
XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
XtSetArg(args[n], XmNlayoutAttrObject, "ar"); n++;
XtSetArg(args[n], XmNlayoutModifier,
          "@ls numerals=:contextual, shapecharset=iso8859-6"); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, argcs
s, n);
renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE_MERGE);
XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
```

EXAMPLE 6–2 Editing a Rendition

The following code creates a `TextField` widget and a RenderTable with a single `Rendition`. Both the `XmNlayoutAttrObject` and `XmNlayoutModifier` pseudo resources have been left unspecified and therefore default to `NULL`. This value means that the layout object associated with the Rendition belongs to the default locale, if one exists.

For this example to work properly, the locale must be set to one whose codeset is ISO 8859-6 and whose locale-specific layout module can support the `IMPLICIT_BASIC` algorithm. The Rendition's `LayoutObject`'s `ImplicitAlg` value is modified through the Rendition's `XmNlayoutModifier` pseudo resource.

```
int n;
Arg args[10];
Widget w;
    XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;
w = XmCreateTextField(parent, "text field", args, 0);
n = 0;
    XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*");
     n++;
    XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, args, n);
renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE_MERGE);
```

EXAMPLE 6–2 Editing a Rendition *(Continued)*

```
XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
    ....
n = 0;
XtSetArg(args[n], XmNlayoutModifier, "@ls implicitalg=basic");
     n++;
XmRenditionUpdate(rendition, args, n);
```

Creating a Render Table in a Resource File

Renditions and render tables should be specified in resource files for a properly internationalized application. When the render tables are specified in a file, the program binaries are made independent of the particular needs of a given locale, and can be easily customized to local needs.

Render tables are specified in resource files with the following syntax:
resource_spec`:[`*tag*`[,`*tag*`]*]`

where *tag* is some string suitable for the `XmNtag` resource of a rendition.

This line creates an initial render table containing one or more renditions as specified. The renditions are attached to the specified tags:

resource_spec`[*|.]` *rendition*`[*|.]`*resource_name*`:`*value*

The following example illustrates the CTL resources related to `XmRendition` that can be set using resource files. The `fontType` must be set to `FONT_IS_XOC` for the layout object to take effect. The `layoutModifier` specified using `@ls` is passed on to the layout object by the rendition object.

For a complete list of resources that can be set on the layout object using `layoutModifier`, see *CAE Specification: Portable Layout Services: Context-dependent and Directional Text*, The Open Group: Feb 1997; ISBN 1-85912-142-X; document number C616.

EXAMPLE 6–3 Creating a Render Table in an Application

Before creating a render table, an application program must first have created at least one of the renditions that is part of the table. The `XmRenderTableAddRenditions()` function, as its name implies, is also used to augment a render table with new renditions. To create a new render table, call the `XmRenderTableAddRenditions()` function with a `NULL` argument in place of an existing render table.

The following code creates a render table using a rendition created with `XmNfontType` set to `XmFONT_IS_XOC`.

```
int n;
Arg args[10];
Widget w;
```

EXAMPLE 6–3 Creating a Render Table in an Application *(Continued)*

```
XmString labelString;
XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;
      /* alef lam baa noon taa - iso8859-6 */
labelString = XmStringGenerate("\307\344\310\346\312\", NULL
                                       XmCHARSET_TEXT, "ArabicShaped");
w = XtVaCreateManagedWidget("a label", xmLabelWidgetClass, parent,
                            XmNlabelString, labelString,
                                 XmNlabelType, XmSTRING,
                            NULL);
n = 0;
XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*-*");
     n++;
XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
XtSetArg(args[n], XmNlayoutAttrObject, "ar"); n++;
XtSetArg(args[n], XmNlayoutModifier,
         "@ls numerals=nominal:contextual, shapecharset=iso8859-6"); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, args, n);
renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE);
XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
```

Horizontal Tabs

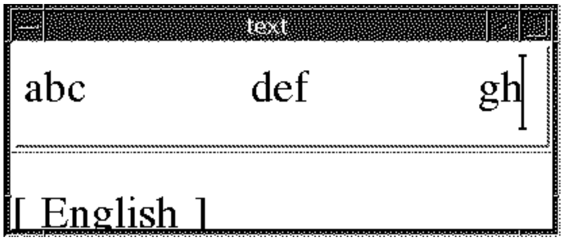
A compound string can contain tab characters that control the placement of text. To interpret those characters on display, a widget refers the a list of tab stops to the rendition in effect for that compound string. However, the dynamic widgets `TextField` and `XmText` do not use the tab resource of the rendition. Instead, the widgets compute the tab width using the formula of `8*(width of character 0)`.

The tab measurement is the distance from the left margin of the compound string display. This distance is measured from the right margin, if the layout direction is right-to-left. Regardless of the direction of the text (Arabic right-to-left or English left-to-right), the tab inserts space to the right or left as specified by the layout direction (`XmNlayoutDirection`).

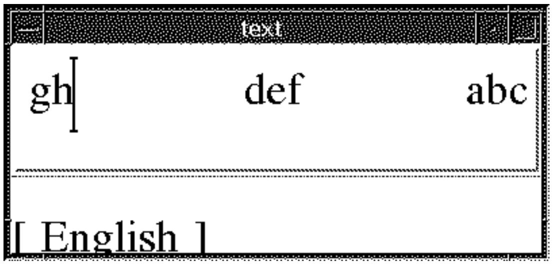
The text following a tab is always aligned at the tab stop. The tab stop is calculated from the start of the widget, which in turn is influenced by `XmNlayoutDirection`. The behavior of the tabs and their interaction with directionality of the text and the `XmNlayoutDirection` of the widget is illustrated in the following figure.

The input for this illustration is `abc\tdef\tgh`.

FIGURE 6–3 Tabbing Behavior



Layout Direction: XmLEFT_TO_RIGHT



Layout Direction: XmRIGHT_TO_LEFT

Mouse Selection

The user makes a primary selection with mouse button 1. Pressing this button deselects any existing selection and moves the insertion cursor and the anchor to the position in the text where the button is pressed. Dragging while holding down mouse button 1 selects all text between the anchor and the pointer position, deselecting any text outside the range.

The text selected is influenced by the resource `XmNeditPolicy`, which can be set to `XmEDIT_LOGICAL` or `XmEDIT_VISUAL`. If the `XmNeditPolicy` is set to `XmEDIT_LOGICAL` and the text selected is bidirectional, the selected text is not contiguous visually and is a collection of segments. The text in the logical buffer does not have a one-to-one correspondence with the display.

As a result, the contiguous buffer of logical characters of bidirectional text is not rendered in a continuous stream of characters. Conversely, when the `XmNeditPolicy` is set to `XmEDIT_VISUAL`, the selected text can be contiguous visually but is segmented in the logical buffer. Therefore, the sequence of selection, deletion, and insertion of bidirectional text at the same cursor point does not result in the same string.

Keyboard Selection

The selection operation available with the mouse is also available with the keyboard. The combination of the Shift and the arrow keys enables the selection of text.

The selected text is influenced by the resource `XmNeditPolicy`, which can be set to `XmEDIT_LOGICAL` or `XmEDIT_VISUAL`. If the `XmNeditPolicy` is set to `XmEDIT_LOGICAL` and the selected text is bidirectional, the selected text is not contiguous visually. Because the text in the logical buffer does not have one-to-one correspondence with the display, the contiguous buffer of logical characters of bidirectional text is not rendered in a continuous stream of characters.

Conversely, when the `XmNeditPolicy` is set to `XmEDIT_VISUAL`, the text selected can be contiguous visually but is segmented in the logical buffer. Therefore, the sequence of selection, deletion, and insertion of bidirectional text at the same cursor point does not result in the same string.

Text Resources and Geometry

The following text resources relate to geometry:

- The render table `XmNrenderTable` that the widget uses to select a font or font set and other attributes in which to display the text.

  The `Text` and `Textfield` widgets can use only the font-related rendition resources, such as `XmNfontType`. These widgets can also specify the attributes of the layout object, such as `XmNlayoutAttrObject`. These widgets usually include a locale identifier, and `XmNlayoutModifier`, which specifies the layout values to be passed through to the Layout Object associated with the XOC associated with this `XmRendition`.
- A resource (`XmNwordWrap`) that specifies whether lines are broken at word boundaries when the text would be wider than the widget.

  Breaking a line at a word boundary does not insert a new line into the text. In the case of cursive languages like Arabic, if the word length is greater than the widget length, the word is wrapped to the next line. However, the first character in the next line is shaped independently of the previous character in the logical buffer.

Porting Instructions

The new Motif library enabled for Complex Text Layout (CTL), is located in `/usr/dt/lib/libXm.so.4`. If your application links to `libXm.so.3` the application does not support CTL. `ldd app_name` shows the library to which the application is linking. To port the existing applications to enable CTL, you need to perform the following steps:

1. Add `-DSUN_CTL` to your `Makefile`.

This flag is important and includes the necessary data structures to support CTL. This value should be set during compilation.

2. Recompile the existing application.

   This recompilation automatically links with the CTL-enabled Motif library `libXm.so.4`.
3. Add the `XmText.translations` resources to your application resource file. Without these resources, the layout engine of the locale does not launch.
4. Refer to the sample application attached to your documentation.

Note – Use the font name that is available and appropriate to your locale in the `fontName` resource.

For example, if you want cell-based character movement (Thai) in `XmTextField` or `XmText` widgets, set the translations of the corresponding widgets as follows:

```
XmText.translations: #override \n\

<Key>osfRight:forward-cell() \n\

<Key>osfLeft:backward-cell() \n\

<Key>osfDelete:delete-next-cell() \n\

<Key>osfBackSpace:delete-previous-cell() \n\
```

CHAPTER 7

Print Filter Enhancement With mp

This chapter describes print enhancement to the mp utility. The chapter discusses the following topics:

Printing for UTF-8

An enhanced mp print filter that can print various input file formats including flat text files written in UTF-8 is available in the current Oracle Solaris environment. This print filter uses TrueType and Type 1 scalable fonts and X11 bitmap fonts available on the Oracle Solaris system. The filter can also make use of printer resident fonts and can act as an X print server client.

The output from the utility is standard PostScript and can be sent to any PostScript printer. The mp utility can also output any page description language when configured as an X Print server client, mp is supported by the print server.

To use the utility, type the following command:

```
system% mp filename | lp
```

You can also use the utility as a filter, since mp accepts stdin stream:

```
system% cat filename | mp | lp
```

You can set the utility as a printing filter for a line printer. For example, the following command sequence tells the printer service LP that the printer lp1 accepts only mp format files. This command also installs the printer lp1 on port /dev/ttya. See the lpadmin(1M) man page for more details.

