International Language Environments Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806–6642–06
December, 2001

Adobe PostScript

011025@2471

Contents

Tables

Figures

Examples

Preface

The *International Language Environments Guide* describes internationalization features that are new in the Solaris™ 9 operating environment. It contains important information on how to use this release to build global software products that support various languages and cultural conventions.

This publication describes the basic attributes associated with language enabling, as well as specific features provided by the Solaris platform to facilitate global application development and administration of language services around the world.

Specifically, this preface contains information about:

- "Who Should Use This Guide" on page 17
- "How This Guide Is Organized" on page 18
- "Related Books and Sites" on page 18

Where appropriate, this guide points you to other guides in the documentation set that contain additional or more detailed information on internationalization features in this release. You get pointers to how to order Sun documents, how to order Sun documents online, and the typographic conventions used in the guide.

Who Should Use This Guide

This guide is intended for software developers and administrators who want to design global products and applications for the Solaris 9 operating environment.

This guide assumes knowledge of the C programming language.

All operating system information pertains to the SunOS™ 5.9 operating environment.

How This Guide Is Organized

The chapters in this guide are organized as follows:

- Chapter 1 describes the new internationalization and localization features in Solaris 9, including the introduction of the euro (€) in several countries.
- Chapter 2 describes support for Codeset Independence, CSI, and the APIs in `libc` for the Solaris 9 product.
- Chapter 3 describes the contents of the Solaris 9 localized product, including localizing the multilingual Solaris product, and new keyboard support, including nineteen new keyboards.
- Chapter 4 describes the Asian supported locales, input systems, and character support.
- Chapter 5 covers the `en_US.UTF-8` locales and the internationalization features incorporated into this release. These include the Cyrillic, Greek, Arabic, Hebrew, Hindi, and Thai input methods, as well as those for the Japanese, Korean, and Simplified and Traditional Chinese input modes.
- Chapter 6 describes Complex Text Layout (CTL) extensions which enable Motif APIs to support writing systems that require complex transformations between logical and physical text representations, such as Arabic, Hebrew, and Thai.
- Chapter 7 explains printing support under the Solaris 9 operating environment, with specific information for European and Asian printing, and the `mp`(1) print filter enhancement.
- Appendix A contains lists of tables of available `iconv` conversions.
- Appendix B contains a table of the partial localization package names on the OS CD.
- Appendix C contains tables representing the language packages on the language CD. There are tables for Simplified Chinese, French, German, Italian, Japanese, Korean, Spanish, Swedish, Traditional Chinese, and Shared.

Related Books and Sites

For information to help developers globalize their applications, refer to the Sun Global Application Developer Corner (Sun GADC)

The Sun Global Application Developer Corner is an updated web version of the previously released Sun Global Application Developer Kit 1.0. It is accessible at :http://www.sun.com/developers/gadc

Anyone in the global development community who is building multilingual applications on Sun's Solaris operating environment and the Java platform can now take advantage of this web version, Sun GADC, at no extra cost to them. With the wide corporate migration to the Solaris operating platform, this free web resource further adds value to developers, independent software vendors, and users who need quick and easy access to Sun's globalization information and resources anytime, anywhere without having to purchase and carry a physical kit with them. Developers can find relevant information and resources on how to create global software applications in one place, saving them time and frustration.

Sun's Global Application Developer Corner contains comprehensive internationalization tools and documentation that address various design and development issues encountered while creating global software, including how to test for global compliance and trouble shoot problems.

The site includes testing tools such as Sun Multibyte English (MBE) locale, which allows developers to test their internationalized applications using pseudo English. This has been extremely useful for English-speaking developers who need to test their applications developed in a specific native language. Sun Multibyte English locale is available for free download. Other useful resources include sample references and code in C, White papers on the various international language support found in the Solaris Operating Environment, technical articles, and useful globalization links for quick reference. There is a checklist available for developers to use as an internationalization testing guide, as well as a contact page for you to ask any Sun globalization related queries.

For information about the Java Development Kit, see `http://java.sun.com/j2se/1.3/docs/guide/intl/index.html`

The *Solaris Common Desktop Environment: Programmer's Guide* is also part of the CDE Developer's AnswerBook™ set that is shipped on the Solaris documentation CD.

OSF/Motif Programmer's Guide, Release 1.2 Englewood Cliffs, New Jersey, Prentice-Hall, 1993. The Open Software Foundation's (OSF) Guide describes how to use the OSF/Motif application programming interface to create Motif applications. It presents an overview of Motif widget set architecture, explains the Motif toolkit, and gives models and examples of Motif applications.

This set of books is essential for successfully developing PostScript applications.

The *PostScript Language Reference Manual (Second Edition)* is the standard reference work for PostScript. It is the definitive documentation of every operator, Display PostScript (DPS), Level 1, and Level 2. The book covers the fundamentals of PostScript as a device-independent printing language. The special capabilities for handling fonts and characters in PostScript are explained. The book's Appendix E also explains standard character sets and encoding vectors. It discusses the organization of fonts that are built into interpreters or supplied from other sources.

Programming the Display PostScript System with X is for application developers who are working with X Windows and Display PostScript. The book documents how to write applications that use Display PostScript to produce information for the screen display and the printer output. It describes coding techniques in detail.

The X Window System has been extended with the X Display PostScript system (often described as X/DPS). It uses application-callable libraries on the client side and corresponding extensions on the X server side.

- Tuthill, Bill, and David Smallberg. *Creating Worldwide Software: Solaris International Developer's Guide*, 2nd edition. Mountain View, California, Sun Microsystems Press, 1997. Available through `books@sun.com` and `www.sun.com/books/`. The book offers a general overview of the internationalization process under the Solaris operating environment.

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at `http://www1.fatbrain.com/documentation/sun`.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

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

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

AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Solaris Internationalization Overview

This section discusses some general information about Internationalization and Localization.

- "New Internationalization and Localization Features" on page 24
- "Internationalization and Localization Defined" on page 25
- "What Is a Locale?" on page 27
- "Using Locale Categories for Localization" on page 31
- "Language Word and Letter Differences" on page 35

The Solaris 9 product includes full Unicode 3.1 support, as defined in Unicode and ISO-10646, for selected locales. The Solaris 9 release is a major release for Sun's international markets. It includes a number of new features. All partial locales, including multibyte locales such as Japanese locales, are now available in the base Solaris 9 product.

The Solaris 9 Operating Environment has been designed to speak the languages of the world since its inception. With a pluggable, service-based approach to globalization, the Solaris internationalization architecture makes it easy for development, deployment and management of applications and language services around the world. In one convenient, multilingual product, users benefit from extensive support for 39 different languages and 133 locales, including complex text layout environments needed to support Thai and Hindi, and bi-directional layout environments for languages like Arabic and Hebrew. Software developers can benefit from the convenience of a single multilingual environment to develop once for deployment in any of the world's locales.

The purpose of an internationalization architecture is to provide a foundation for language-independent software development. The Solaris internationalization architecture provides a flexible and pluggable method of handling input methods, character set encodings, codeset conversion and other basic aspects of language services. Developers can choose between the powerful tools already provided, or to customize their own environment. They can deploy applications in multiple language environments without knowing how input methods work or which codeset convertor needs to be enabled, simply by following standard APIs. They are also given the opportunity to customize particular language attributes. If a developer needs to change convertor tables or add a new input method editor, the architecture was designed to provide this. Solaris has the flexibility to meet the needs of any global application developer.

With the release of the source code for the Solaris X globalization framework to the open community in the fall of 2000, developers now have the ability to enhance compatibility and interoperability of global applications by following a common reference implementation while also participating in the evolution of the code base. Solaris takes a codeset independent approach to globalization, so whether users prefer to operate in native encoded environments or instead join the growing world of Unicode, Solaris can accommodate their needs. The Solaris framework gives the power to scale across platforms with a rich set of data converters designed to ensure interoperability between various encodings and various platforms (from Windows, MAC, for example). With powerful globalization tools and standards-based interfaces, Solaris provides developers with an ideal solution to scale their global business.

Solaris also helps multinational corporations scale their server administration worldwide. Unlike competitive platforms, Solaris uses a service-based approach to administration of language services. Solaris was designed for server administrators to enable language services remotely across a worldwide network, regardless of the client system. This client-independent approach allows for easy upgrade of the system without changing client applications. For example, an Arabic-speaking user needing to read an email from an internet cafe in Paris would still be able to read that email in his or her own language without modifying the local client application. Solaris allows administrators to scale their language services quickly around the world.

New Internationalization and Localization Features

The following features are new to the Solaris 9 release:

- Additional Unicode (UTF-8) locale support features for Thailand, India, Hong Kong, Turkey, Egypt, Brazil, Finland, and Belgium (Walloon).
- Latin-3 character support in Unicode locales.
- PCL support in `mp` printing filter.
- Traditional Chinese (Hong Kong) `Big5+HKSCS` locale `(zh_HK.Big5HK).`
- Traditional Chinese (Hong Kong) UTF-8 locale `(zh_HK.UTF-8).`
- Thai UTF-8 locale `(th_TH.UTF-8).`

- Thai ISO8859–11 locale `(th_TH.ISO8859-11).`
- Hindi script support in Unicode locales.
- Hindi UTF-8 locale `(hi_IN.UTF-8).`
- Collation locales for Asian Solaris.
- Enhanced Chinese locales to support GB18030–2000 and HKSCS.
- New zh_CN.GB18030 locale to support the new GB18030–2000 standard.
- HKSCS `iconv` modules.
- New Chinese input methods.
- Thai input methods enhancement.
- Input Method Auxiliary Window support for Asian Solaris.
- Enhanced `Stdudctool -` Support for migration of UDC (User Defined Character) from Microsoft Windows. Localized for all Asian locales.
- Additional Japanese `iconv` modules – Conversions for IBM mainframe codesets and conversions between Unicode and the Shift_JIS for Microsoft codeset.
- Euro currency. Only those locales participating in using the euro use the euro symbol as their national currency symbol. The other ISO8859-15 locales support the euro symbol € .
- Multibyte partial locale – framework of multibyte locale support is included in the base Solaris product.
- Enhanced Unicode `iconv` modules. The `iconv` modules have been added and enhanced for various new Unicode encoding formats and international and *de facto* industry standard codesets.
- Unicode 3.1 support in Unicode locales.
- Support of new `iconv` code conversions for ISO8859–16.

Internationalization and Localization Defined

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 Solaris system is internationalized, providing the infrastructure and interfaces you need to create internationalized software.

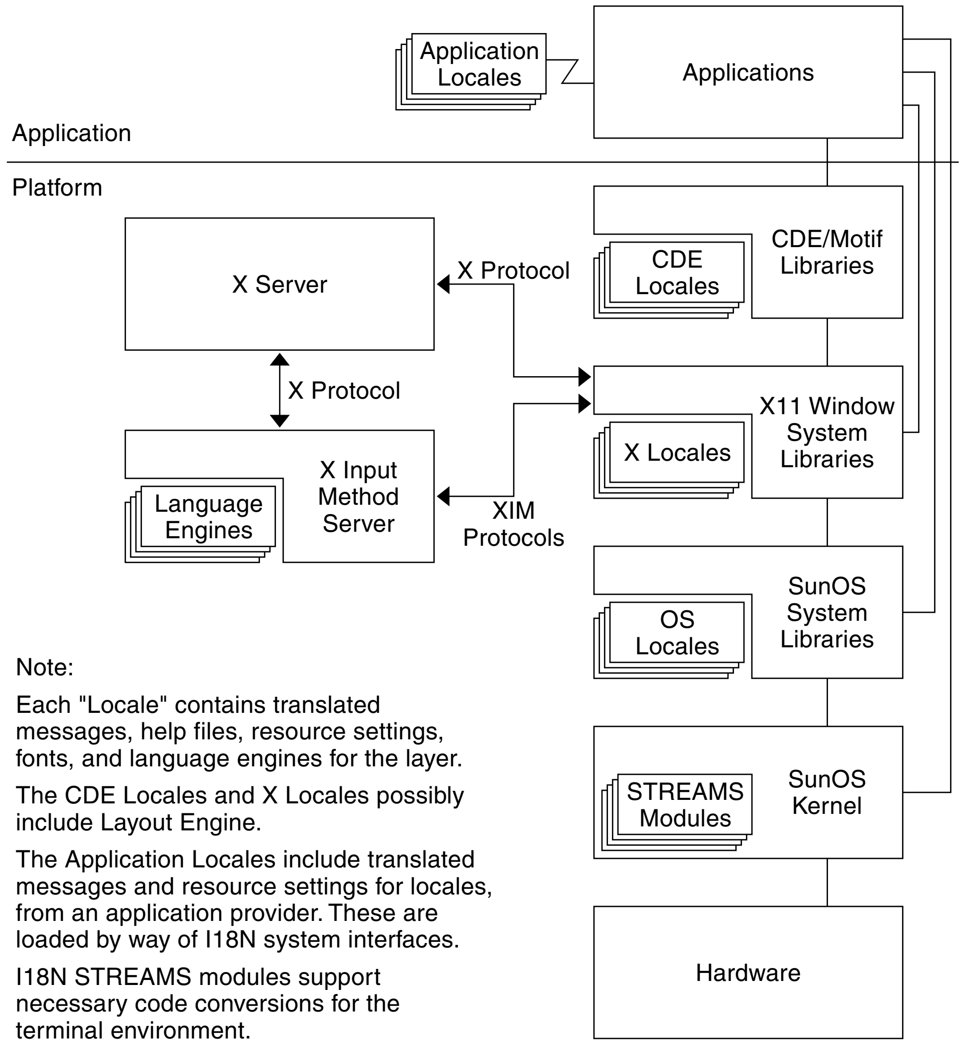
Basic Steps in Internationalization

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

- Use the interfaces described in this book to create software with an environment that can be modified dynamically without the necessity of recompiling the software.
- Divide software into executable code and messages. The messages include all printable and displayable messages that the user sees. 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 in their translated form. 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 described in the following diagram.