```
system# lpadmin -p lp1 -v /dev/ttya -I MP
system# accept lp1
system# enable lp1
```

Using lpfilter(1M), you can add the utility for a filter as follows:

```
system# lpfilter -f lp1 -F pathname
```

The command tells LP that a converter (in this case, mp) is available through the filter description file named *pathname*. *pathname* contains the following information:

```
Input types: simple
Output types: MP
Command: /usr/bin/mp
```

The filter converts the default type file input to PostScript output using /usr/bin/mp.

To print a UTF-8 text file, use the following command:

```
system% lp -T MP UTF-8-file
```

Refer to the mp(1) man page for more detail.

mp Print Filter Enhancement Overview

The mp print filter is enhanced in the current Oracle Solaris release. The latest mp can work internally in three different modes to produce the output file in a locale to print international text. The available modes are:

- Working with the locale-specific font configuration file mp.conf
- Working with the locale-specific PostScript prolog file prolog.ps
- Working as an Xprt (X Print Server) client

The following sections describe when to use a specific printing method and which configuration and supporting files are used by mp for these printing methods.

Using mp With the Locale-Specific Font Configuration File mp.conf

If the -D or -P option is not given in the command line, this printing method is the default method, unless the prolog.ps file is present in either of the/usr/openwin/lib/locale/$LANG/print or /usr/lib/lp/locale/$LANG/mp directories. The prolog.ps file forces mp to print using PostScript embedded fonts in the file. Even if a prolog.ps exists in a locale, using the -M option ignores the prolog.ps file and uses an mp.conf file instead, if one exists.

This method uses the /usr/lib/lp/locale/$LANG/mp/mp.conf font configuration file. You probably do not need to change this file unless you need to print using alternate fonts. This file can be configured with TrueType, Type 1, or .pcf fonts.

Using mp With the Locale-Specific PostScript Prolog Files

The /usr/lib/lp/locale/C/ directory contains .ps print page layout files common for this mode of printing. A description of how to customize these files is provided in "Adding and Customizing prolog Files" on page 162

If the -D or -P option is not given in the command line, and /usr/openwin/lib/locale/$LANG/print/prolog.ps exists, then the prolog.ps file is prepended to the output. Depending upon the print style of the .ps prolog page, the layout file is also prepended to the output.

This method of printing makes use of PostScript font files only. Customization of prolog.ps files is described in "Adding and Customizing prolog Files" on page 162.

Using mp as an Xprt (X Print Server) Client

Using mp as an Xprt client enables mp to print the output of any printer connected to a network supported by an Xprt print service. As an Xprt client, mp supports PostScript and many versions of PCL.

The Xprt client attempts a connection to an Xprt server based on the following rules:

- When the -D *printer_name@machine[:dispNum]* or -P *printer_name@machine[:dispNum]* options are used with the mp command, mp attempts to connect to an Xprt print service on *machine[:dispNum]* with *printer_name*.

  If the above attempted connection to *machine[:dispNum]* fails or if the argument given to -D or -P is just *printer_name*, then the mp command checks the XPSERVERLIST for Xprt servers that support the *printer_name* argument. For example:

  ```
  system% setenv XPSERVERLIST "machine1[:dispNum1] machine2[:dispNum2] ..."
  ```
- If no server is found using above rules, mp checks for an XPDISPLAY environment variable set to *machine[:dispNum]*. For example:

  ```
  system% setenv XPDISPLAY "machine[:dispNum]"
  ```

- If the XPDISPLAY variable is not set or if the variable is invalid, mp tries to connect to the default display :2100. If the default display value is also invalid, mp exits with an error message.

  The /usr/lib/lp/locale/C/mp directory contains .xpr print page sample layout files for Xprt client. The sample files are for 300 dpi printers. If the target printer has a different dpi value, the dpi value of the sample files is automatically converted to the resolution of the target printer.

Localization With the mp.conf Configuration File

Configuration files provide the flexibility for adding or changing font entries, or font group entries.

The system default configuration file is /usr/lib/lp/locale/$LANG/mp/mp.conf where $LANG is a locale environment variable in the locale in which printing occurs. You can specify a personal configuration file with the -u *config.file path* option.

A ligature or variant glyph that has been encoded as a character for compatibility is called a *presentation form*. The mp.conf file is used mainly for mapping the intermediate code points in a locale to the presentation forms in the encoding of the font that is used to print that code point.

Intermediate code points can either be wide characters, or output of the Portable Layout Services (PLS) layer. Complex Text Layout printing requires the intermediate code points to be PLS output. The default intermediate code generated by mp is PLS output.

Font formats currently supported are Portable Compiled Format (PCF), TrueType, and Type1 format. Both system-resident and printer-resident Type1 fonts are supported. Keep in mind the following information about the format and contents of the mp.conf configuration file:

- Lines must begin with a valid keyword (directive).
- Arguments to a keyword must appear on the same line as the keyword.
- Lines that begin with a # character are treated as comments until the end of the line.
- Numeric arguments that begin with 0x are interpreted as a hexadecimal number.

The different sections in the mp.conf file include:

- Font aliasing
- Font group definition
- Mapping from the intermediate code ranges to the font group in a locale
- Associating each font with the shared object that maps the intermediate code points to the presentation forms in the font encoding

Font Aliasing

The font aliasing section of the `mp.conf` file is used to define alias names for each font used for printing. Each line in this section is of the following form:

`FontNameAlias` *font-alias-name font-type font-path*

font-alias-name
: The usual convention for aliasing a font name is to specify the encoding/script name of the font followed by a letter that indicates whether the font is Roman, Bold, Italic, or BoldItalic (R, B, I or BI).

  For example,`/usr/openwin/lib/X11/fonts/75dpi/courR18.pcf.Z,` because it is an iso88591 Roman font, can be assigned the alias name iso88591R.

font-type
: Possible values are `PCF` for `.pcf` fonts, `Type1` for Adobe Type1 fonts, and `TrueType` for TrueType fonts. Only these three kinds of fonts can be configured in this `mp.config` file.

font-path
: The absolute path name for the font files. For Type1 printer-resident fonts, just specify the font name, such as Helvetica.

For example,

```
FontNameAlias   prnHelveticaR   Type1   Helvetica
```

Font Group Definition

You can combine same-type fonts to form a font group. The format of the font group is as follows:

keyword	`FontGroup`.
fontgroupname	The group name for the fonts.
GroupType	The font type. Create font groups for the same type of fonts only (PCF, Type1, TrueType).
Roman	The Roman font name in the font group.
Bold	The Bold font name in the font group.
Italic	The Italic font name in the font group.
BoldItalic	The BoldItalic font name in the font group.

For creating a group, only a Roman font entry is required. The Bold, Italic, and BoldItalic fonts are optional. The different types of fonts are used to display the header lines for mail or news articles, for example. If only the Roman font is defined, that font is used in place of other fonts.

Mapping Section

The mapping section of the `mp.conf` files maps from the intermediate code ranges to the font group in a locale. The format for each line in this section is as follows.

keyword	MapCode2Font.
range_start	A 4–byte hexadecimal value, starting with `0x`, that indicates the start of the code range to map to one or more font groups.
range_end	Indicates the end of the code range to map. If the values is '-', only a single intermediate code point is mapped to the target font.
group	A Type1, PCF, or TrueType font group with which the presentation forms are to be printed.

Association Section

The association section of the `mp.conf` file associates each font with the shared object that maps the intermediate code points to the presentation forms in the font encoding. The format for each line in this section is as follows:

keyword	`CnvCode2Font`.
font alias name	The alias name defined for the font.
mapping function	Takes in the intermediate code and returns presentation forms in font encoding, which is in turn used to get the glyph index and draw the glyph.
file path having mapping function	The `.so` file name that contains the mapping function. You can use the utility in `dumpcs` to ascertain the intermediate code set for EUC locales.

Note – The current TrueType engine used by `mp` (1) can work only with format 4 and PlatformID 3 `cmap`. You can only configure Microsoft `.ttf` files. Additionally, the character map encoding has to be Unicode or Symbol for the TrueType font engine to work correctly. Because most of the `.ttf` fonts in the Oracle Solaris environment obey these restrictions, you can map all TrueType fonts in Oracle Solaris software within the `mp.conf` file.

You can create a shared object that maps a font to correspond with a PCF `type1` X Logical Fonts Description (XLFD). You can then create a shared object that maps from the intermediate code range to the encoding specified by XLFD. For example:

```
-monotype-arial-bold-r-normal-bitmap-10-100-75-75-p-54-iso8859-8
```

The corresponding PCF font is:

```
/usr/openwin/lib/locale/iso_8859_8/X11/fonts/75dpi/ariabd10.pcf.Z
```

This font is encoded in ISO 8859-8, so shared objects have to map between intermediate code and corresponding ISO 8859-8 code points.

If a TrueType font with XLFD:

```
-monotype-arial-medium-r-normal--0-0-0-0-p-0-iso8859-8
```

has the corresponding font:

```
/usr/openwin/lib/locale/iso_8859_8/X11/fonts/TrueType/arial__h.ttf
```

you should map between the intermediate code and Unicode, because the `cmap` encoding for the previous TrueType font is in Unicode. In the example of this TrueType font, suppose a sample intermediate code in the `en_US.UTF-8` locale that corresponds to a Hebrew character (produced by the PLS layer) is `0xe50000e9`. Because the font is Unicode encoded, design the function within the corresponding `.so` module in such a way that when you are passing `0xe50000e9`, the output corresponds to presentation form in Unicode. The example here is `0x000005d9`.

The function prototype for the mapping function should be:

```
unsigned int function(unsigned int inter_code_pt)
```

The following are optional keyword/value pairs that you can use in `mp.conf`:

```
PresentationForm        WC/PLSOutput
```

The default value is `PLSOutput`. If the user specifies `WC`, then the intermediate code points that are generated are wide characters. For CTL printing, this default value should be used.

If the locale is a non-CTL locale and the keyboard value is PLSOutput, that value is ignored and the `mp` generates wide-character codes instead.

You can use the optional keyword/value pairs listed in the following table if the locale supports CTL. These variables can assume any of the possible values given in the middle column of the table.

TABLE 7–1 Optional Keyword/Value Pairs

Optional Keyword	Optional Value	Default

`Orientation`	`ORIENTATION_LTR/`
	`ORIENTATION_RTL/`
	`ORIENTATION_CONTEXTUAL`
		`ORIENTATION_LTR`

TABLE 7–1 Optional Keyword/Value Pairs *(Continued)*

Optional Keyword	Optional Value	Default
`Numerals`	`NUMERALS_NOMINAL/` `NUMERALS_NATIONAL/` `NUMERALS_CONTEXTUAL`	`NUMERALS_NOMINAL`
`TextShaping`	`TEXT_SHAPED/` `TEXT_NOMINAL/` `TEXT_SHFORM1/` `TEXT_SHFORM2/` `TEXT_SHFORM3/` `TEXT_SHFORM4`	`TEXT_SHAPED`

▼ How to Add a Printer-Resident Font

The example in the following procedure illustrates how to add a new PCF, TrueType, or Type1 printer-resident font to the configuration file.

Complete this procedure to replace the currently configured font. In the first two steps, a PCF font used to display the characters in the range `0x00000021` - `0x0000007f` is replaced with a TrueType font.

1 **Before you add a new font, look at various components in the configuration file that correspond to the currently configured font.**

```
FontNameAlias iso88591R  PCF  /usr/openwin/lib/X11/fonts/75dpi/courR18PCF.Z
FontNameAlias iso88591B  PCF  /usr/openwin/lib/X11/fonts/75dpi/courB18PCF.Z
.
.
.
FontGroup       iso88591         PCF       iso88591R iso88591B
.
.
.
MapCode2Font    0x00000020      0x0000007f      iso88591
.
.
.
CnvCode2Font iso88591R _xuiso88591 /usr/lib/lp/locale/$LANG/mp/xuiso88591.so
CnvCode2Font iso88591B _xuiso88591 /usr/lib/lp/locale/$LANG/mp/xuiso88591.so
```

For example, you could map the `/usr/openwin/lib/locale/ja/X11/fonts/TT/HG-MinchoL.ttf` fonts to the `en_US.UTF-8` locale. Because `HG-MinchoL.ttf` is a Unicode TrueType font file, you use the `.so` module mapping function to directly return the incoming `ucs-2` code points.

```
unsigned short _ttfjis0201(unsigned short ucs2) {
                 return(ucs2);
         }
```

a. **Save the mapping to the `ttfjis0201.c` file.**

b. **Create a shared object file.**

   ```
   cc -G -Kpic -o ttfjis0201.so ttfjis0201.c
   ```

2 **To map a PCF file, such as `/usr/openwin/lib/locale/ja/X11/fonts/75dpi/gotmrk20.pcf.Z`, check the following encoding that corresponds to XLFD in the `/usr/openwin/lib/locale/ja/X11/fonts/75dpi/fonts.dir` file.**

```
-sun-gothic-medium-r-normal--22-200-75-75-c-100-jisx0201.1976-0
```

a. **For `jisx0201` encoding, prepare a shared object that maps from `ucs-2` to `jisx0201`. Obtain the mapping table for creating the `.so` module. For a Unicode locale, find the character set mappings to Unicode in the `ftp.unicode.org/pub/MAPPINGS/` directory.**

b. **Use these mappings to write a `xu2jis0201.c` file:**

   ```
   unsigned short _xu2jis0201(unsigned short ucs2) {
                            if(ucs2 >= 0x20 && ucs2 <= 0x7d )
                                    return (ucs2);
                            if(ucs2==0x203e)
                                    return (0x7e);
                            if(ucs2 >= 0xff61 && ucs2 <= 0xff9f)
                                    return (ucs2 - 0xff60 + 0xa0);
                           return(0);
                    }
   ```

c. **When you create a mapping file, include all the `usc–2` to `jisx0201` cases.**

   ```
   cc  -G -o xu2jis0201.so xu2jis0201.c
   ```

▼ How to Create a Shared Object File

The examples in the following procedure how you how to create shared object files.

1 **To add a font, edit the lines of the following example that correspond to sections of the `mp.conf` file.**

This example shows how to add the TrueType font. The `.so` path points to the `xu2jis0201.so` file.

```
FontNameAlias   jis0201R TrueType /home/fn/HG-Minchol.ttf
FontGroup     jis0201 TrueType jis0201R
MapCode2Font  0x0020     0x007f  jis0201
CnvCode2Font   jis0201R      _ttfjis0201 <.so path>
```

Note – To add a PCF font, change the keyword from TrueType to PCF.

---

2 **Invoke the `mp` command with the changed `mp.conf` file to print the range `0x0020-0x007f` in the new font.**

You can map other Japanese character ranges with the same `.so` file, For example, you could map the range `0x0000FF61 0x0000FF9F`.

Note – To maintain backward compatibility, you can use the `/usr/openwin/lib/locale/$LANG/print/prolog.ps` file to create output in the current locale. When you use the `prolog.ps` file, no configuration file is required.

You can find a sample `mp.conf` file in the `/usr/lib/lp/locale/en_US.UTF-8/mp` directory.

Adding and Customizing `prolog` Files

The `prolog` files can be divided into two main categories:

- PostScript `prolog` files `(.ps)`
- X print server client `prolog` files`(.xpr)`.

PostScript File Customization

The PostScript files fall into the following categories:

- Common `prolog` file
- Print layout `prolog` files

Locale-Dependent `prolog` Files

The purpose of the `prolog.ps` file is to set up non-generic fonts. Applications use these predefined PostScript font names for printing. The `prolog` file must define at least the following font names for Desk Set Calendar manager and `mp`:

- LC_Times-Roman
- LC_Times-Bold
- LC_Helvetica
- LC_Helvetica-Bold
- LC_Courier
- LC_Helvetica-BoldOblique
- LC_Times-Italic

The following example uses these fonts to print the particular local character set specified:

```
100 100 moveto
/LC_Times-Roman findfont 24 scale font setfont
(Any text string in your locale) show
```

The Oracle Solaris localization kit provides a sample `prolog.ps` file for the Japanese environment. Alternatively, this file is found in the `/usr/openwin/lib/locale/ja/print/` directory.

The following example shows how to add or change composite fonts in an existing `prolog.ps` file.

```
%
(Foo-Fine) makecodeset12
(Base-Font) makeEUCfont
%
```

You could define a composite font called `LC_Base-Font`, for example. `LC_Base-Font` might be a composite of a *Foo-Fine* font that contains a locale character set and a *Base-Font*. You do not need in-depth knowledge of PostScript programming to add or change a font.

The best way to create a `prolog.ps` file is to study the example version. In the example `prolog.ps`, two routines need to be written: `makecodeset12` and `makeEUCfont`. The routine `makecodeset12` sets the local font-encoding information. This routine might differ from locale to locale. The routine `makeEUCfont` combines the base font and the locale font to form a composite font. The creator of the `prolog` file should have good knowledge of PostScript in order to write `makecodeset12` and `makeEUCfont`.

The `prolog.ps` file support is reserved for backward compatibility only. Do not create a new `prolog.ps` file for generating printed output for a locale. Use `mp.conf` instead.

The path for `prolog.ps` file is

```
/usr/openwin/lib/locale/$LANG/print/prolog.ps
```

Common PostScript `prolog` Files

The common `prolog` file is `mp.common.ps`.

Every other page layout `prolog` file needs to include this file.

The `mp.common.ps` file resides in the `/usr/lib/lp/locale/C/mp/` directory. This file contains a PostScript routine to re-encode a font from the standard encoding to the ISO 8859–1 encoding. The `.reencodeISO` routine is called from the print layout `prolog` files to change encoding of the fonts. Usually this `prolog` file does not need any customization. If you create your own `prolog` file, set the environment variable `MP_PROLOGUE` to point to the directory that contains the modified `prolog` files.

Print Layout `prolog` Files

The print layout `prolog` files, `mp.*.ps` files, contain routines for controlling the page layout for printing. In addition to issuing a header and a footer for a print page with user name, print date, and page number, these `prolog` files can provide other information. For example, the `prolog` files can give effective print area dimensions and landscape and portrait mode of printing to be used.

The Print Layout `prolog` files are:

- `mp.pro.ps`
- `mp.pro.alt.ps`
- `mp.pro.fp.ps`
- `mp.pro.ps`
- `mp.pro.ts.ps`
- `mp.pro.altl.ps`
- `mp.pro.ff.ps`
- `mp.pro.l.ps`
- `mp.pro.ll.ps`
- `mp.pro.tm.ps`

A set of standard functions needs to be defined in every `prolog` file. These functions are called when a new print page starts, a print page ends, or a new column ends. The implementations of these functions define the print attributes of the printout.

The following PostScript variables are defined at runtime by the `mp` binary. All the print layout files can use these variables for printing dynamic information such as `user name`, `subject`, `print time`. This information taken from the variables normally appears in the header or footer of the print page.

User	The name of the user who is running `mp`, obtained from the system `passwd` file.
MailFor	Variable used to hold the name of the type of article to print. The possible values for this variable are: ▪ Listing for – When the input is a text file ▪ Mail for – When the input is a mail file ▪ Article from – When the input is an article from a news group
Subject	The subject taken from the mail and news headers. You can use the `-s` option to force a subject to the mail and news files as well as to normal text files.
Timenow	The time of print that appears in the header and footer. This information is taken from the `localtime()` function.

The following functions are implemented in print layout `prolog` files. All of these functions can use subfunctions.

`endpage`	Usage: `page_number endpage` Called when the bottom of a printed page is reached. This function restores the graphic context of the page and issues a `showpage`. In some `prolog` files the header and footer information is displayed in a page-by-page mode rather than in a column-by-column mode. You can implement this function to call subfunctions that display the header and footer gray-scale lozenges.
`newpage`	Usage: `page_number newpage` Routines or commands to be executed when a new page begins. Setting landscape print mode, saving the print graphic context, and translating the page coordinates are some of the functions for these routines.
`endcol`	Usage: `page_number col_number endcol` Used to display header and footer information, move to the new print position, and so forth.

To add new print layout `prolog` files, you need to define the following variables explicitly within the print layout `prolog` file:

NumCols	Number of columns in a print page. Default is 2.
PrintWidth	Width of print area in inches. Default is 6.
PrintHeight	Height of print area in inches. Default is 9.

`.xpr` Files

These files are located by default at `/usr/lib/lp/locale/C/mp/`. An `.xpr` file corresponds to each PostScript `prolog` layout file except the `mp.common.ps` file. You can define an alternate `prolog` directory by defining the `MP_PROLOGUE` environment variable.

These files work as keyword/values pairs. Lines that start with # are considered comments. Spaces separate different tokens unless explicitly stated. Three main sections for each `.xpr` file are bound by the following keyword pairs:

- `STARTCOMMON/ENDCOMMON`
- `STARTPAGE/ENDPAGE`
- `STARTCOLUMN/ENDCOLUMN`

- `STARTFORCEDPAGE/ENDFORCEDPAGE`
- `STARTFORCEDCOLUMN/ENDFORCEDCOLUMN`

Certain keyword/value pairs can be used in these three areas. Each area is described in the following section.

`STARTCOMMON/ENDCOMMON` Keywords

All the keyword/value pairs that appear after the `STARTCOMMON` keyword and before the `ENDCOMMON` keyword define general properties of the print page. Different valid values for a keyword are separated by using a slash (/) character.

`ORIENTATION 0/1`
: 0 means the printing occurs in portrait and 1 means in landscape.

`PAGELENGTH` *unsigned-integer*
: A value that indicates the number of lines per logical page.

`LINELENGTH` *unsigned-integer*
: A value that indicates the number of single-column characters per line.

`NUMCOLS` *unsigned-integer*
: The number of logical pages per physical page.

`HDNGFONTSIZE` *unsigned-integer*
: The heading-font point size in decipoints.

`BODYFONTSIZE` *unsigned-integer*
: The body-font point size in decipoints.

`PROLOGDPI` *unsigned-integer*
: The dots-per-inch scale in which the current `.xpr` file is created.

`YTEXTBOUNDARY` *unsigned-integer*
: This y-coordinate establishes the boundary for text printing in a page or logical page (column). This boundary is used as an additional check to see whether text printing is occurring within the expected area. This boundary is needed for Complex Text Layout and EUC printing, as character height information obtained from corresponding fonts can be wrong.

`STARTTEXT` *unsigned-integerunsigned-integer*
: The decipoint x/y points where the actual text printing starts in the first logical page in a physical page.

`PAGESTRING 0/1`
: The 1 indicates that a page string needs to be appended before the page number in the heading.

  0 indicates that only the page number is displayed.

`EXTRAHDNGFONT` *font string 1*, *font string 2*, ... *font string n*

The font strings are X Logical Font Descriptions. The token that separates the keyword `EXTRAHDNGFONT` from the comma-separated font name list is a quote " character, not a space or tab. These fonts are given preference over the built-in fonts when the heading is printed. Usually, `EXTRABODYFONT` is used to assign printer-resident fonts that are configured in the `/usr/openwin/server/etc/XpConfig/C/print/models/<model name>/fonts` directory.

The `fonts.dir` file contains the XLFD of the printer-resident fonts.

In the `.xpr` file, a font usually is specified as shown in the following example:

```
 "-monotype-Gill Sans-Regular-r-normal- -*-%d-*-*-p-0-iso8859-2"
```

The `%d`, if present, is replaced by `mp` to the point size of the current heading fonts in the `.xpr` file. The x resolution and y resolution are specified by `*`. The average width field is set as `0` to indicate selection of a scalable font, if possible. You can also provide more specific font names.

`EXTRABODYFONT` *font string 1*, *font string 2*, ... *font string n*

The same as `EXTRAHDNGFONT`, except that these fonts are used to print the page body.

`XDISPLACEMENT` *signed/unsigned int*

Provides the x coordinate displacement to be applied to the page for shifting the contents of the page in the x direction. This displacement can be a +ve or -ve value.

`YDISPLACEMENT` *signed/unsigned int*

The same as x displacement, except that the shifting happens in the y direction.

These two keywords are useful when you deal with some printers that have nonstandard margin widths that require you to shift the printed contents in a page.

`STARTPAGE/ENDPAGE` Keywords

The keyword value pairs in this section are bound by `STARTPAGE` and `ENDPAGE` keywords. This section contains drawing and heading information that is to be applied for a physical page. A physical page can contain many logical pages, but all the drawing routines that are contained between these keywords are applied only once to a physical page.

The valid drawing entities are `LINE` and `ARC`. The `XDrawLine()` and `XDrawArc()` functions are executed on values of these keywords.

The dimensions within this section are mapped in PROLOGDPI units. Angles are in degrees.

`LINE x1 y1 x2 y2`	The x/y unsigned coordinates define a pair of points for connecting a line.
`ARC x y width height angle1 angle2`	x and y are both unsigned integers that represent the arc origin. Width and height are unsigned integers that represent the width and height of the arc.

`USERSTRINGPOS x y`	Unsigned coordinates represent the position in which the user information is printed on the heading.
`TIMESTRINGPOS x y`	Unsigned coordinates represent the position in which the time for printing is printed on the heading.
`PAGESTRINGPOS x y`	Unsigned coordinates represent the position to print the page string for each printed page.
`SUBJECTSTRINGPOS x y`	Unsigned coordinates represent the position to print the subject in the page.

`STARTFORCEDPAGE/ENDFORCEDPAGE` Section

When the `-n` option is given to `mp`, all the decorations given within a `STARTPAGE/ENDPAGE` section do not print. However, everything included within a `STARTFORCEDPAGE/ENDFORCEDPAGE` section prints even if the `-n` option is given.

`STARTCOLUMN/ENDCOLUMN` Section

All keywords are the same as described in "`STARTPAGE/ENDPAGE` Keywords" on page 167 except that the entries in this section are applied `NUMCOLS` times to a physical page. If `NUMCOLS` is 3, then the printable area of the physical page is divided into three, and lines, arcs, or heading decorations appear three times per page.

`STARTFORCEDCOLUMN/ENDFORCEDCOLUMN` Section

When the `-n` option is given to `mp`, all the decorations given within a `STARTCOLUMN/ENDCOLUMN` section do not print. However, everything included within a `STARTFORCEDCOLUMN/ENDFORCEDCOLUMN` section prints even if the `-n` option is given.

Creating a New `.xpr` File

When you create a new `.xpr prolog` file, you specify only the values that differ from the default.

The following table lists the `mp` program defaults for different keywords if these values are not specified in the `.xpr` file for the `STARTCOMMON/ENDCOMMON` section:

TABLE 7–2 `STARTCOMMON/ENDCOMMON` Keyword Values

Keyword	Value
`ORIENTATION`	`0`
`PAGELENGTH`	`60`
`LINELENGTH`	`80`

TABLE 7–2 STARTCOMMON/ENDCOMMON Keyword Values *(Continued)*

Keyword	Value
YTEXTBOUNDARY	3005
NUMCOLS	01
HDNGFONTSIZE	120
PROLOGDPI	300
STARTTEXT	135 280
PAGESTRING	0

No default values are needed for the other two sections bound by STARTPAGE/ENDPAGE and STARTCOLUMN/ENDCOLUMN.

To create a page with no decoration, use four logical pages per physical page in portrait format. Specify the following sections and values:

```
STARTCOMMON
NUMCOLS 04
LINELENGTH 20
ENDCOMMON
```

When you create a page with no decoration, you do not need to specify the following two sections:

```
STARTPAGE/ENDPAGE
STARTCOLUMN/ENDCOLUMN
```

These parameters are not needed if you are not putting decorations on the printed page. All the coordinates are in 300 dpi by default unless you are not specifying the PROLOGDPI keyword. If the target printer resolution is different, the .xpr file is scaled to fit into that resolution by the program.

Before you create an .xpr file, you must know the paper dimensions. For U.S. paper, 8.5x11 inches, for a printer of resolution 300 dpi, 2550X3300 are the total dimensions. Most printers cannot print from the top left corner of the paper. Instead, some margin space is assigned around the physical paper. Even if you try to print from 0,0 the printing will not be in the top left corner of the page. Consider this limitation when you create a new .xpr file.

◆ ◆ ◆ APPENDIX A

Compose and Dead Key Input

This appendix describes the compose key sequences in different input modes and the dead key input.

How to Use Compose and Dead Key Input

To insert characters with diacritical marks or special characters from Latin-1, Latin-2, Latin-4, Latin-5, and Latin-9, you must type a Compose key sequence, as described in the following examples.

To display the Ä character:

1. Press and release the Compose key.
2. Press Shift and the A key simultaneously. Release Shift-A.
3. Press and release the ” key.

To display the ¿, character:

1. Press and release the Compose key.
2. Press and release the ? key.
3. Press and release the ? key.

When there is no Compose key available on your keyboard, you can emulate its operation by simultaneously pressing the Control key and the Shift key.

For the input of the Euro currency symbol (Unicode value U+20AC) from the locale, you can use any one of following input sequences:

- AltGraph and E together
- AltGraph and 4 together
- AltGraph and 5 together

With these input sequences, you press both keys simultaneously. If no AltGraph key is available on your keyboard, you can use certain alternative euro sign input sequences such as Compose e = or Compose c =.

The following tables show the most commonly used compose sequences for Latin-1, Latin-2, Latin-3, Latin-4, Latin-5, and Latin-9 script input for the Oracle Solaris operating system.

The following table lists the common Latin-1 Compose key sequences.

TABLE A–1 Common Latin-1 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
spacebar	spacebar	no-break space
s	1	superscripted 1
s	2	superscripted 2
s	3	superscripted 3
!	!	inverted exclamation mark
x	o	currency symbol ¤
p	!	paragraph symbol ¶
/	u	mu u
'	"	acute accent
,	, (comma)	cedilla Ç
"	"	diaeresis
-	^	macron
o	o	degree °
x	x	multiplication sign x
+	-	plus-minus ±
-	-	soft hyphen –
-	:	division sign ÷
-	a	ordinal (feminine) ª
-	o	ordinal (masculine) º
-	, (comma)	not sign ¬
.	.	middle dot ·
1	2	vulgar fraction ½

TABLE A–1 Common Latin-1 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
1	4	vulgar fraction ¼
3	4	vulgar fraction ¾
<	<	left double angle quotation mark «
>	>	right double angle quotation mark »
?	?	inverted question mark ¿
A	‘ (backquote)	A grave À
A	' (single quote)	A acute Á
A	*	A ring above Å
A	"	A diaeresis Ä
A	^	A circumflex Â
A	~	A tilde Ã
A	E	AE diphthong Æ
C	, (comma)	C cedilla Ç
C	o	copyright sign ©
D	-	Capital eth ð
E	‘ (backquote)	E grave È
E	'	E acute É
E	"	E diaeresis Ë
E	^	E circumflex Ê
I	‘ (backquote)	I grave Ì
I	’	I acute Í
I	"	I diaeresis Ï
I	^	I circumflex Î
L	-	pound sign £
N	~	N tilde Ñ
O	‘ (backquote)	O grave Ò
O	’	O acute Ó
O	/	O slash Ø

TABLE A–1 Common Latin-1 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
O	"	O diaeresis Ö
O	^	O circumflex Ô
O	~	O tilde Õ
R	0	registered mark ®
T	H	Thorn þ
U	‘ (backquote)	U grave Ù
U	'	U acute Ú
U	"	U diaeresis Ü
U	^	U circumflex Û
Y	'	Y acute ý
Y	-	yen sign ¥
a	‘ (backquote)	a grave à
a	'	a acute á
a	*	a ring above å
a	"	a diaeresis ä
a	~	a tilde ã
a	^	a circumflex â
a	e	ae diphthong æ
c	, (comma)	c cedilla ç
c	/	cent sign ¢
c	o	copyright sign ©
d	-	eth ð
e	‘ (backquote)	e grave è
e	'	e acute é
e	"	e diaeresis ë
e	^	e circumflex ê
i	‘ (backquote)	i grave ì
i	'	i acute í

TABLE A–1 Common Latin-1 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
i	"	i diaeresis ï
i	^	i circumflex î
n	~	n tilde ñ
o	‘ (backquote)	o grave ò
o	'	o acute ó
o	/	o slash ø
o	"	o diaeresis ö
o	^	o circumflex ô
o	~	o tilde õ
s	s	German double s ß also known as sharp S
t	h	thorn þ
u	‘ (backquote)	u grave ù
u	'	u acute ú
u	"	u diaeresis ü
u	^	u circumflex û
y	'	y acute y
y	"	y diaeresis ÿ
|	|	broken bar ¦

The following table lists the common Latin-2 Compose key sequences.

TABLE A–2 Common Latin-2 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
k	k	kra
A	_	A macron
E	_	E macron
E	.	E dot above
G	,	G cedilla

TABLE A–2 Common Latin-2 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
`I`	`_`	I macron
`I`	`~`	I tilde
`I`	`a`	I ogonek
`K`	`,`	K cedilla
`L`	`,`	L cedilla
`N`	`,`	N cedilla
`O`	`_`	O macron
`R`	`,`	R cedilla
`T`	`|`	T stroke
`U`	`~`	U tilde
`U`	`a`	U ogonek
`U`	`_`	U macron
`N`	`N`	Eng
`a`	`_`	a macron
`e`	`_`	e macron
`e`	`.`	e dot above
`g`	`,`	g cedilla
`i`	`_`	i macron
`i`	`~`	i tilde
`i`	`a`	i ogonek
`k`	`,`	k cedilla
`l`	`,`	l cedilla
`n`	`,`	n cedilla
`o`	`_`	o macron
`r`	`,`	r cedilla
`t`	`|`	t stroke
`u`	`~`	u tilde
`u`	`a`	u ogonek

TABLE A–2 Common Latin-2 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
u	_	u macron
n	n	eng

The following table lists the common Latin-3 Compose key sequences.

TABLE A–3 Common Latin-3 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
C	>	C circumflex
C	.	C dot above
G	>	G circumflex
G	.	G dot above
H	>	H circumflex
J	>	j circumflex
S	>	S circumflex
U	u	U breve
c	>	c circumflex
c	.	c dot above
g	>	g circumflex
g	.	g dot above
h	>	h circumflex
j	>	j circumflex
s	>	s circumflex
u	u	u breve

The following table lists the common Latin-4 Compose key sequences.

TABLE A–4 Common Latin-4 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
k	k	kra

TABLE A–4 Common Latin-4 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
A	_	A macron
E	_	E macron
E	.	E dot above
G	,	G cedilla
I	_	I macron
I	~	I tilde
I	a	I ogonek
K	,	K cedilla
L	,	L cedilla
N	,	N cedilla
O	_	O macron
R	,	R cedilla
T	|	T stroke
U	~	U tilde
U	a	U ogonek
U	_	U macron
N	N	Eng
a	_	a macron
e	_	e macron
e	.	e dot above
g	,	g cedilla
i	_	i macron
i	~	i tilde
i	a	i ogonek
k	,	k cedilla
l	,	l cedilla
n	,	n cedilla
o	_	o macron

TABLE A–4 Common Latin-4 Compose Key Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
r	,	r cedilla
t	|	t stroke
u	~	u tilde
u	a	u ogonek
u	_	u macron
n	n	eng

The following table lists the common Latin-5 Compose key sequences.

TABLE A–5 Common Latin-5 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
G	u	G breve
I	.	I dot above
g	u	g breve
i	.	i dotless

The following table lists the Common Latin-9 Compose key sequences.

TABLE A–6 Common Latin-9 Compose Key Sequences

Press Compose, then Press and Release	Press and Release	Result
o	e	Ligature oe
O	E	Ligature OE
Y	"	Y diaeresis

If you are using a keyboard that has accent dead keys, use the following compose key sequences. The `dead_acute` key name comes from the X11 registered `keysym` names of `X_dead_acute` as shown at `/usr/openwin/include/X11/keysymdef.h`. The `SunFA_Circum` and such key names come from `X11 keysym` names such as `SunXK_FA_Circum` shown at `/usr/openwin/include/X11/Sunkeysym.h`.

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys

Press and Release	Press and Release	Result
dead_grave	spacebar	grave accent
dead_acute	apostrophe	acute accent
dead_acute	spacebar	apostrophe
dead_diaeresis	double quote	diaeresis
dead_diaeresis	spacebar	diaeresis
dead_circumflex	spacebar	circumflex accent
dead_circumflex	slash	vertical line
dead_circumflex	0	degree sign
dead_circumflex	1	superscript one
dead_circumflex	2	superscript two
dead_circumflex	3	superscript three
dead_circumflex	period	middle dot
dead_circumflex	exclamation point	broken bar
dead_circumflex	minus	macron
dead_circumflex	underscore	macron
dead_cedilla	comma	cedilla
dead_cedilla	minus	not sign
dead_tilde	spacebar	tilde
dead_grave	A	A with grave
dead_acute	A	A with acute
dead_circumflex	A	A with circumflex
dead_tilde	A	A with tilde