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.

FIGURE 1–1 Functions and Structure of Locales in Solaris

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 there is country-dependent formatting or other specifications. A program's locale defines its codesets, date and time formatting conventions, monetary conventions, decimal formatting conventions, and collation (sort) order.

A locale can be composed of both a base language, the country (territory) of use, and 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 allows for specific differences by country, such as currency units 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 `LC_ALL` is set, it overrides not only `LANG`, but all the separate locale categories as well.

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 it from the locale without the modifier.

All 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 as `C`.

Full and Partial Locales

A full Solaris locale has all of the listed functions and the localized system messages in the relevant language. "Partial locales" have no localized messages installed. All locales in the 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 using the Language CD, all of the above locales become "full locales," because they have access to a fully translated desktop. The language CD contains message translations for the following languages:

- German
- French
- Spanish
- Swedish
- Italian
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

All partial locales are available on the Software CD; message translations are available on the Languages CD.

All English locales are also full locales and are available on the Software CD.

Behavior Affected by Locales

Different cultures often use different conventions for writing the date and time, formatting numbers, delimiting words and phrases, and quoting material. Throughout the system, a locale determines the behavior of the following:

- Encoding and processing of text data.
- Identifying the language and encoding of resource files and their text values.
- Rendering and layout of text strings.
- Interchanging text that is used for interclient text communication.
- Encoding and decoding for interclient text communication.
- Selecting the input method (that is, which codeset is generated) and the processing of text data.
- 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 Solaris environment separates language and culture-dependent information from the application and saves it outside the application. By separating the language and culture-dependent information from the application, the developer does not 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 currency symbol for the locale, thousand separator, sign position, the number of fractional digits, and so forth.
`LC_NUMERIC`	Specifies the decimal delimiter (or radix character), the thousand 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, 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) consists of text shaping and directionality.

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 but to the user, they could look awkward, wrong, or even offensive. The following sections describe the elements in the Solaris operating environment that you can control and specify so that you can successfully localize 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
Great Britain	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 (.).

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 different 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. it is a state decision.

Date Formats

The following table shows some of the date formats used around the world. Notice that even within a country, there can be variations.

TABLE 1–2 International Date Formats

Locale	Convention	Example
Canadian (English)	dd/mm/yy	24/08/01
Danish	yyyy-mm-dd	2001–08–24
Finnish	dd.mm.yyyy	24.08.2001
French	dd/mm/yyyy	24/08/2001
German	yyyy-mm-dd	2001–08–24
Italian	dd/mm/yy	24/08/01
Norwegian	dd-mm-yy	24–08–01
Spanish	dd-mm-yy	24-08-01
Swedish	yyyy-mm-dd	2001-08-24
GB-English	dd/mm/yy	24/08/01
US-English	mm-dd-yy	08-24-01
Thai	dd/mm/yyyy	24/08/2001

Numbers

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 G.B. and U.S. 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 G.B.-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

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

Locale	Large Number
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
Swedish	4 294 967 295,00
GB	4,294,967,295.00
US	4,294,967,295.00
Thai	4,294,967,295.00

Note – There are no particular locale conventions that specify how to separate numbers in a list.

Currency

Currency units and presentation order vary greatly around the world. Local and international symbols for currency can differ. The following 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

TABLE 1–4 International Monetary Conventions *(Continued)*

Locale	Currency	Example
Swedish	Krona (Kr)	1 234,56 Kr
GB	Pound (£)	£1,234.56
US	Dollar ($)	$1,234.56
Thai	Baht	2539 Baht
Euro	Euro (€)	€ 5,000

Solaris 9 software supports the euro currency. Local currency symbols are still available for backward compatibility.

TABLE 1–5 User Locales To Support the Euro Currency

Region	Locale Name	ISO Codeset
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
Finland	`fi_FI.ISO8859-15`	8859-1
France	`fr_FR.ISO8859-15`	8859-15
Germany	`de_DE.ISO8859-15`	8859-15
Greece	`el_GR.ISO8859-7`	8859–7
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
Estonia	`et_EE.ISO8859–15`	8859–15
Spain	`es_ES.ISO8859-15`	8859-15
Sweden	`sv_SE.ISO8859-15`	8859-15
Great Britain	`en_GB.ISO8859-15`	8859-15
U.S.A.	`en_US.ISO8859-15`	8859-15

Euro locales are based on the ISO8859–15 codeset.

Keep in mind that a *converted* currency amount can take up more or less space than the original amount. To illustrate: $1,000 can become € 1.307.000.

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

TABLE 1–6 German Locale and Corresponding `LC_MONETARY`

Locale	LC_MONETARY
de_DE.ISO8859–1	DM
de_DE.ISO8859–15	Euro
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. In languages such as Chinese, Japanese and Thai, however, there is often 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, Chinese even more.

Western European Alphabets

The alphabets of most western European countries are similar to the standard 26-character alphabet used in English-speaking countries, but there are often 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, and two phonetic scripts (or syllabaries), hiragana and katakana.

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 been declining 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 because that is what the Japan Industry Standard (JIS) requires. 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 it is possible to completely avoid the use of kanji, most Japanese readers find it hard to understand a text that is composed without any kanji.

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 which are known as jamos. About 3000 characters, or jamos, 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 combining them, 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 they 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 GB2312 (`zh` locale) and more than 20,000 characters in the GBK charset (`zh.GBK` locale). In Taiwan, current standards require more than 13,000 characters; 6000 others have been recently standardized but are considered rare.

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 one 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, and predates the English language by thousands of years. Hebrew is an example of a bidirectional script, in that Hebrew letters are written and read from right to left, while numbers are read from left to right. If any English text is embedded in Hebrew text, it 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 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. There are also no uppercase letters in Hebrew.

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) which consist of three types: consonant letters, independent vowels and dependant vowel signs, and 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, and can combine or change shape depending on their context. Like Hebrew, Hindi text makes no distinction between uppercase or 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.

Note – On SPARC™ 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.

Any keyboard can be used to input characters from any locale since input is handled by the Solaris operating environment.

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 Solaris system provides no support for tracking output page size; this 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 section discusses several internationalization features contained in the Solaris 9 environment.

Support for Codeset Independence

EUC is an abbreviation for Extended Unix Code. The Solaris 9 operating environment supports non-EUC encodings such as PC-Kanji in Japan, Big-5 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 Solaris 9 environment provides a solid framework to enable both EUC and non-EUC codeset support. This support is called *Codeset Independence*, or CSI.

The goal of CSI is to remove EUC dependencies on specific codesets or encoding methods from Solaris OS libraries and commands. The CSI architecture allows the Solaris operating environment to support any UNIX file system safe encoding. CSI supports a number of new codesets, such as UTF-8, PC-Kanji, and Big-5.

CSI Approach

Codeset Independence enables application and platform software developers to keep their code independent of encoding, such as UTF-8, and also provides the ability to adopt any new encoding without having to modify the source code. This architecture approach differs from Java™ internationalization in that Java requires applications to be Unicode-dependent and also requires code conversions throughout the application.

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 codesets. However, the following assumptions about file code encodings (codesets) still apply to the Solaris 9 environment:

- 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

The following lists the CSI-enabled commands in the Solaris 9 environment. The man page for each command has an attribute section which indicates whether or not the command is CSI enabled.

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

/usr/lib/diffh	/usr/xpg4/bin/more	`bdiff`
/usr/sbin/accept	/usr/xpg4/bin/mv	`cancel`
/usr/sbin/reject	/usr/xpg4/bin/nice	`cat`
/usr/ucb/lpr	/usr/xpg4/bin/nohup	`catman`
/usr/xpg4/bin/awk	/usr/xpg4/bin/od	`chgrp`
/usr/xpg4/bin/cp	/usr/xpg4/bin/pr	`chmod`
/usr/xpg4/bin/date	/usr/xpg4/bin/rm	`chown`
/usr/xpg4/bin/du	/usr/xpg4/bin/sed	`cmp`
/usr/xpg4/bin/ed	/usr/xpg4/bin/sort	`col`
/usr/xpg4/bin/edit	/usr/xpg4/bin/tail	`comm`
/usr/xpg4/bin/egrep	/usr/xpg4/bin/tr	`compress`
/usr/xpg4/bin/env	/usr/xpg4/bin/vedit	`cpio`
/usr/xpg4/bin/ex	/usr/xpg4/bin/vi	`csh`
/usr/xpg4/bin/expr	/usr/xpg4/bin/view	`csplit`
/usr/xpg4/bin/fgrep	`acctcom`	`cut`
/usr/xpg4/bin/lp	`apropos`	`diff`
/usr/xpg4/bin/ls	`batch`	`diff3`

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

Solaris 9 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 because they are EUC-dependent functions:

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

In the Solaris 9 product, `libgen (/usr/ccs/lib/libgen.a)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. Therefore, when developing an internationalized application, do not directly access the locale database. Instead, use the internationalization APIs in `libc`, described in "Internationalization APIs in `libc`" on page 47.

Note – When working with the Solaris 9 environment, use the locale databases that are included with the Solaris 9 product. Do not use locales from previous Solaris versions.

Process Code Format

The process code format, which is also known as wide-character code format in the Solaris 9 product, is private and subject to change in a future release. Therefore, when developing an international application, do not assume the process code format is the same. Instead, use the internationalization APIs in `libc` described in "Internationalization APIs in `libc`" on page 47.

Note – However, it is guaranteed that all Unicode locales' process code is in UTF-32 representation. For more detail on the UTF-32, refer to "The Unicode Standard Version 3.0" 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 two or three 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/EUC 9899:1990/Amendment 1, added new internationalization features, collectively known as Multibyte Support Environment (MSE), by defining additional internationalization APIs, notably for multibyte codesets 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 in their own right. 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 be written to handle multibyte characters using the same programming model that is used for single-byte characters.

Dynamically Linked Applications

Solaris 9 product users can choose how to 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 other than C and POSIX using the `setlocale` function will fail. Statically linked applications can be 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, if you type:

```
% /usr/bin/ldd /sbin/sh
```

the command indicates the `/sbin/sh` command is not a dynamically linked program.

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

If you type:

```
% /usr/bin/ldd /usr/bin/ls
```

the command displays the following message:

```
libc.so.1 =>     /usr/lib/libc.so.1
libdl.so.1 => /usr/lib/libdl.so.1
```

This message indicates that the `/usr/bin/ls` command has been dynamically linked with two libraries, `libc.so.1` and `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`.

For more information on filters see the *Linker and Libraries Guide*.

The following two lists show the stub entry points in `libw` and `libintl`. The first is for `libw`, and the second is for `libintl`.

fgetwc	iswpunct	wscncat	wscoll
fgetws	iswspace	wcsncmp	wscpy
fputwc	iswupper	wcsncpy	wscspn
fputws	iswxdigit	wcspbrk	wsdup
getwc	putwc	wcsrchr	wslen
getwchar	putwchar	wcsspn	wsncasecmp
getws	putws	wcstod	wsncat
isenglish	strtows	wcstok	wsncmp
isideogram	towlower	wcstol	wsncpy
isnumber	towupper	wcstoul	wspbrk
isphonogram	ungetwc	wcswcs	wsprintf
isspecial	watoll	wcswidth	wsrchr
iswalnum	wcscat	wcsxfrm	wsscanf
iswalpha	wcschr	wctype	wsspn
iswcntrl	wcscmp	wcwidth	wstod
iswctype	wcscoll	wscasecmp	wstok
iswdigit	wcscpy	wscat	wstol
iswgraph	wcscspn	wschr	wstoll
iswlower	wcsftime	wscmp	wstostr
iswprint	wcsclen	wscol	wsxfrm

This shorter list contains 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 Solaris 9 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 Solaris 9 environment functions also support XPG4 semantics. Refer to the ctype(3C) man page for details.