dead_diaeresis	A	A with diaeresis
dead_grave	a	a with grave
dead_acute	a	a with acute
dead_circumflex	a	a with circumflex
dead_tilde	a	a with tilde

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
dead_diaeresis	a	a with diaeresis
dead_cedilla	C	C with cedilla
dead_cedilla	c	c with cedilla
dead_grave	E	E with grave
dead_acute	E	E with acute
dead_circumflex	E	E with circumflex
dead_diaeresis	E	E with diaeresis
dead_grave	e	e with grave
dead_acute	e	e with acute
dead_circumflex	e	e with circumflex
dead_diaeresis	e	e with diaeresis
dead_grave	I	I with grave
dead_acute	I	I with acute
dead_circumflex	I	I with circumflex
dead_diaeresis	I	I with diaeresis
dead_grave	i	i with grave
dead_acute	i	i with acute
dead_circumflex	i	i with circumflex
dead_diaeresis	i	i with diaeresis
dead_tilde	N	N with tilde
dead_tilde	n	n with tilde
dead_grave	O	O with grave
dead_acute	O	O with acute
dead_circumflex	O	O with circumflex
dead_tilde	O	O with tilde
dead_diaeresis	O	O with diaeresis
dead_grave	o	o with grave
dead_acute	o	o with acute

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
dead_circumflex	o	o with circumflex
dead_tilde	o	o with tilde
dead_diaeresis	o	o with diaeresis
dead_cedilla	S	S with cedilla
dead_cedilla	s	s with cedilla
dead_grave	U	U with grave
dead_acute	U	U with acute
dead_circumflex	U	U with circumflex
dead_diaeresis	U	U with diaeresis
dead_grave	u	u with grave
dead_acute	u	u with acute
dead_circumflex	u	u with circumflex
dead_diaeresis	u	u with diaeresis
dead_acute	Y	Y with acute
dead_acute	y	y with acute
dead_diaeresis	y	y with diaeresis
SunFA_Grave	spacebar	grave accent
SunFA_Grave	A	A with grave
SunFA_Grave	a	a with grave
SunFA_Grave	E	E with grave
SunFA_Grave	e	e with grave
SunFA_Grave	I	I with grave
SunFA_Grave	i	i with grave
SunFA_Grave	O	O with grave
SunFA_Grave	o	o with grave
SunFA_Grave	U	U with grave
SunFA_Grave	u	u with grave
SunFA_Acute	apostrophe	acute accent

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
SunFA_Acute	spacebar	apostrophe
SunFA_Acute	A	A with acute
SunFA_Acute	a	a with acute
SunFA_Acute	C	C with acute
SunFA_Acute	c	c with acute
SunFA_Acute	E	E with acute
SunFA_Acute	e	e with acute
SunFA_Acute	I	I with acute
SunFA_Acute	i	i with acute
SunFA_Acute	L	L with acute
SunFA_Acute	l	l with acute
SunFA_Acute	N	N with acute
SunFA_Acute	n	n with acute
SunFA_Acute	O	O with acute
SunFA_Acute	o	o with acute
SunFA_Acute	R	R with acute
SunFA_Acute	r	r with acute
SunFA_Acute	S	S with acute
SunFA_Acute	s	s with acute
SunFA_Acute	U	U with acute
SunFA_Acute	u	u with acute
SunFA_Acute	Y	Y with acute
SunFA_Acute	y	y with acute
SunFA_Acute	Z	Z with acute
SunFA_Acute	z	z with acute
SunFA_Cedilla	comma	cedilla
SunFA_Cedilla	minus	not sign
SunFA_Cedilla	C	C with cedilla

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
SunFA_Cedilla	c	c with cedilla
SunFA_Cedilla	G	G with cedilla
SunFA_Cedilla	g	g with cedilla
SunFA_Cedilla	K	K with cedilla
SunFA_Cedilla	k	k with cedilla
SunFA_Cedilla	L	L with cedilla
SunFA_Cedilla	l	l with cedilla
SunFA_Cedilla	N	N with cedilla
SunFA_Cedilla	n	n with cedilla
SunFA_Cedilla	R	R with cedilla
SunFA_Cedilla	r	r with cedilla
SunFA_Cedilla	S	S with cedilla
SunFA_Cedilla	s	s with cedilla
SunFA_Cedilla	T	T with cedilla
SunFA_Cedilla	t	t with cedilla
SunFA_Circum	spacebar	circumflex accent
SunFA_Circum	0	degree sign
SunFA_Circum	1	superscript one
SunFA_Circum	2	superscript two
SunFA_Circum	3	superscript three
SunFA_Circum	exclamation point	broken bar
SunFA_Circum	minus	macron
SunFA_Circum	underscore	macron
SunFA_Circum	period	middle dot
SunFA_Circum	slash	vertical line
SunFA_Circum	A	A with circumflex
SunFA_Circum	a	a with circumflex
SunFA_Circum	C	C with circumflex

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
SunFA_Circum	c	c with circumflex
SunFA_Circum	E	E with circumflex
SunFA_Circum	e	e with circumflex
SunFA_Circum	G	G with circumflex
SunFA_Circum	g	g with circumflex
SunFA_Circum	H	H with circumflex
SunFA_Circum	h	h with circumflex
SunFA_Circum	I	I with circumflex
SunFA_Circum	i	i with circumflex
SunFA_Circum	J	J with circumflex
SunFA_Circum	j	j with circumflex
SunFA_Circum	O	O with circumflex
SunFA_Circum	o	o with circumflex
SunFA_Circum	S	S with circumflex
SunFA_Circum	s	s with circumflex
SunFA_Circum	U	U with circumflex
SunFA_Circum	u	u with circumflex
SunFA_Diaeresis	double quote	diaeresis
SunFA_Diaeresis	spacebar	diaeresis
SunFA_Diaeresis	A	A with diaeresis
SunFA_Diaeresis	a	a with diaeresis
SunFA_Diaeresis	E	E with diaeresis
SunFA_Diaeresis	e	e with diaeresis
SunFA_Diaeresis	I	I with diaeresis
SunFA_Diaeresis	i	i with diaeresis
SunFA_Diaeresis	O	O with diaeresis
SunFA_Diaeresis	o	o with diaeresis
SunFA_Diaeresis	U	U with diaeresis

TABLE A–7 Compose Key Sequences Based on Accent Dead Keys *(Continued)*

Press and Release	Press and Release	Result
SunFA_Diaeresis	u	u with diaeresis
SunFA_Diaeresis	y	y with diaeresis
SunFA_Diaeresis	Y	Y with diaeresis
SunFA_Tilde	spacebar	tilde
SunFA_Tilde	A	A with tilde
SunFA_Tilde	a	a with tilde
SunFA_Tilde	N	N with tilde
SunFA_Tilde	n	n with tilde
SunFA_Tilde	O	O with tilde
SunFA_Tilde	o	o with tilde

The following compose key sequences are supported in the Greek input mode. Some compose key sequences start with accent dead keys. The abbreviation "ordfemenine" stands for feminine ordinal indicator key.

TABLE A–8 Compose Key Sequences in Greek Input Mode

Press and Release	Press and Release	Result
semicolon	a	lowercase Greek_alpha with tonos
semicolon	e	lowercase Greek_epsilon with tonos
semicolon	h	lowercase Greek_eta with tonos
semicolon	i	lowercase Greek_iota with tonos
semicolon	o	lowercase Greek_omicron with tonos
semicolon	y	lowercase Greek_upsilon with tonos
semicolon	v	lowercase Greek_omega with tonos
semicolon	A	uppercase Greek_alpha with tonos
semicolon	E	uppercase Greek_epsilon with tonos
semicolon	H	uppercase Greek_eta with tonos
semicolon	I	uppercase Greek_iota with tonos
semicolon	O	uppercase Greek_omicron with tonos

TABLE A–8 Compose Key Sequences in Greek Input Mode *(Continued)*

Press and Release	Press and Release	Result
semicolon	Y	uppercase Greek_upsilon with tonos
semicolon	V	uppercase Greek_omega with tonos
dead_acute	Greek_alpha	lowercase Greek_alpha with tonos
dead_acute	Greek_epsilon	lowercase Greek_epsilon with tonos
dead_acute	Greek_eta	lowercase Greek_eta with tonos
dead_acute	Greek_iota	lowercase Greek_iota with tonos
dead_acute	Greek_omicron	lowercase Greek_omicron with tonos
dead_acute	Greek_upsilon	lowercase Greek_upsilon with tonos
dead_acute	Greek_omega	lowercase Greek_omega with tonos
dead_acute	Greek_ALPHA	uppercase Greek_alpha with tonos
dead_acute	Greek_EPSILON	uppercase Greek_epsilon with tonos
dead_acute	Greek_ETA	uppercase Greek_eta with tonos
dead_acute	Greek_IOTA	uppercase Greek_iota with tonos
dead_acute	Greek_OMICRON	uppercase Greek_omicron with tonos
dead_acute	Greek_UPSILON	uppercase Greek_upsilon with tonos
dead_acute	Greek_OMEGA	uppercase Greek_omega with tonos
dead_acute	a	lowercase Greek_alpha with tonos
dead_acute	e	lowercase Greek_epsilon with tonos
dead_acute	h	lowercase Greek_eta with tonos
dead_acute	i	lowercase Greek_iota with tonos
dead_acute	o	lowercase Greek_omicron with tonos
dead_acute	y	lowercase Greek_upsilon with tonos
dead_acute	v	lowercase Greek_omega with tonos
dead_acute	A	uppercase Greek_alpha with tonos
dead_acute	E	uppercase Greek_epsilon with tonos
dead_acute	H	uppercase Greek_eta with tonos
dead_acute	I	uppercase Greek_iota with tonos
dead_acute	O	uppercase Greek_omicron with tonos

TABLE A–8 Compose Key Sequences in Greek Input Mode *(Continued)*

Press and Release	Press and Release	Result
dead_acute	Y	uppercase Greek_upsilon with tonos
dead_acute	V	uppercase Greek_omega with tonos
colon	i	lowercase Greek_iota with dialytika
colon	y	lowercase Greek_upsilon with dialytika
colon	I	uppercase Greek_iota with dialytika
colon	Y	uppercase Greek_upsilon with dialytika
dead_diaeresis	i	lowercase Greek_iota with dialytika
dead_diaeresis	y	lowercase Greek_upsilon with dialytika
dead_diaeresis	I	uppercase Greek_iota with dialytika
dead_diaeresis	Y	uppercase Greek_upsilon with dialytika
dead_diaeresis	Greek_iota	lowercase Greek_iota with dialytika
dead_diaeresis	Greek_upsilon	lowercase Greek_upsilon with dialytika
dead_diaeresis	Greek_IOTA	uppercase Greek_iota with dialytika
dead_diaeresis	Greek_UPSILON	uppercase Greek_upsilon with dialytika