Internationalization APIs in libc

The Solaris 9 environment offers two sets of APIs:

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

There are many benefits of making applications to process in wide-character codes, since they are fixed-width unit of logical entities, without having to consider 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(3S) and fwscanf(3S) or by using conversion functions like mbtowc(3C) and mbsrtowcs(3C) after the input. To convert output data from wide-character format to multibyte character format, use output functions like fwprintf(3S) and fprintf(3S), or apply conversion functions like wctomb(3C) and wcsrtombs(3C) before the output.

The following tables describe the internationalization APIs included in the Solaris 9 product.

This table describes the messaging function APIs in libc.

TABLE 2–1 Messaging Function in libc

Library Routine	Description
catclose()	Close a message catalog

TABLE 2–1 Messaging Function in `libc` *(Continued)*

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

This table describes the code conversion function APIs in `libc`.

TABLE 2–2 Code Conversion in `libc`

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

This table describes the regular expression APIs in `libc`.

TABLE 2–3 Regular Expressions in `libc`

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

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

TABLE 2–4 Wide Character Class in `libc`

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

This 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

This table lists the query locale data in `libc`

TABLE 2–6 Query Locale Data in `libc`

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

This table describes the character classification function APIs in `libc`.

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

Library Routine	Description
`isalpha()`	Is character alphabetic?
`isupper()`	Is character uppercase?
`islower()`	Is character lowercase?
`isdigit()`	Is character a digit?
`isxdigit()`	Is character a hex digit?
`isalnum()`	Is character alphabetic or digital?
`isspace()`	Is character a space?
`ispunct()`	Is character a punctuation mark?
`isprint()`	Is character printable?
`iscntrl()`	Is character a control character?

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

Library Routine	Description
`isascii()`	Is character an ASCII character?
`isgraph()`	Is character a visible character?
`isphonogram()`	Is wide character a phonogram?
`isideogram()`	Is wide character an ideogram?
`isenglish()`	Is wide character in English alphabet from a supplementary codeset?
`isnumber()`	Is wide character a digit from a supplementary codeset?
`isspecial()`	Is special wide character from a supplementary codeset?
`iswalpha()`	Is wide character alphabetic?
`iswupper()`	Is wide character uppercase?
`iswlower()`	Is wide character lowercase?
`iswdigit()`	Is wide character a digit?
`iswxdigit()`	Is wide character a hex digit?
`iswalnum()`	Is wide character an alphabetic character or digit?
`iswspace()`	Is wide character a white space?
`iswpunct()`	Is wide character a punctuation mark?
`iswprint()`	Is wide character a printable character?
`iswgraph()`	Is wide character a visible character?
`iswcntrl()`	Is wide character a control character?
`iswascii()`	Is wide character an ASCII character?
`toupper()`	Convert a lowercase character to uppercase.
`tolower()`	Convert an uppercase character to lowercase.
`towupper()`	Convert a lowercase wide character to uppercase.
`towlower()`	Convert an uppercase wide character to lowercase.
`towctrans()`	Wide-character mapping

This 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

This 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

This 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.

This 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
`mbrlen()`	Get number of bytes in character (restartable)
`mbsinit()`	Determine conversion object status
`mbrtowc()`	Convert a character to a wide-character code (restartable)

TABLE 2–11 Multibyte Handling in `libc` *(Continued)*

Library Routine	Description
`mbsrtowcs()`	Convert a character string to a wide-character string (restartable)
`mblen()`	Get number of bytes in a character
`mbtowc()`	Convert a character to a wide-character code
`mbstowcs()`	Convert a character string to a wide-character string

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

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

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

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

Library Routine	Description
`wcrtomb()`	Convert a wide-character code to a character (restartable)
`wcstol()`	Convert wide-character string to long integer
`wcstoul()`	Convert wide-character string to unsigned long integer
`wcstod()`	Convert wide-character string to double precision
`wcsrtombs()`	Convert a wide-character string to a character string (restartable).
`wcscat()`	Concatenate wide-character strings.

This 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
`wsprintf()`	Generate wide-character string according to format
`wsscanf()`	Formatted input conversion
`fwprintf()`	Print formatted wide-character output
`fwscanf()`	Convert formatted wide-character input
`wprintf()`	Print formatted wide-character output
`wscanf()`	Convert formatted wide-character input
`swprintf()`	Print formatted wide-character output
`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

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

TABLE 2–14 Wide Strings`libc`

Library Routine	Description
`wscasecmp()`	Compare wide-character strings, ignore case differences.
`wsncasecmp()`	Process code-string operations
`wcsstr()`	Find a wide-character substring

TABLE 2–14 Wide Strings`libc` *(Continued)*

Library Routine	Description
`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

This 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
`getwchar()`	Get multibyte character from `stdin`, convert to wide character
`fgetws()`	Get multibyte string from stream, convert to wide character
`getws()`	Get multibyte string from `stdin`, convert to wide character
`fputwc()`	Convert wide character to multibyte character, puts to stream
`fwide()`	Set stream orientation
`putwchar()`	Convert wide character to multibyte character, puts to `stdin`
`fputws()`	Convert wide character to multibyte string, puts to stream
`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, see the `xgettext`(1), *xgettext(1)* and `msgfmt`(1) man pages, respectively.

User Defined and User Extensible Code Conversions

Solaris users can create user-defined codeset converters by using the `geniconvtbl` utility.

This permits 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 Solaris codeset conversions is also supported.

More details and also examples can be found from `geniconvtbl`(1) and `geniconvtbl`(4) man pages. Sample input source files for the utility are also available for reference from the `/usr/lib/iconv/geniconvtbl/srcs/` directory.

Once the user-defined code conversions are prepared and placed as specified in the `geniconvtbl`(1) man page, users can use the code conversions from the `iconv`(1) utility and the `iconv`(3C) functions of both 32-bit and 64-bit Solaris operating environments.

CHAPTER 3

Localization in the Solaris 9 Environment

This section discusses several localization features contained in the Solaris 9 environment.

Software Support for Localization

This section contains information about the Solaris 9 locale packages, localization content on the Solaris 9 CD-ROMs, localization functions in the interfaces, and script enabling.

Summary of the Solaris 9 Locale Packages

All Solaris 9 locale packages are classified into two categories. The first category is for partial locales, which are the enablers of the locales. With partial locales installed on the system, users can run applications on the target locales, while the OS/GUI messages from Solaris are English. All partial locale packages are available on the Solaris OS CDs.

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

- German
- French
- Spanish
- Swedish
- Italian
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

Full locale packages are available on the languages CD. Partial locale packages (locale enablers) must be installed in order for the full locales to be functional.

The new partial locales for this release are the addition of UTF-8 locales for Russian and Polish, two new locales for Catalan, a new Thai locale, a new Indic locale, two new Traditional Chinese locales, and a new Simplified Chinese locale. The locale names are.

- `ar_EG.UTF-8`
- `ca_ES.ISO8859-1`
- `ca_ES.ISO8859-15`
- `fi_FI.UTF-8`
- `fr_BE.UTF-8`
- `pl_PL.UTF-8`
- `pt_BR.UTF-8`
- `ru_RU.UTF-8`
- `tr_TR.UTF-8`
- `th_TH.UTF-8`
- `hi_IN.UTF-8`
- `zh_HK.BIG5HK`

  This is a Traditional Chinese (Hong Kong) Big5–HKSCS locale. It is a full locale if the Traditional Chinese message packages are installed from the language CD.
- `zh_HK.UTF-8`

  This is a Traditional Chinese (Hong Kong) UTF—8 locale. It is a full locale if the Traditional Chinese message packages are installed from the language CD
- `zh_CN.GB18030.`

  This is a Simplified Chinese GB18030–2000 locale. It is a full locale if the Simplified Chinese message packages are installed from the language CD

Localization Content on Solaris 9 CD-ROMs

Partial locales are selected at the beginning of the install procedure on the OS CD-ROM. Full locales are automatically installed from the Language CD-ROM according to the locale selections made at the beginning of the install procedure.

The distribution of locales is shown in the following table.

TABLE 3–1 Solaris 9 Installation CD-ROMs

Disk	Contents
Solaris OS CD-ROM	Solaris 9 Operating System with all partial locales
Language CD-ROM	Message translations for many languages with locale specific utilities

As mentioned, the locales include partial locales. These are based on core locales for the main language. For example, the `fr_CA.ISO8859-1` (French Canadian) locale is based on the `fr_FR.ISO8859-1` (French) locale. These partial locales utilize the messages that are delivered into its parent locale (French for `fr_CA`). A partial locale contains only English messages.

Localization Functions in Solaris Interfaces

The OS 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 X input method and X output method so that the X11 applications can allow local text input and display. Fonts are provided to enable applications to display characters from various languages.

CDE/Motif is built on top of the X11 window system. Hence, it can utilize the X11 locale capability through X11 APIs. 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.

Script Enabling for the Solaris 9 Environment

The Solaris 9 base product provides multiple levels of script enabling, such as simple ASCII support, Latin/European support, Asian multibyte support, and Arabic/Hebrew bidirectional support.

The interfaces defined within the X/Open specification are capable of supporting a large set of languages and territories, including the types of script listed in the following table:

TABLE 3–2 Script

Script	Description
Latin Language	Americas, Eastern/Western Europe, Turkish
Greek	Greek
East Asia	Japanese, Korean, and Chinese
Indic	Thai
Bidirectional	Arabic and Hebrew
Cyrillic	Russian

Localization in the Multilingual Solaris Product

The multilingual Solaris 9 product is a super set of the base Solaris product. It includes all partial locales (including multibyte locales) that provide the functionality needed to input, display, and print text. Additionally it includes several language translations (user interface and documentation) and some additional software such as BCP support, fonts, and utilities on the Language CD.

The English Unicode locale (`en_US.UTF-8`) is installed as the default, while other locales are installed when the locale is selected as `install locale` during the Solaris install process. Since the `UTF-8` locales require all the languages fonts, basic fonts supporting all languages are also installed as the default.

The file system safe universal transformation format, or `UTF-8`, is an encoding defined by X/Open as a multibyte representation of Unicode. `UTF-8` encompasses almost all of the characters for traditional single-byte and multibyte locales for European and Asian languages for Solaris locales.

Additional locale support is packaged according to the geographic region that the locales support. During the Solaris installation process, you are prompted to choose which geographic regions require your support. The locale support available after installation has finished depends on the choices made at this stage.

Supported Locales

The following tables list all the locales supported by the Solaris 9 environment. The locale names have been updated in keeping with international naming standards.

TABLE 3–3 Asia

Locale	User Interface	Territory	Codeset	Language Support
`hi_IN.UTF-8`	English	India	UTF-8[1]	Hindi (UTF-8) Unicode 3.1
`ja`	Japanese	Japan	eucJP[2]	Japanese (EUC)
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[3]	Japanese (PC kanji)
				JIS X 0201-1976
				JIS X 0208-1990
`ja_JP.UTF-8`	Japanese	Japan	UTF-8	Japanese (UTF-8) Unicode 3.1
`ko_KR.EUC`	Korean	Korea	5601 [4]	Korean (EUC) KSC 5601-1987
`ko_KR.UTF-8`	Korean	Korea	UTF-8	Korean (UTF-8) Unicode 3.1
`th_TH.UTF-8`	English	Thailand	UTF-8	Thai (UTF-8) Unicode 3.1
`zh_CN.EUC`	Simplified Chinese	PRC	gb2312[5]	Simplified Chinese (EUC) GB2312-1980
`zh_CN.GBK`	Simplified Chinese	PRC	GBK [6]	Simplified Chinese (GBK) 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) Unicode 3.1

TABLE 3–3 Asia *(Continued)*

Locale	User Interface	Territory	Codeset	Language Support
`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) Unicode 3.1
`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) Unicode 3.1

1. UTF-8 is the UTF-8 defined in ISO/IEC 10646–1:2000 and also Unicode 3.1.
2. eucJP signifies the Japanese EUC codeset. 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.
3. PCK is also known as Shift JIS (SJIS).
4. 5601 signifies the Korean EUC codeset containing KS C 5636 and KS C 5601–1987.
5. gb2312 signifies Simplified Chinese EUC codeset, which contains GB 1988–80 and GB 2312–80.
6. GBK signifies GB extensions. This includes all GB 2312–80 characters and all Unified Han characters of ISO/IEC 10646–1, as well as Japanese Hiragana and Katakana characters. It also includes many characters of Chinese, Japanese, and Korean character sets and of ISO/IEC 10646–1

TABLE 3–4 Australasia

Locale	User Interface	Territory	Codeset	Language Support
`en_AU.ISO8859-1`	English	Australia	ISO8859-1	English (Australia)
`en_NZ.ISO8859-1`	English	New Zealand	ISO8859-1	English (New Zealand)

TABLE 3–5 Central America

Locale	User Interface	Territory	Codeset	Language Support
`es_CR.ISO8859-1`	Spanish	Costa Rica	ISO8859-1	Spanish (Costa Rica)
`es_GT.ISO8859-1`	Spanish	Guatemala	ISO8859-1	Spanish (Guatemala)
`es_NI.ISO8859-1`	Spanish	Nicaragua	ISO8859-1	Spanish (Nicaragua)
`es_PA.ISO8859-1`	Spanish	Panama	ISO8859-1	Spanish (Panama)
`es_SV.ISO8859-1`	Spanish	El Salvador	ISO8859-1	Spanish (El Salvador)

TABLE 3–6 Central Europe

Locale	User Interface	Territory	Codeset	Language Support
cs_CZ.ISO8859-2	English	Czech Republic	ISO8859-2	Czech (Czech Republic)
de_AT.ISO8859-1	German	Austria	ISO8859-1	German (Austria)
de_AT.ISO8859-15	German	Austria	ISO8859-15	German (Austria, ISO8859-15 - Euro)
de_CH.ISO8859-1	German	Switzerland	ISO8859-1	German (Switzerland)
de_DE.UTF-8	German	Germany	UTF-8	German (Germany, Unicode 3.1)
de_DE.ISO8859-1	German	Germany	ISO8859-1	German (Germany)
de_DE.ISO8859-15	German	Germany	ISO8859-15	German (Germany, ISO8859-15 - Euro)
fr_CH.ISO8859-1	French	Switzerland	ISO8859-1	French (Switzerland)
hu_HU.ISO8859-2	English	Hungary	ISO8859-2	Hungarian (Hungary)
pl_PL.ISO8859-2	English	Poland	ISO8859-2	Polish (Poland)
pl_PL.UTF-8	English	Poland	UTF-8	Polish (Poland, Unicode 3.1
sk_SK.ISO8859-2	English	Slovakia	ISO8859-2	Slovak (Slovakia)

TABLE 3–7 Eastern Europe

Locale	User Interface	Territory	Codeset	Language Support
bg_BG.ISO8859-5	English	Bulgaria	ISO8859-5	Bulgarian (Bulgaria)
et_EE.ISO8859-15	English	Estonia	ISO8859-15	Estonian (Estonia)
hr_HR.ISO8859-2	English	Croatia	ISO8859-2	Croatian (Croatia)
lt_LT.ISO8859-13	English	Lithuania	ISO8859-13	Lithuanian (Lithuania)
lv_LV.ISO8859-13	English	Latvia	ISO8859-13	Latvian (Latvia)
mk_MK.ISO8859-5	English	Macedonia	ISO8859-5	Macedonian (Macedonia)
ro_RO.ISO8859-2	English	Romania	ISO8859-2	Romanian (Romania)
ru_RU.KOI8-R	English	Russia	KOI8-R	Russian (Russia, KOI8-R)

TABLE 3–7 Eastern Europe *(Continued)*

Locale	User Interface	Territory	Codeset	Language Support
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, Unicode 3.1)
sh_BA.ISO8859-2@bosnia	English	Bosnia	ISO8859-2	Bosnian (Bosnia)
sl_SI.ISO8859-2	English	Slovenia	ISO8859-2	Slovenian (Slovenia)
sq_AL.ISO8859-2	English	Albania	ISO8859-2	Albanian (Albania)
sr_YU.ISO8859-5	English	Serbia	ISO8859-5	Serbian (Serbia)
tr_TR.ISO8859-9	English	Turkey	ISO8859-9	Turkish (Turkey)
tr_TR.UTF-8	English	Turkey	UTF-8	Turkish (Turkey, Unicode 3.1)

TABLE 3–8 Middle East

Locale	User Interface	Territory	Codeset	Language Support
He	English	Israel	ISO8859-8	Hebrew (Israel)

TABLE 3–9 North Africa

Locale	User Interface	Territory	Codeset	Language Support
ar_EG.UTF-8	English	Egypt	UTF-8	Arabic (Egypt)
Ar	English	Egypt	ISO8859-6	Arabic (Egypt)

TABLE 3–10 North America

Locale	User Interface	Territory	Codeset	Language Support
en_CA.ISO8859-1	English	Canada	ISO8859-1	English (Canada)
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., Unicode 3.1)

TABLE 3–10 North America *(Continued)*

Locale	User Interface	Territory	Codeset	Language Support
`fr_CA.ISO8859-1`	French	Canada	ISO8859-1	French (Canada)
`es_MX.ISO8859-1`	Spanish	Mexico	ISO8859–1	Spanish (Mexico)

TABLE 3–11 North Europe

Locale	User Interface	Territory	Codeset	Language Support
`da_DK.ISO8859-1`	English	Denmark	ISO8859–1	Danish (Denmark)
`da_DK.ISO8859-15`	English	Denmark	ISO8859–15	Danish (Denmark, ISO8859–15 Euro)
`fi_FI.ISO8859-1`	English	Finland	ISO8859–1	Finnish, Unicode 3.1)
`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)
`no_NO.ISO8859-1`@bokmal	English	Norway	ISO8859–1	Norwegian (Norway -Bokmal)
`no_NO.ISO8859-1`@nyorsk	English	Norway	ISO8859–1	Norwegian (Norway -Nynorsk)
`sv_SE.ISO8859-1`	Swedish	Sweden	ISO8859–1	Swedish (Sweden)
`sv_SE.ISO8859-15`	Swedish	Sweden	ISO8859–15	Swedish (Sweden, ISO8859–15 Euro)
`sv_SE.UTF-8`	Swedish	Sweden	UTF-8	Swedish (Sweden, Unicode 3.1)

TABLE 3–12 South America

Locale	User Interface	Territory	Codeset	Language Support
`es_AR.ISO8859-1`	Spanish	Argentina	ISO8859-1	Spanish (Argentina)
`es_BO.ISO8859-1`	Spanish	Bolivia	ISO8859-1	Spanish (Bolivia)
`es_CL.ISO8859-1`	Spanish	Chile	ISO8859-1	Spanish (Chile)
`es_CO.ISO8859-1`	Spanish	Colombia	ISO8859-1	Spanish (Colombia)
`es_EC.ISO8859-1`	Spanish	Ecuador	ISO8859-1	Spanish (Ecuador)
`es_PE.ISO8859-1`	Spanish	Peru	ISO8859-1	Spanish (Peru)
`es_PY.ISO8859-1`	Spanish	Paraguay	ISO8859-1	Spanish (Paraguay)
`es_UY.ISO8859-1`	Spanish	Uruguay	ISO8859-1	Spanish (Uruguay)

TABLE 3–12 South America *(Continued)*

Locale	User Interface	Territory	Codeset	Language Support
`es_VE.ISO8859-1`	Spanish	Venezuela	ISO8859-1	Spanish (Venezuela)
`pt_BR.ISO8859-1`	English	Brazil	ISO8859-1	Portuguese (Brazil)
`pt_BR.UTF-8`	English	Brazil	UTF-8	Portuguese (Brazil, Unicode 3.1)

TABLE 3–13 South Europe

Locale	User Interface	Territory	Codeset	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)
`el_GR.ISO8859-7`	English	Greece	ISO8859-7	Greek (Greece)
`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, Unicode 3.1)
`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, Unicode 3.1)
`pt_PT.ISO8859-1`	English	Portugal	ISO8859-1	Portuguese (Portugal)
`pt_PT.ISO8859-15`	English	Portugal	ISO8859-15	Portuguese (Portugal, ISO8859-15 - Euro)

TABLE 3–14 Western Europe

Locale	User Interface	Territory	Codeset	Language Support
`en_GB.ISO8859-1`	English	Great Britain	ISO8859-1	English (Great Britain)
`en_IE.ISO8859-1`	English	Ireland	ISO8859-1	English (Ireland)
`fr_BE.ISO8859-1`	French	Belgium-Walloon	ISO8859-1	French (Belgium-Walloon, Unicode 3.1)
`fr_BE.UTF-8`	French	Belgium-Walloon	UTF-8	French (Belgium-Walloon, Unicode 3.1)
`fr_FR.ISO8859-1`	French	France	ISO8859-1	French (France)

TABLE 3–14 Western Europe *(Continued)*

Locale	User Interface	Territory	Codeset	Language Support
`fr_FR.UTF-8`	French	France	UTF-8	French (France, Unicode 3.1)
`nl_BE.ISO8859-1`	English	Belgium-Flemish	ISO8859-1	Dutch (Belgium-Flemish)
`nl_NL.ISO8859-1`	English	Netherlands	ISO8859-1	Dutch (Netherlands)

Multiple Key Compose Sequences for Locales

Many of the Solaris locales, especially the European and Unicode locales, allow input of various characters by using so-called "dead key sequences" which are also known as compose key sequences.

The compose key sequence based input is 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 sequences. For more complete information about the compose key sequences, see "English/European Input Mode" on page 105

TABLE 3–15 Diacritical Characters Created with Compose Key

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

Note – If the current locale's codeset does not have a corresponding character, a compose sequence cannot be used. For example, since there is no `Z` with a `caron` in ISO8859–1, it is not possible to input a `Z` with a `caron` in the `en_US.ISO8859-1` locale.

Keyboard Support in the Solaris 9 Product

Solaris recognizes and supports various keyboards with different key layouts made for specific regions, and layout support for both Sun SPARC and Intel Architecture (IA) platforms. Solaris 9 supports the following regional keyboards:

TABLE 3–16 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
	Czechoslovakia	X		X
	Denmark	X	X	X
	Finland		X	
	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

TABLE 3–16 Support for Regional Keyboards *(Continued)*

Region	Country	Sun Keyboard Type 4/5/5c	Sun Keyboard Type 6	PC Keyboard
	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	

For regions with keyboard layouts conforming to the International Standard, such as China, use the keyboard layout support provided for the U.S.A. to input the locale's characters, since 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

Users can change keyboard layouts in the Solaris product by using the DIP switch settings under most of 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 – Users cannot change layouts of Type 6 keyboards since there is no such DIP switches at the back of the keyboards. Some Type 5 and 5c keyboards , for instance, US, US/Unix, and Japanese keyboards have jumpers instead of DIP switches. There are no utilities or tools for both SPARC and IA platforms (apart from a standard UNIX tool, like `xmodmap`(1)) bundled into the Solaris 9 operating environment for switching 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 3–17 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 (Netherlands4.kt)	000111
8	Norway (Norway4.kt)	001000
9	Portugal (Portugsl4.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

TABLE 3–17 Layouts for Type 4, 5, and 5c Keyboards *(Continued)*

DIP Switch	Keyboard (Keytable file)	Setting in Binary
16 (0x10)	Korea (Korea4.kt)	010000
17 (0x11)	Traditional Chinese	010001
33 (0x21)	U.S.A. (US5.kt)	100001
34 (0x22)	U.S.A./Unix (US-UNIX5.kt)	100010
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 (Netherlands5.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)	Traditional Chinese (5.kt)	110000
49 (0x31)	Japan (Japan5.kt)	110001
50 (0x32)	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)	Turkey-Q5 (TurkeyQ5.kt)	111000
57 (0x39)	Greece (Greece5.kt)	111001

TABLE 3–17 Layouts for Type 4, 5, and 5c Keyboards *(Continued)*

DIP Switch	Keyboard (Keytable file)	Setting in Binary
58 (0x3a)	Arabic (Arabic5.kt)	111011
59 (0x3b)	Lithuania (Lithuania5.kt)	111010
60 (0x3c)	Belgium (Belgian5.kt)	111100
62 (0x3e)	Turkey-F5 (TurkeyF5.kt)	111110
63 (0x3f)	Canada/French (Canada_Fr5_TBITS5.kt)	111111

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

Changing the layout from one keyboard layout to another layout (Czech for example), requires the following steps:

1. Find out 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 in the decimal number.

   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.