semicolon	semicolon	Greek tonos
colon	colon	diaeresis/dialytika
ordfeminine	0	plus-minus sign
ordfeminine	1	section sign
ordfeminine	2	superscript two
ordfeminine	3	superscript three
ordfeminine	5	broken bar
ordfeminine	6	copyright sign
ordfeminine	7	not sign
ordfeminine	8	soft hyphen

TABLE A–8 Compose Key Sequences in Greek Input Mode *(Continued)*

Press and Release	Press and Release	Result
ordfeminine	9	degree sign
ordfeminine	hyphen	vulgar fraction one half
ordfeminine	backslash	pound sign
ordfeminine	braceleft	modifier letter reversed comma
ordfeminine	braceright	modifier letter apostrophe
ordfeminine	bracketleft	left-pointing double angle quotation mark
ordfeminine	bracketright	right-pointing double angle quotation mark
SunFA_Acute	a	lowercase Greek_alpha with tonos
SunFA_Acute	e	lowercase Greek_epsilon with tonos
SunFA_Acute	h	lowercase Greek_eta with tonos
SunFA_Acute	i	lowercase Greek_iota with tonos
SunFA_Acute	o	lowercase Greek_omicron with tonos
SunFA_Acute	y	lowercase Greek_upsilon with tonos
SunFA_Acute	v	Greek_omega with tonos
SunFA_Acute	A	uppercase Greek_alpha with tonos
SunFA_Acute	E	uppercase Greek_epsilon with tonos
SunFA_Acute	H	uppercase Greek_eta with tonos
SunFA_Acute	O	uppercase Greek_omicron with tonos
SunFA_Acute	I	uppercase Greek_iota with tonos
SunFA_Acute	Y	uppercase Greek_upsilon with tonos
SunFA_Acute	V	uppercase Greek_omega with tonos
SunFA_Acute	Greek_alpha	lowercase Greek_alpha with tonos
SunFA_Acute	Greek_epsilon	lowercase Greek_epsilon with tonos
SunFA_Acute	Greek_eta	lowercase Greek_eta with tonos
SunFA_Acute	Greek_iota	lowercase Greek_iota with tonos
SunFA_Acute	Greek_omega	lowercase Greek_omega with tonos

TABLE A–8 Compose Key Sequences in Greek Input Mode *(Continued)*

Press and Release	Press and Release	Result
SunFA_Acute	Greek_omicron	lowercase Greek_omicron with tonos
SunFA_Acute	Greek_upsilon	lowercase Greek_upsilon with tonos
SunFA_Acute	Greek_ALPHA	uppercase Greek_alpha with tonos
SunFA_Acute	Greek_EPSILON	uppercase Greek_epsilon with tonos
SunFA_Acute	Greek_ETA	uppercase Greek_eta with tonos
SunFA_Acute	Greek_IOTA	uppercase Greek_iota with tonos
SunFA_Acute	Greek_OMICRON	uppercase Greek_omicron with tonos
SunFA_Acute	Greek_UPSILON	uppercase Greek_upsilon with tonos
SunFA_Acute	Greek_OMEGA	uppercase Greek_omega with tonos
SunFA_Diaeresis	i	lowercase Greek_iota with dialytika
SunFA_Diaeresis	y	lowercase Greek_upsilon with dialytika
SunFA_Diaeresis	I	uppercase Greek_iota with dialytika
SunFA_Diaeresis	Y	uppercase Greek_upsilon with dialytika
SunFA_Diaeresis	Greek_iota	lowercase Greek_iota with dialytika
SunFA_Diaeresis	Greek_upsilon	lowercase Greek_upsilon with dialytika
SunFA_Diaeresis	Greek_IOTA	uppercase Greek_iota with dialytika
SunFA_Diaeresis	Greek_UPSILON	uppercase Greek_upsilon with dialytika

TABLE A–9 Compose Key Sequences in Greek Input Mode With Three Keys

Press and Release	Press and Release	Press and Release	Result
semicolon	colon	y	lowercase Greek_upsilon with dialytika and tonos
colon	semicolon	y	lowercase Greek_upsilon with dialytika and tonos
semicolon	colon	i	lowercase Greek_iota with dialytika and tonos

TABLE A–9 Compose Key Sequences in Greek Input Mode With Three Keys *(Continued)*

Press and Release	Press and Release	Press and Release	Result
colon	semicolon	i	lowercase Greek_iota with dialytika and tonos
dead_acute	dead_diaeresis	y	lowercase Greek_upsilon with dialytika and tonos
dead_diaeresis	dead_acute	y	lowercase Greek_upsilon with dialytika and tonos
dead_acute	dead_diaeresis	i	lowercase Greek_iota with dialytika and tonos
dead_diaeresis	dead_acute	i	lowercase Greek_iota with dialytika and tonos
dead_acute	dead_diaeresis	Greek_upsilon	lowercase Greek_upsilon with dialytika and tonos
dead_diaeresis	dead_acute	Greek_upsilon	lowercase Greek_upsilon with dialytika and tonos
dead_acute	dead_diaeresis	Greek_iota	lowercase Greek_iota with dialytika and tonos
dead_diaeresis	dead_acute	Greek_iota	lowercase Greek_iota with dialytika and tonos
SunFA_Acute	SunFA_Diaeresis	i	lowercase Greek_iota with dialytika and tonos
SunFA_Diaeresis	SunFA_Acute	i	lowercase Greek_iota with dialytika and tonos
SunFA_Acute	SunFA_Diaeresis	y	lowercase Greek_upsilon with dialytika and tonos
SunFA_Diaeresis	SunFA_Acute	y	lowercase Greek_upsilon with dialytika and tonos
SunFA_Acute	SunFA_Diaeresis	Greek_iota	lowercase Greek_iota with dialytika and tonos
SunFA_Diaeresis	SunFA_Acute	Greek_iota	lowercase Greek_iota with dialytika and tonos
SunFA_Acute	SunFA_Diaeresis	Greek_upsilon	lowercase Greek_upsilon with dialytika and tonos
SunFA_Diaeresis	SunFA_Acute	Greek_upsilon	lowercase Greek_upsilon with dialytika and tonos

TABLE A–10 Compose Key Sequences in Greek Input Mode With Four Keys

Press and Release	Press and Release	Press and Release	Press and Release	Result
semicolon	colon	semicolon	colon	Greek dialytika tonos
colon	semicolon	colon	semicolon	Greek dialytika tonos

APPENDIX B

Language Support Features and Enhancements

This appendix deals with the language enhancement support features introduced in Oracle Solaris 10 over different releases. It contains the following sections:

Input Method Features

This section describes the language support features related to input methods that were added in different versions of Oracle Solaris operating system

Internet Intranet Input Method Framework (IIIMF) Hangul Language Engine

This feature is included in the Solaris 10 10/08 release.

The Hangul LE (Language Engine) is a new Korean input method that enhances user experience. Hangul LE has the following features:

- User-friendly GUI
- More convenient Hangul or Hanja input functionalities

For more information, see the `Hangul LE` help.

`libchewing 0.3.0`

This feature is included in the Solaris 10 5/08 release.

Chewing input method (IM) is based on `libchewing`, which is an open-source library for Traditional Chinese input. `libchewing` is upgraded to the `libchewing` 0.3.0 version. `libchewing` 0.3.0 includes the following features:

- Incompatibility with API/ABI
- UTF-8 based language engine core for common Unicode environment
- Includes the `libchewing-data` subproject
- Zuin fixes and symbol improvements
- New binary form of user hash data to speed up loading and solving hash data corruption
- Improved calculation of internal tree and phone constants
- Revised `tsi.src` for richer phrases and avoiding crashes
- Merge phone and phrase from `CNS11643`
- Improved Han-Yu PinYin to use table-lookup implementation
- Experimental frequency evaluation that recomputes chewing lifetime
- Implementation of the choice mechanism for symbol pairs
- Experimental, memory-mapping based, binary data handling to speed up data loading

Input Method Switcher Enhancement and EMEA Keyboard Layout Emulation Support

This feature is included in the Solaris 10 8/07 release.

The input method switcher application, `gnome-im-switcher-applet`, is replaced with `iiim-panel`, a `stand-alone` `GTK+` application. `iiim-panel` now starts and resides on the GNOME panel automatically when you log in to the Java Desktop System (Java DS) in UTF-8 or Asian locales. `iiim-panel` can also run in the Common Desktop Environment (CDE).

IIIMF supports language engines that emulate the EMEA keyboard layout such as French, Polish or Dutch.

For more information, see the `iiim-properties` online help.

IIIMF and Language Engines

This feature is included in the Solaris 10 11/06 release.

The Internet Intranet Input Method Framework (IIIMF) has been upgraded from revision 10 to revision 12.

This framework provides the following new features:

- Input Method Switcher — Displays input method status and switches input languages. You can add the input method switcher to the Java Desktop System (Java DS) panel. Select Add to Panel -> Utility -> Input Method Switcher to add the input method switcher to the Java DS panel.
- Utility for `iiim-properties` — Supports various input method preferences. Use one of the following methods to start the `iiim-properties` utility:
  - Select Launch -> Preferences -> Desktop Preferences -> Input Methods.
  - Click mouse button 3 on the Input method switcher and select Preference.
  - In the CDE environment, select Tool -> Input Method Preference from the CDE main menu or type `iiim-properties` at the command prompt.