   The correct binary value for the Czech keyboard is 110101.
3. Be a super user. 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) above.

   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` Settings.
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 110101.

Changing Between Keyboards on Intel Systems

On Intel architecture systems, a keyboard is selected during the `kdmconf(1M)` part of the installation. To change this at any time after installation, first exit your GUI desktop environment to the command line mode. As super-user, type `kmdconfig` to run the program. Then follow instructions to get the keyboard layout you choose.

The following figure is for the Turkish F keyboard.

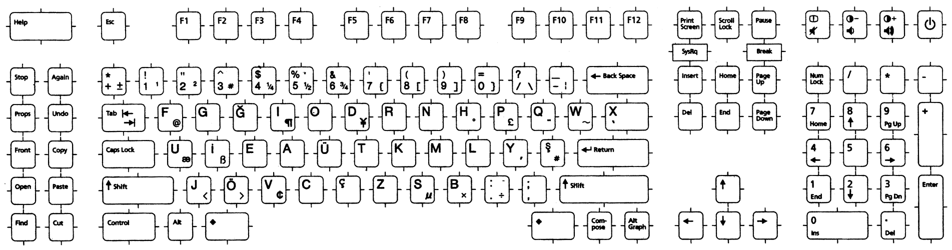


FIGURE 3–1 Turkish F Keyboard

The following figure is for the Turkish Q keyboard.

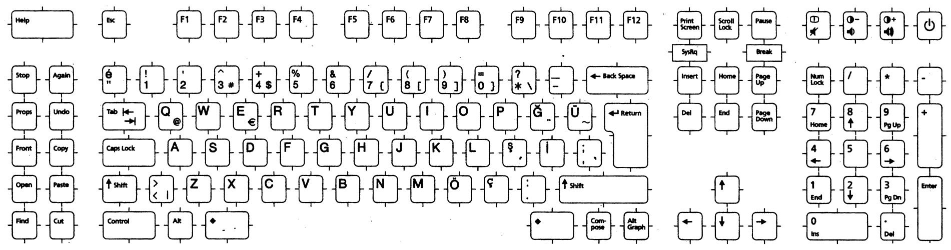


FIGURE 3–2 Turkish Q Keyboard

The following figure is for the Belgian keyboard.

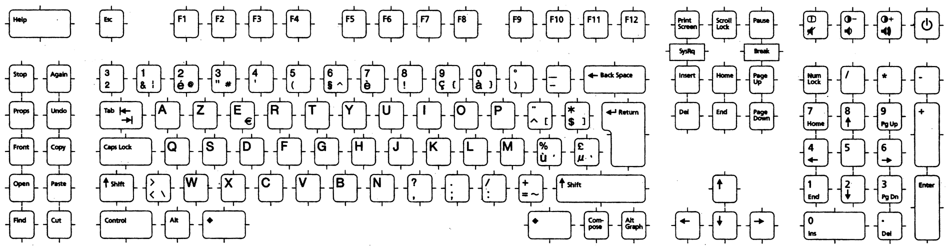


FIGURE 3–3 Belgian Keyboard

The following figure is for the Danish keyboard.

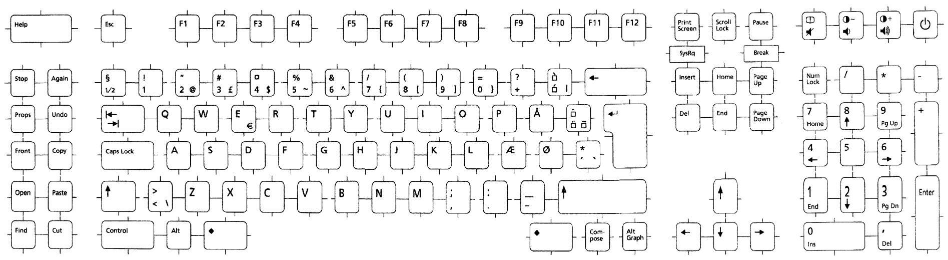


FIGURE 3–4 Danish Keyboard

The following figure is for the Finnish keyboard.

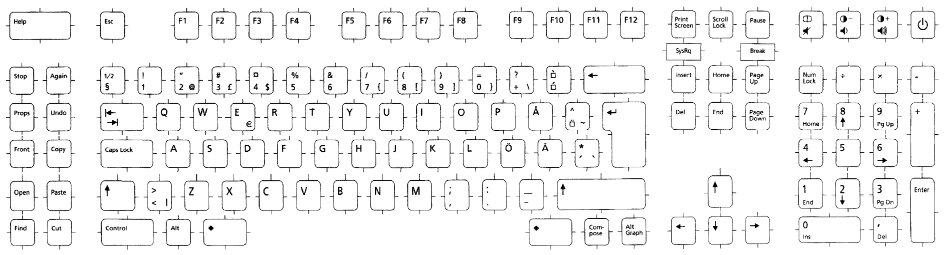


FIGURE 3–5 Finnish Keyboard

The following figure is for the French keyboard.

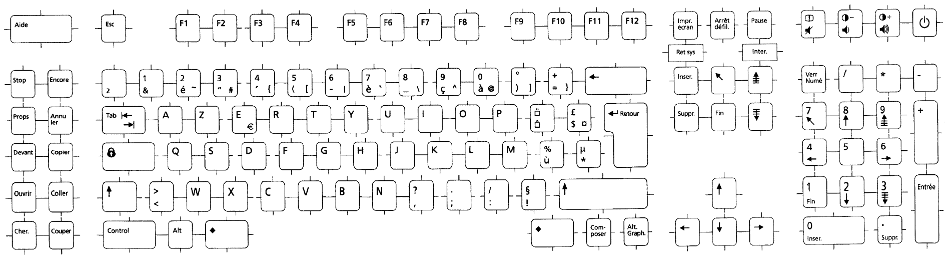


FIGURE 3–6 French Keyboard

The following figure is for the German keyboard.

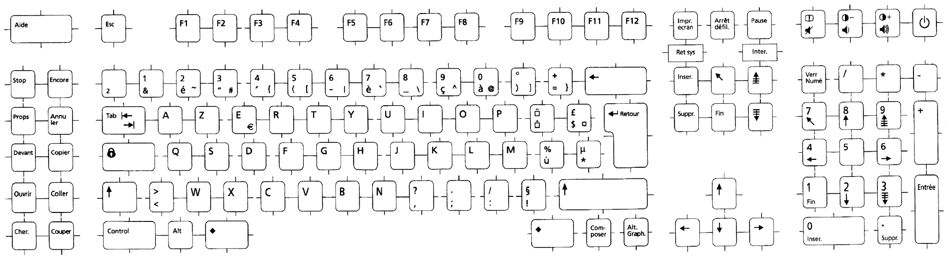


FIGURE 3–7 German Keyboard

The following figure is for the Italian keyboard.

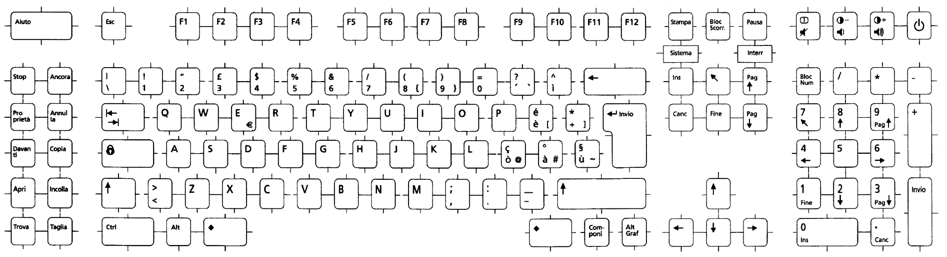


FIGURE 3–8 Italian Keyboard

The following is for the Netherlands (Dutch) keyboard,

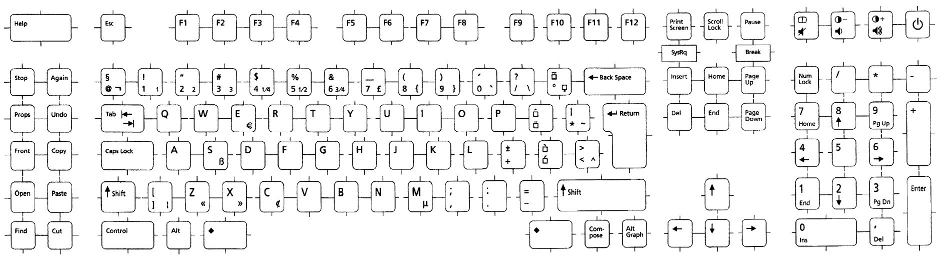


FIGURE 3–9 Netherlands (Dutch) Keyboard

The following figure is for the Norwegian keyboard.

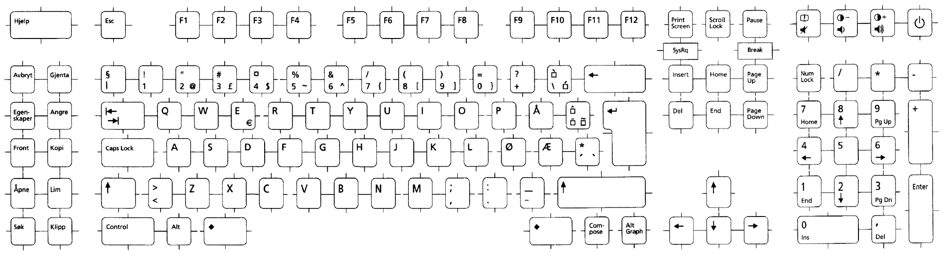


FIGURE 3–10 Norwegian Keyboard

The following figure is for the Portuguese keyboard.

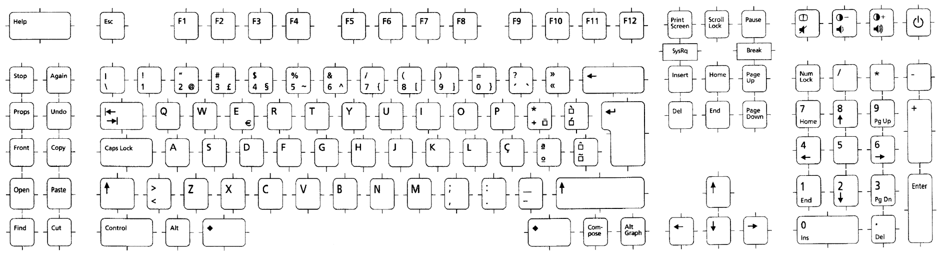


FIGURE 3–11 Portuguese Keyboard

The following figure is for the Spanish keyboard.

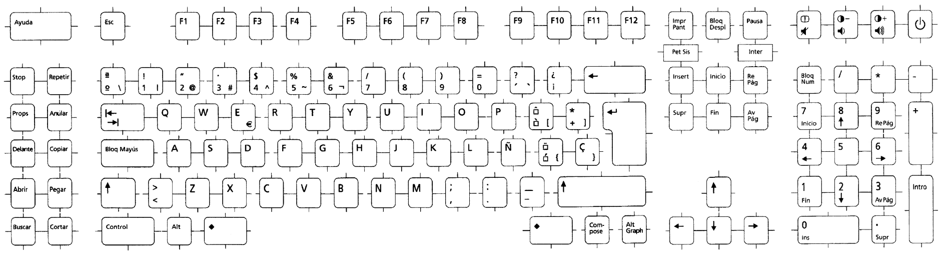


FIGURE 3–12 Spanish Keyboard

The following figure is for the Swedish keyboard.

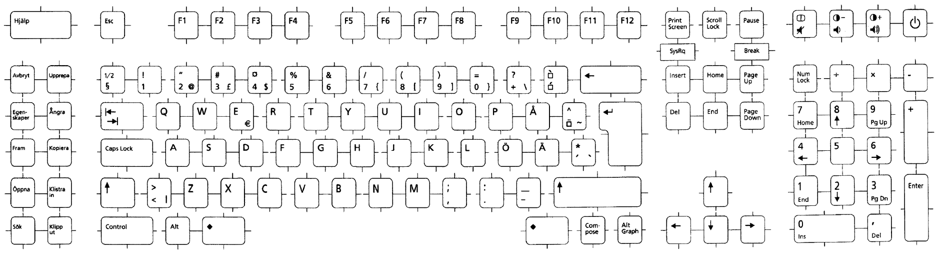


FIGURE 3–13 Swedish Keyboard

The following is for the Swiss (French) keyboard.

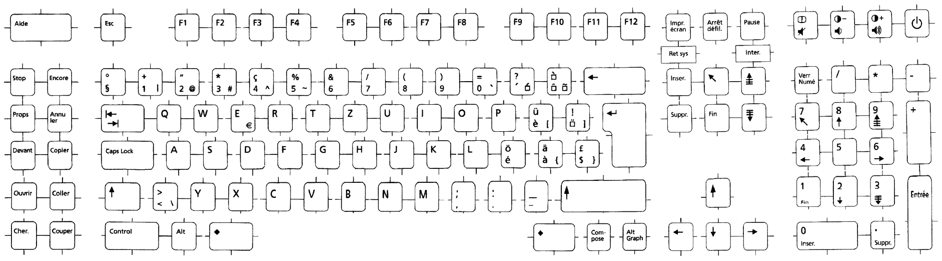


FIGURE 3–14 Swiss (French) Keyboard

The following figure is for the Swiss (German) keyboard.

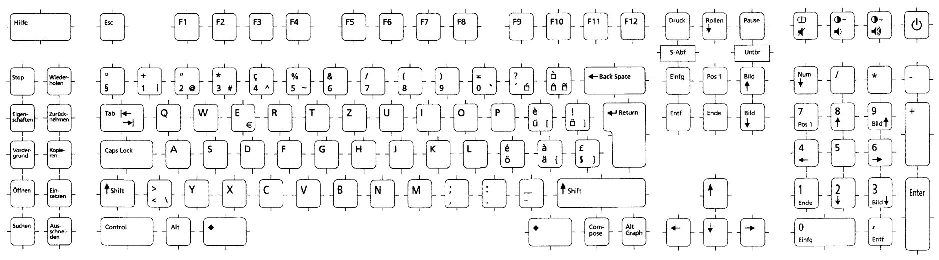


FIGURE 3–15 Swiss (German) Keyboard

The following figure is for the Traditional Chinese keyboard.

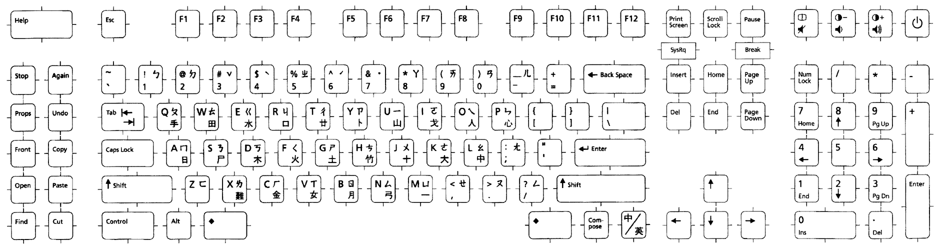


FIGURE 3–16 Traditional Chinese Keyboard

The following figure is for the United Kingdom keyboard.

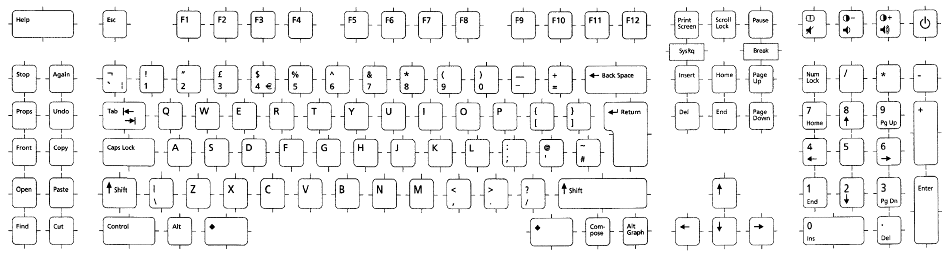


FIGURE 3–17 United Kingdom Keyboard

The following figure is for the US/UNIX keyboard.

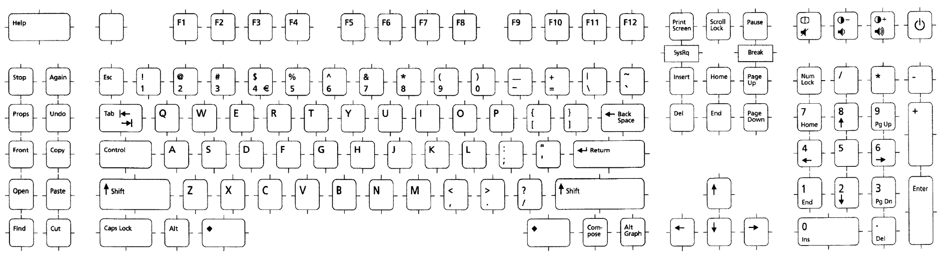


FIGURE 3–18 US/UNIX Keyboard

The following figure is for the United States keyboard.

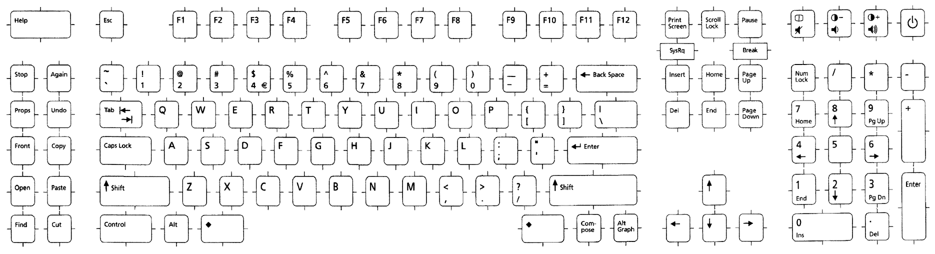


FIGURE 3–19 United States Keyboard

Font Support

For a complete and updated table, refer to /usr/lib/iconv and to the `iconv`(5) man page.

CHAPTER 4

Supported Asian Locales

The following sections show these Asian supported locales

- "Simplified Chinese Localization" on page 84
- "Traditional Chinese Localization" on page 88
- "Japanese Localization" on page 90
- "Korean Localization" on page 94

Asian Supported Locales

The following table shows the a summary of Asian supported locales.

TABLE 4–1 Summary of Asian Locales

Language	Locale Name	Description	Supported Character Set
Korean	`ko`	Korean (EUC)	KS X
	`ko.UTF-8`	Korean (UTF-8)	KSC 5700–2000
Simplified Chinese	`zh_CN.EUC`	Simplified Chinese (EUC)	GB 2312-1980
	`zh_CN.GBK`	Simplified Chinese (GBK)	GBK
	`zh_CN.GB18030`	Simplified Chinese (GB18030–2000)	GB18030–2000
	`zh_CN.UTF-8`	Simplified Chinese (UTF-8)	Unicode 3.1

TABLE 4–1 Summary of Asian Locales *(Continued)*

Language	Locale Name	Description	Supported Character Set
Traditional Chinese	`zh_TW.EUC`	Traditional Chinese (EUC)	CNS 11643 –1992
	`zh_TW.UTF-8`	Traditional Chinese (UTF-8)	Unicode 3.1
	`zh_TW.BIG5`	Traditional Chinese (BIG5)	BIG5
	`zh_HK.BIG5HK`	Traditional Chinese (BIG5+HKSCS)	BIG5+HKSCS
	`zh_HK.UTF–8`	Traditional Chinese (UTF-8)	Unicode 3.1
Japanese	`ja`	Japanese (EUC)	JIS X 0201-1976
	`ja_JP.PCK`	Japanese (PCK)[1]	JIS X 0208-1990
	`ja_JP.UTF-8`	Japanese (UTF-8)	JIS X 0212-1990
			VDC [2]
			UDC [3]
Thai	`th_TH.TIS620`	Thai (TIS620.2533)	TIS620.2533
	`th_TH.UTF-8`	Thai (UTF-8)	Unicode 3.1
	`th_TH.ISO8859-11`	`Thai(ISO8859-11)`	`ISO8859-11`
Hindi	`hi_IN.UTF-8`	Hindi(UTF-8)	Unicode 3.1

1. ja_JP.PCK (does not support JIS X 0212–1990).
2. VDC: Vendor defined character. VDCs occupy unused (reserved) code points of JIS X 0208–1990 or JIS X 0212–1990.
3. UDC: User defined character. UDCs occupy unused (reserved) code points of JIS X 0208–1990 or JIS X 0212–1990 (also unused for VDCs).

Input Method Auxiliary Window Support for Simplified and Traditional Chinese

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

- Input methods switching
- Input methods properties configuration
- Chinese full_width/half_width character mode switching

- Chinese punctuation/English punctuation mode switching.
- Virtual keyboard.
- Input method utilities - provides a way to create a remove input methods based on the codetable.
- Input method management panel, with input method properties setting, input method selection by user
- Lookup tables for GB2312/GBK/GB18030–2000/HKSCS/CNS/BIG5/Unicode characters
- Code table management tool

For more detailed information, please see the *Simplified Chinese User's Guide* and the *Traditional Chinese User's Guide*.

The input method auxiliary windows supports all UTF-8 and the following Chinese locales:

- `zh/zh_CN.EUC`
- `zh.GBK/zh_CN.GBK`
- `zh.UTF-8/zh_CN.UTF-8`
- `zh_TW/zh_TW.EUC`
- `zh_TW.BIG5`
- `zh_TW.UTF-8`
- `zh_HK.BIG5HK`
- `zh_HK.UTF-8`
- `zh_CN.GB18030`

Two kinds of input methods are supported:

- Methods based on a codetable such as CangJie
- Methods developed by a vendor (such as NewPinYin, or NeiMa)

The interface model for auxiliary window support is shown in the following figure.

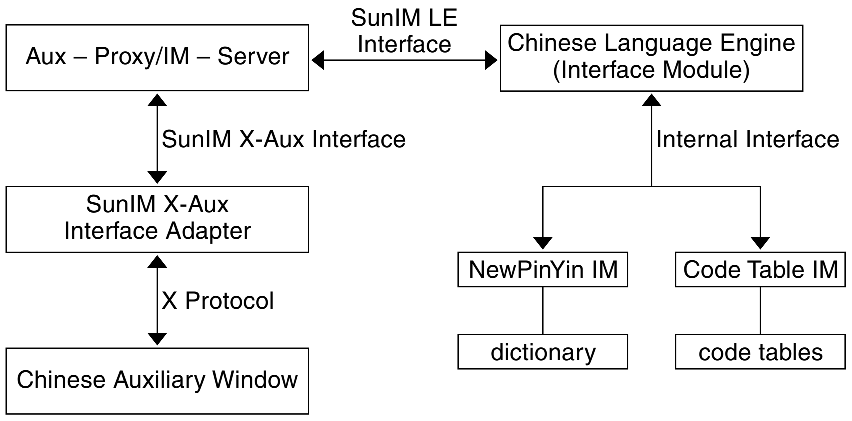


FIGURE 4–1 Interface Model for Auxiliary Window Support

Simplified Chinese Localization

Simplified Chinese in the Solaris 9 environment provides four locales: `zh`, `zh.GBK`, `zh_CN.GB18030`, and `zh.UTF-8`, and . In the `zh` locale, the EUC scheme is used to encode GB2312–80. The `zh.GBK` locale supports the GBK codeset, which is a superset of GB2312–80.

The new GB18030–2000 codeset is now supported in the `zh_CN.GB18030` locale.

Simplified Chinese is used mostly in the People's Republic of China (PRC) and in Singapore.

The following input methods are supported for the `zh` locale:

- New QuanPin
- New ShuangPin
- QuanPin
- ShuangPin
- GB2312 NeiMa
- English-Chinese
- Optional Codetable Input Methods
- Input Method Auxiliary Window Support for Simplified Chinese

The following are the Chinese input methods available in the Chinese locales:

- New PinYin input method for all Simplified Chinese locales
- New ShuangPin input method for all Simplified Chinese locales

- New Chuyin input method for Traditional Chinese locales
- Cantonese input methods for all Chinese locales
- English-Chinese input methods for all Chinese locales
- Codepoint input methods:
  - GB2312
  - GBK
  - GB18030–2000
  - HKSCS
  - CNS
  - BIG5
  - Unicode
- Other Codetable-based Traditional Chinese input methods:
  - Cangjie
  - Chuyin
  - Dayi

The following input methods are supported for the `zh_CN.GB18030` locale:

- New QuanPin
- New ShuangPin
- QuanPin
- ShuangPin
- GB18030–2000 NeiMa
- English-Chinese
- Optional Codetable Input Methods
- Input Method Auxiliary Window Support for Simplified Chinese

The following input methods are supported for both the `zh.GBK` and the `zh.UTF-8` locales:

- New QuanPin
- New ShuangPin
- QuanPin
- ShuangPin
- GBK NeiMa
- English-Chinese
- Optional Codetable Input Methods
- Input Method Auxiliary Window Support for Simplified Chinese

The auxiliary window for Chinese input methods provides a friendly and extensible input method user interface for all Chinese locales. The following list contains some of the functions supported by the auxiliary window:

- Input methods switching.