Each language engine has also been upgraded to the IIIMF revision 12 base. The Japanese language engines, ATOK12 and Wnn6, have been updated to ATOK for Oracle Solaris and Wnn8 respectively. ATOK for Oracle Solaris is equivalent to ATOK17. A new Chinese chewing input method has also been added to the IIIMF.

Korean Language Engine With Auxiliary Window Support

This feature is included in the Solaris 10 3/05 release.

Korean users of the Oracle Solaris operating system can benefit from more comprehensive keyboard input method support for the Korean language.

The new Korean Language Engine with auxiliary window support offers Korean users the following auxiliary windows to control and configure the Korean input method (IM).

- User-based preferences within one window.
- A virtual keyboard environment within another window for point-and-click selection of Korean characters.
- Within another window, users can choose the symbols that they need from special characters that are based on code points.
- Organize all the windows within a special palette of control.

This IM supports three separate keyboard layouts: `2 beol sik`, `3 beol sik 390`, and `3 beol sik final`.

Common Transliteration-based Input Method for All Indian Languages

This feature is included in the Solaris 10 3/05 release.

Users who operate within any Unicode (UTF-8) locale of the Oracle Solaris operating system can now easily and intuitively input characters from Indian regional languages. Users who interact with CDE applications, StarOffice, or Mozilla can more easily interact with Indian scripts.

After selecting the transliteration-based input method (IM), users can type phonetic equivalents of Indian language scripts in English. These equivalents are then displayed in the script that is selected, and are correctly shaped and rendered with the help of an underlying layout and shaper module. As transliteration is the most commonly used input method for Indian languages, this support can greatly enhance the usability of the eight Indian scripts that are provided in the Oracle Solaris software.

Wubi Input Method

This feature is included in the Solaris 10 3/05 release.

The Wubi input method (IM) is widely used in China. The encoding rule for Wubi IM is based on the radical or stroke shape of Chinese characters. Users can rapidly type Chinese characters through a standard keyboard rather than through slower, phonetic-based input methods.

Input Method Support for Indic

This language support feature has been added to the Solaris 10 3/05 release.

Input support for Indian regional language keyboards has been added to the Oracle Solaris operating system. Indic language users can type Indic language characters by using their preferred keyboard layouts in the Oracle Solaris operating system.

File Encoding Examiner

This feature is included in the Solaris 10 5/08 release.

The File Encoding Examiner `fsexam` enables you to convert the name of a file, or the contents of a plain-text file, from a legacy character encoding to UTF-8 encoding.The `fsexam` utility include the following new features:

- Encoding list customization
- Encoding auto-detection
- Support for dry runs, log files, batch conversion, file filtering, symbolic files, command line, and special file types like the compress file

For more information, see the `fsexam(1)` and `fsexam(4)` man pages.

More Japanese `iconv` Modules for Unicode

This feature is included in the Solaris 10 8/07 release.

Starting with this release, the following two types of codeset conversions between the Unicode and Japanese codesets have been added:

- In conversion from or to `eucJP`, `PCK` (SJIS), and `ms932`, `iconv` now supports UTF-16, UCS-2, UTF-32, UCS-4 and their fixed endian variants, such as UTF-16BE and UTF-16LE, and UTF-8.
- `iconv` now supports codeset name `eucJP-ms` to provide conversion between Japanese EUC and Unicode in the same way as Windows. All Unicode encoding variants mentioned previously, are also supported with `eucJP-ms`.

For more information, see the `iconv_ja(5)` man page.

Zero-Country Code Keyboard Layout Support

This feature is included in the Solaris 10 8/07 release and newer releases.

This feature provides a new command option `kbd -s language`. This option enables users to configure keyboard layouts in kernel. The Zero-CountryCode keyboard layout feature is particularly useful on SPARC and x86 systems. In prior releases, all "non-self-ID keyboards" were always recognized as US layout keyboard but now, with the `kbd -s` command, it is possible to correctly configure the keyboard layout in kernel

For more information, see the `kbd(1)` man page.

Unicode Version 4.0 Support

This feature is included in the Solaris 10 3/05 release.

Unicode Version 4.0 introduces 1226 new characters over Unicode Version 3.2.

For more information about UTF-8 byte sequences on Unicode 3.2, see Unicode 3.2 and 4.0 Support in "New Internationalization and Localization Features" on page 18.

The UTF-8 character representation has been also changed to a more secure form. The UTF-8 Corrigendum was originally published in the Unicode Version 3.1 and later updated at the Unicode Version 3.2.

This feature also implements the more secure UTF-8 character representation and byte sequences in the `iconv` code conversions and the following OS-level multibyte functions:

- `mbtowc`(3C)
- `mbstowcs`(3C)
- `mbrtowc`(3C)
- `mblen`(3C)
- `mbsrtowcs`(3C)
- `fgetwc`(3C)
- `mbrlen`(3C)

For more information, see Unicode Standard 4.0 (ISBN 0-321-18578-1).

Code Conversions for Internationalized Domain Name Support

This feature is included in the Solaris Express 10/03 release.

Internationalized Domain Name (IDN) enables the use of non-English native language names as host and domain names. To use non-English host and domain names, application developers must convert names into ASCII Compatible Encoding (ACE) names in their applications as specified in the RFC 3490. System administrators and end users must use ACE names in the existing system files and applications where the networking or system administration applications do not support non-English IDNs.

This feature aids in the code conversion. It provids the conversion API with various supported option arguments, a dedicated IDN encoding conversion utility, and `iconv` code conversions. See the following man pages for detail:

- `libidnkit`(3LIB)
- `idn_decodename`(3EXT)
- `idn_decodename2`(3EXT)

- idn_encodename(3EXT)
- idnconv(1)
- iconv_en_US.UTF-8(5)

New iconv Code Conversions

This feature is included in the Solaris Express 11/04 release.

Various new iconv code conversions between single-byte PC and Windows code pages and various Unicode forms have been added. Also, several major Asian code pages and UCS-2LE have been added.

Standard Type Services Framework

This feature is included in the Solaris Express 9/03 release.

The Standard Type Services Framework (STSF) is a pluggable object-based architecture that allows users to access typographically sophisticated text layout and rendering. The pluggable architecture of the framework gives users the ability to use different font rasterization engines and text layout processors to achieve the desired visual representation. The pluggable architecture also manages fonts and enables application-specific fonts to be created. STSF includes both a standalone API and an X server extension to handle rendering on the server side for improved efficiency.

For more information about the project and how to use the API, see http://stsf.sourceforge.net.

Additional Indic Scripts for Support in Unicode Locales

This feature is included in the Solaris 10 3/05 release.

In addition to the current support for Hindi, Oracle Solaris software supports the following Indic scripts:

- Bengali
- Gurmukhi
- Gujarati
- Tamil.
- Malayalam
- Telugu
- Kannada

Speakers of these Indian regional languages have language support in the Oracle Solaris operating system for any of the Unicode locale environments that Oracle Solaris supports.

HKSCS-2001 Support in Hong Kong Locales

This feature is included in the Solaris 10 3/05 release.

HKSCS-2001 is a new version of the Hong Kong Supplementary Character Set (HKSCS). This new version adds 116 characters to the previous HKSC-1999 character set. HKSCS-2001 is supported in these Oracle Solaris Hong Kong locales: zh_HK.BIG5HK and zh_HK.UTF-8.

Index