- Input methods properties configuration.
- Lookup tables for GB2312/GBK/GB18030–2000/HKSCS/CNS/BIG5/Unicode character sets.
- Code table management tool.
- Visual keyboard.

For more detailed information about Auxiliary Windows for Chinese input methods, please see *Simplified Chinese User's Guide* and *Chinese User's Guide*.

The following table shows the TrueType fonts for the `zh` locale.

TABLE 4–2 TrueType Fonts for the `zh_CN.EUC` Locale

Full Family Name	Subfamily	Format	Vendor	Encoding
Fangsong	R	TrueType	Hanyi	GB2312.1980
Hei	R	TrueType	Monotype	GB2312.1980
Kai	R	TrueType	Monotype	GB2312.1980
Song	R	TrueType	Monotype	GB2312.1980

The following table shows the bitmap fonts for the `zh` locale.

TABLE 4–3 Bitmap Fonts for the `zh_CN.EUC` Locale

Full Family Name	Subfamily	Format	Encoding
Song	B	PCF (14,16)	GB2312.1980
Song	R	PCF (12,14,16,20,24)	GB2312.1980

The following table shows the TrueType fonts for the `zh_CN.GBK` Locale.

TABLE 4–4 TrueType Fonts for the `zh_CN.GBK` Locale

Full Family Name	Subfamily	Format	Vendor	Encoding
Fansong	R	TrueType	Zhongyi	GBK
Hei	R	TrueType	Zhongyi	GBK
Kai	R	TrueType	Zhongyi	GBK
Song	R	TrueType	Zhongyi	GBK

The following table shows the bitmap fonts for the `zh_CN.GBK` locale.

TABLE 4–5 Bitmap Fonts for the `zh_CN.GBK` Locale

Full Family Name	Subfamily	Format	Encoding
Song	R	PCF (12,14,16,20,24)	GBK

The following table shows the TrueType fonts for the `zh_CN.GB18030` Locale

TABLE 4–6 TrueType Fonts for the `zh_CN.GB18030` Locale

Family Name	Weight	Format	Vendor	Encoding
FangSong	R	TrueType	FangZheng	GB18030–2000
Song	R	TrueType	FangZheng	GB18030–2000
Hei	R	TrueType	FangZheng	GB18030–2000
Kai	R	TrueType	FangZheng	GB18030–2000

The following table shows Bitmap Fonts for the `zh_CN.GB18030` Locale

TABLE 4–7 Bitmap Fonts for the `zh_CN.GB18030` Locale

Family Name	Weight	Format	Encoding
Song	R	PCF(12,14,16,20,24)	GB18030–2000

The following table shows the supported codeset conversions for Simplified Chinese.

TABLE 4–8 Codeset Conversions for Simplified Chinese

Code	Symbol	Target Code	Symbol
`GB2312-80`	`zh_CN.euc`	ISO 2022-7	`zh_CN.iso2022-7`
`ISO 2022-7`	`zh_CN.iso2022-7`	GB2312-80	`zh_CN.euc`
`GB2312-80`	`zh_CN.euc`	ISO 2022-CN	`zh_CN.iso2022-CN`
`HZ-GB-2312`	`HZ-GB-2312`	GB2312–80	`zh_CN.euc`
`HZ-GB-2312`	`HZ-GB-2312`	GBK	`zh_CN.gbk`
`HZ-GB-2312`	`HZ-GB-2312`	UTF-8	`UTF-8`
`ISO-2022-CN`	`zh_CN.iso2022-CN`	GB2312-80	`zh_CN.euc`
`UTF-8`	`UTF-8`	GB2312-80	`zh_CN.euc`
`GB2312-80`	`zh_CN.euc`	UTF-8	`UTF-8`

TABLE 4–8 Codeset Conversions for Simplified Chinese *(Continued)*

Code	Symbol	Target Code	Symbol
`zh.GBK`	`zh_CN.gbk`	ISO2022-CN	`zh_CN.iso2022-CN`
`ISO2022-CN`	`zh_CN.iso2022-CN`	zh.GBK	`zh_CN.gbk`
`UTF-8`	`UTF-8`	zh.GBK	`zh_CN.gbk`
`zh.GBK`	`zh_CN.gbk`	UTF-8	`UTF-8`
`UTF-8`	`UTF-8`	ISO2022-CN	`zh_CN.iso2022-CN`
`ISO2022-CN`	`zh_CN.iso2022-CN`	UTF-8	`UTF-8`
`zh_CN.GB18030`	`zh_CN.gb18030`	UTF-8	`UTF-8`
`UTF-8`	`UTF-8`	zh_CN.GB18030	`zh_CN.gb18030`

Traditional Chinese Localization

Traditional Chinese in the Solaris 9 product provides five locales:

- `zh_TW.EUC` where the EUC scheme is used to encode the CNS11643.1992 codeset.
- `zh_TW.BIG5` where the locale supports BIG5.
- `zh_TW.UTF-8` where the locale supports the latest Unicode standard.
- `zh_HK.BIG5HK` where the locale supports BIG5 plus HKSCS.
- `zh_HK.UTF-8` where the locale supports the latest Unicode standard.

Traditional Chinese is used mostly in Taiwan and Hong Kong, China. The following input methods are supported in the zh_TW.EUC, zh_TW.BIG5, and zh_TW.UTF-8 locales:

- New ChuYin
- ChuYin
- TsangChieh
- Array
- BoShiaMy
- DaYi
- JianYi
- Cantonese
- EUC NeiMa
- BIG5 NeiMa
- English-Chinese
- Optional Codetable Input Methods (such as PinYin)
- Input Method Auxiliary Window Support for Traditional Chinese

The following input methods are supported in the `zh_HK.BIG5HK` and `zh_HK.UTF-8` locales.

- ChuYin
- TsangChieh
- Array
- BoShiaMy
- DaYi
- JianYi
- Cantonese
- BIG5+HKSCS NeiMa
- English-Chinese
- Optional Codetable Input Methods (such as PinYin)
- Input Method Auxiliary Window Support for Traditional Chinese
- New ChuYin

The following table shows Traditional Chinese TrueType Fonts for the `zh_TW` locales.

TABLE 4–9 Traditional Chinese TrueType Fonts for the `zh_TW` Locales

Full Family Name	Subfamily	Format	Vendor	Encoding
Hei	R	TrueType	Hanyi	CNS11643.1992
Kai	R	TrueType	Hanyi	CNS11643.1992
Ming	R	TrueType	Hanyi	CNS11643.1992

The following table shows the Traditional Chinese bitmap fonts for the `zh_TW` Locales.

TABLE 4–10 Traditional Chinese Bitmap Fonts for the `zh_TW` Locales

Full Family Name	Subfamily	Format	Encoding
Ming	R	PCF (12,14,16,20,24)	CNS11643.1992

The following table shows the TrueType fonts for the `zh_HK.BIG5HK` locale:

TABLE 4–11 TrueType fonts for the `zh_HK.BIG5HK` locale

Family Name	Weight	Format	Vendor	Encoding
Ming	R	TrueType	FangZheng	Big5–HKSCS
Hei	R	TrueType	FangZheng	Big5–HKSCS
Kai	R	TrueType	FangZheng	Big5–HKSCS

The following table shows the Bitmap Fonts for the `zh_HK.BIG5HK` Locale:

TABLE 4–12 Bitmap Fonts for the `zh_HK.BIG5HK` Locale

Family Name	Weight	Format	Encoding
Ming	R	PCF(12,14,16,20,24)	Big5–HKSCS

The following table shows the supported codeset conversions for Traditional Chinese.

TABLE 4–13 Codeset Conversions for Traditional Chinese

Code	Symbol	Target Code	Symbol
`BIG5`	`zh_TW-big5`	CNS 11643	`zh_TW-euc`
`BIG5`	`zh_TW-big5`	ISO-2022–CN	`zh_TW-iso2022-CN-EXT`
`BIG5`	`zh_TW-big5`	UTF-8	`UTF-8`
`CNS 11643`	`zh_TW-euc`	BIG5	`zh_TW-big5`
`CNS 11643`	`zh_TW-euc`	UTF-8	`UTF-8`
`CNS 11643`	`zh_TW-euc`	ISO 2022-7	`zh_TW-iso2022-7`
`ISO 2022-7`	`zh_TW-iso2022-7`	CNS 11643	`zh_TW-euc`
`CNS 11643`	`zh_TW-euc`	ISO 2022-CN-EXT	`zh_TW-iso2022-CN-EXT`
`ISO 2022-CN`	`zh_TW-iso2022-CN-EXT`	BIG5	`zh_TW-big5`
`ISO 2022-CN-EXT`	`zh_TW-iso2022-CN-EXT`	CNS 11643	`zh_TW-euc`
`UTF-8`	`UTF-8`	BIG5	`zh_TW-big5`
`UTF-8`	`UTF-8`	CNS 11643	`zh_TW-euc`
`CNS 11643`	`zh_TW-euc`	UTF-8	`UTF-8`
`UTF-8`	`UTF-8`	ISO 2022-7	`zh_TW-iso2022-7`
`ISO 2022-7`	`zh_TW-iso2022-7`	UTF-8	`UTF-8`
`BIG5+HKSCS`	`zh_HK-hkscs`	UTF-8	`UTF-8`
`UTF-8`	`UTF-8`	BIG5+HKSCS	`zh_HK-hkscs`

Japanese Localization

This section describes Japanese locale-specific information.

Japanese Locales

Three Japanese locales, which support different character encoding, are available in the Solaris 9 environment. The `ja` (or `ja_JP.eucJP`) locale is based on the Japanese EUC. 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 `eucJP`(5) for a map between Japanese EUC and the character set. See the `PCK(5)` man page for the map between PCK and the character set.

Japanese Character Sets

Supported Japanese character sets are:

- JIS X 0201–1976
- JIS X 0208–1990
- JIS X 0212–1990

JIS X 0212–1990 is not supported in the `ja_JP.PCK` locale.

Vendor-defined character (VDC) and user-defined character (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 that the code points are 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. Type1 font is also used by UDC.

Japanese bitmap fonts are described in the following table.

TABLE 4–14 Japanese Bitmap Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
`gothic`	R, B	PCF(12,14,16,20,24)		JISX0208.1983, JISX0201.1976
`minchou`	R	PCF(12,14,16,20,24)		JISX0208.1983, JISX0201.1976
`hg gothic b`	R	PCF(12,14,16,18,20,24)	RICOH	JISX0208.1983, JISX0201.1976

TABLE 4–14 Japanese Bitmap Fonts *(Continued)*

Full Family Name	Subfamily	Format	Vendor	Encoding
`hg mincho l`	R	PCF(12,14,16,18,20,2)	RICOH	JISX0208.1983, JISX0201.1976
`heiseimin`	R	PCF(12,14,16,18,20,24)	RICOH	JISX0212.1990

Japanese TrueType fonts are described in the following table.

TABLE 4–15 Japanese TrueType Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
`hg gothic b`	R	TrueType	RICOH	JIS X 0208.1983, JIS X 0201.1976
`hg mincho l`	R	TrueType	RICOH	JIS X 0208.1983, JIS X 0201.1976
`heiseimin`	R	TrueType	RICOH	JIS X 0212.1990

Japanese Input Systems

ATOK12 is the default Japanese input system in the Solaris 9 environment. It is available for all Japanese locales and all UTF-8 locales (when the Japanese locale is installed). The Wnn6 Japanese input system is also available for all Japanese lo9cales. You can switch input systems from the Workspace menu. For japanese Solaris 1.x BCP support, the kkcv Japanese input system is available.

An example procedure of Japanese input using ATOK12 is as follows:

1. Turn conversion mode on: (Enter Control + Space)
2. Enter Kana character text: (Type ’kanjihenkan” for example)
3. Convert to Kanji character tet: (Enter Space). If you want to display other Kanji characters, press the Space bar to display the conversion candidate table. Enter the number you want to select and commit.
4. Commit the Kanji character text: (Enter Return for the entire text). (Press the Down Arrow key for the highlight/focused characters.)
5. Turn conversion mode off: (Control + Space).

Terminal Setting for Japanese Terminals

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

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

  ```
  % setenv LANG ja
  % stty defeucw
  ```

- If your terminal is not a CDE Terminal emulator, but the codeset of your terminal is the same as that of the current locale, use this setting, too.
- If your terminal's codeset doesn't match that of the current locale, use `setterm`(1) to enable code conversion. For example, if you are in locale `ja` but your terminal requires PCK (Shift JIS code), specify:

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

  See `setterm`(3CURSES) for details.

Japanese `iconv` Module

Several Japanese codeset conversions are supported with `iconv`(1) and *iconv(3)*. See the *iconv_ja(5)* man page for details.

User-Defined Character Support

User-defined character `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`.

Differences Between Partial and Full Locales

The following components are only available in the Japanese full locale environment with the Languages CD.

- All translations such as `message`, `help`, `man page` and `document`
- Translated message, help and manpages
- `Wnn6` Japanese input system
- Japanese Solaris 1.x BCP support
- ATOK12
- Mincho (min*) and bold (gotb*) typeface bitmap fonts

- JIS X 0212 Type1 fonts for printing
- Japanese-specific dumb printer and 'jpostprint' support
- Legacy Japanese utilities and libraries (for example, `kanji`).

Korean Localization

In December 1995, the Korean government announced a standard Korean codeset, KS C 5700, which is based on ISO 10646-1/Unicode 2.0.

The ISO-10646 character set uses 2 (UCS-2); Universal Character Set (two-byte form) or 4 (UCS-4) bytes to represent each character.

The ISO-10646 character set cannot be used directly on IBM PC-based operating systems. For example, the kernel and many other modules of the Solaris operating environment interpret certain byte values as control instructions, such as a null character (0x00) in any string. The ISO-10646 character set can be encoded with any bit combinations in the first or subsequent bytes. The ISO-10646 characters cannot be freely transmitted through the Solaris system with these limitations.

In order to establish a migration path, the ISO-10646 character set defines the UCS Transformation Format (UTF), which recodes the ISO-10646 characters without using C0 controls (0x00..0x1F), C1 controls (0x80..0x9F), space (0x20), and DEL (0x7F).

The `ko.UTF-8` is a Solaris locale to support KSC-5700, the Korean standard codeset. It supports all characters in the previous KSC 5601 and all 11,172 Korean characters. Korean UTF-8 supports the Korean language-related ISO-10646 characters and fonts. Because ISO-10646 covers all characters in the world, all of the various input methods and fonts are supplied so that you can input and output any character in any language. Before Universal UTF/UCS becomes available, Korean UTF-8 supports the ISO-10646 code subset that is related to Korean characters as well as all other characters in the previous Korean standard codeset, and extended ASCII.

In the `ko` locale, the EUC scheme is used to encode KSC 5601-1987. The `ko.UTF-8` locale supports the KSC 5700-1995/Unicode 2.0 codeset, which is a superset of KSC 5601-1987. These two locales look the same to the end user, but the internal character encoding is different. The Korean Solaris product supports the following input methods:

For the `ko` locale:

- Hangul 2–BeolSik (1 set of consonants and 1 set of vowels)
- Hangul-Hanja conversion
- Special character
- Hexadecimal code

For the `ko.UTF-8` locale:

- Hangul 2–BeolSik (1 set of consonants and 1 set of vowels)
- Hangul-Hanja conversion
- Special character
- Hexadecimal code

The following table shows the Korean bitmap fonts for the `ko` locale.

TABLE 4–16 Solaris 9 Korean Bitmap Fonts for the `ko` Locale

Full Family Name	Subfamily	Format	Encoding
Gothic	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Graphic	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Haeso	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Kodig	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Myeongijo	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Pilki	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987
Round gothic	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1987

The following table shows the Korean bitmap fonts for the `ko.UTF-8` locale

TABLE 4–17 Solaris 9 Korean Bitmap Fonts for the `ko.UTF-8` Locale

Full Family Name	Subfamily	Format	Encoding
Gothic	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)
Graphic	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)
Haeso	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)
Kodig	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)
Myeongijo	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)
Pilki	R/B	PCF (12,14,16,18,20,24)	KSC 5601-1992 (Johap)

The following table shows the Korean TrueType Fonts for the `ko/ko.UTF-8` Locales

TABLE 4–18 Solaris 9 Korean TrueType Fonts for the `ko/ko.UTF-8` Locales

Full Family Name	Subfamily	Format	Vendor	Encoding
Kodig/Gothic	R	TrueType	Hanyang	Unicode

TABLE 4–18 Solaris 9 Korean TrueType Fonts for the `ko/ko.UTF-8` Locales *(Continued)*

Full Family Name	Subfamily	Format	Vendor	Encoding
Myeongjo	R	TrueType	Hanyang	Unicode
Haeso	R	TrueType	Hanyang	Unicode
RoundGothic	R	TrueType	Hanyang	Unicode

The following table shows the Korean `ICONV`.

TABLE 4–19 Korean `ICONV`

Code	Symbol	Target Code	Symbol
`KSC 5601-1987`	1506	UTF-8	UTF-8
`ISO 646`	646	KSC 5601-1987	5601
`KSC 5601-1987`	EUC-KR	UTF-8	UTF-8
`KSC 5601-1987`	KSC5601	UTF-8	UTF-8
`UTF-8`	UTF-8	KSC 5601-1987	5601
`UTF-8`	UTF-8	KSC 5601-1987	EUC-KR
`UTF-8`	UTF-8	KSC 5601-1987	KSC 5601
`UTF-8`	ko-KR-UTF-8	IBM CP 933	cp 933
`UTF-8`	ko-KR-UTF-8	KSC 5601-1987	ko_KR-euc
`UTF-8`	ko-KR-UTF-8	ISO2022-KR	ko_KR-iso2022-7
`UTF-8`	ko-KR-UTF-8	KSC 5601-1987 - Johap	ko_KR-johap
`UTF-8`	ko-KR-UTF-8	KSC5601-1992 - Johap	ko_KR-johap92
`IBM CP933`	cp933	UTF-8	ko_KR-UTF-8
`KSC 5601-1987`	ko_KR-euc	UTF-8	ko_KR-UTF-8
`KSC 5601-1987`	ko_KR-euc	ISO 2022-KR	ko_KR-iso2022-7
`KSC 5601-1987`	ko_KR-euc	KSC 5601-1987 - Johap	ko_KR-johap
`KSC 5601-1987`	ko_KR-euc	KSC 5601-1992 - Johap	ko_KR-johap92
`KSC 5601-1987`	ko_KR-euc	KSC 5601-1992-Annex:4	ko_KR-nbyte
`ISO 2022-KR`	iso2022-7	UTF-8	ko_KR-UTF-8
`ISO 2022-KR`	iso2022-7	KSC 5601-1987	ko_KR-euc
`KSC 5601-1987 - Johap`	ko-KR-johap	UTF-8	ko_KR-UTF-8

TABLE 4–19 Korean `ICONV` *(Continued)*

Code	Symbol	Target Code	Symbol
`KSC 5601-1987 - Johap`	ko-KR-johap	KSC 5601-1987	ko_KR-euc
`KSC 5601-1992 - Johap`	ko-KR-johap92	`UTF-8`	ko_KR-UTF-8
`KSC 5601-1992 - Johap`	ko-KR-johap92	KSC 5601-1987	ko_KR-euc
`KSC 5601-1992 - Annex:4`	ko-KR-nbyte	KSC 5601-1987	ko_KR-euc

CHAPTER 5

Overview of `UTF-8` Locale Support

This section describes the following

Unicode Overview

The Unicode standard is the universal character encoding standard used for representation of text for computer processing. It is fully compatible with the International Standard ISO/IEC 10646-1:2000, and upcoming ISO/IEC 10646–2, 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 brings order to a state of affairs that makes exchanging international text files difficult. 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 characters Unicode can support is 1,114,112 through seventeen 16-bit planes. Each plane can support 65,536 different code points.

Among these more than one million code points the Unicode can support, currently Unicode 3.1 defines 94,140 characters at plane 0, 1, 2, and 14. Plane 15 and 16 are for characters for Private Use which is also known as User-Defined Characters. The Plane 15 and 16 together can support total 131,068 user-defined characters.

Unicode can be presented by using any of the following representation formats:

- 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 Solaris Unicode locales.

UTF-16 is a 16-bit encoding form of Unicode. In UTF-16, characters up to 65535 are encoded as single 16-bit values; characters mapped above 65535 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. In UTF-32, characters are encoded as single 21-bit values, but there is no such entity with 21-bit size. Normally 32-bit entity is used to represent UTF-32. This form is used as process code (wide-character code) in Solaris Unicode locales.

For more detail on Unicode and ISO/IEC 10646 and their various representation forms, refer to "The Unicode Standard, Version 3.0" 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, http://www.unicode.org/, also contains information and updates on the standards.

Unicode Locale: `en_US.UTF-8` Support Overview

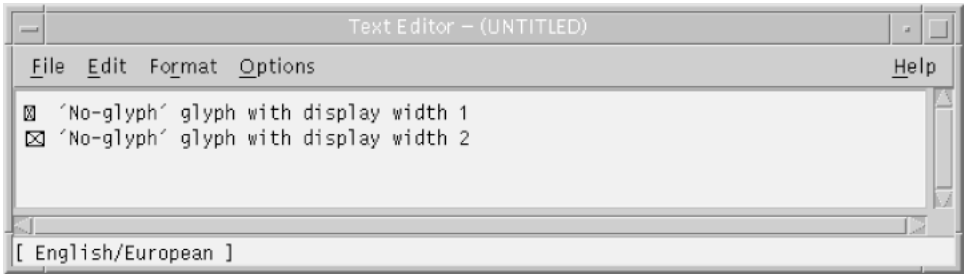
There is currently support for the Unicode 3.1 in Unicode/UTF-8 locales. Solaris 9 includes a large number of UTF-8 locales. The `en_US.UTF-8` locale is a significant Unicode locale in the Solaris 9 product. It supports and provides multiscript processing capability by using `UTF-8` as its codeset. This locale can input and output text in multiple scripts, and was the first locale with this capability in the Solaris operating environment. Other UTF-8 locales are similar in capabilities to `en_us.UTF-8` and the information 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 2.0 and ISO/IEC 10646-1.

Unicode locales in Solaris support process for every code point value, which is defined in Unicode 3.1 and ISO/IEC 10646-1 and -2. Supported scripts include not only pan-European scripts and Asian scripts but also complex text layout scripts such as Arabic, Hebrew, Hindi, and Thai. Due to limited font resources, Solaris 9 software includes only character glyphs from the following character sets:

- ISO8859-1 (most Western European languages, such as English, French, Spanish, and German)
- ISO8859-2 (most Central European languages, such as Czech, Polish, and Hungarian)
- ISO8859-4 (Scandinavian and Baltic languages)
- ISO8859-5 (Russian)
- ISO8859-6 (Arabic, including many more presentation form character glyphs)
- ISO8859–7 (Greek)
- ISO8859–8 (Hebrew)
- ISO8859-9 (Turkish)
- TIS 620.2533 (Thai, including many more presentation form character glyphs)
- ISO8859–15 (most Western European languages with euro sign)
- GB 2312–1980 (Simplified Chinese)
- JIS X 0201–1976, JIS X 0208–1990 (Japanese)
- KS C 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 a user displays characters for which the `en_US.UTF-8` locale does not have corresponding glyphs, the locale displays a "no-glyph" glyph instead, as in the following example:



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

The locale is installation-time selectable as are any other locales for an installation option and also as a system default locale.

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

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

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 several fonts that apply different input methods.

The pre-edit area displays the string that is being pre-edited. This 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 pre-edit and status areas are separate from the client's window.

For more detail, refer to XmNpreeditType resource description at *VendorShell*(3X) man page.

Note – In the Solaris 9 environment, native Asian input methods exist for Simplified/Traditional Chinese, Japanese, and Korean. This method is in addition to the current multi-script input methods for Unicode locales.

This section includes descriptions of selected input methods, how to use them, and how to switch between them.

Script Selection and Input Modes

Solaris Unicode locales support multiple scripts. Every Unicode locale has a total of fourteen input modes:

- English/European
- Cyrillic
- Greek
- Arabic
- Hebrew
- Thai
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese
- Traditional Chinese (Hong Kong)
- Hindi
- Unicode Hexadecimal and Octal code input methods
- Table lookup input method

Accessing Input Mode

You can switch into a certain input mode through a Compose key combination or through the input mode selection window. To access the input mode selection window, press mouse button 1 in the status area of your application (at the bottom left corner of your application window). This is labeled `input mode`, as shown in the example in the following figure.

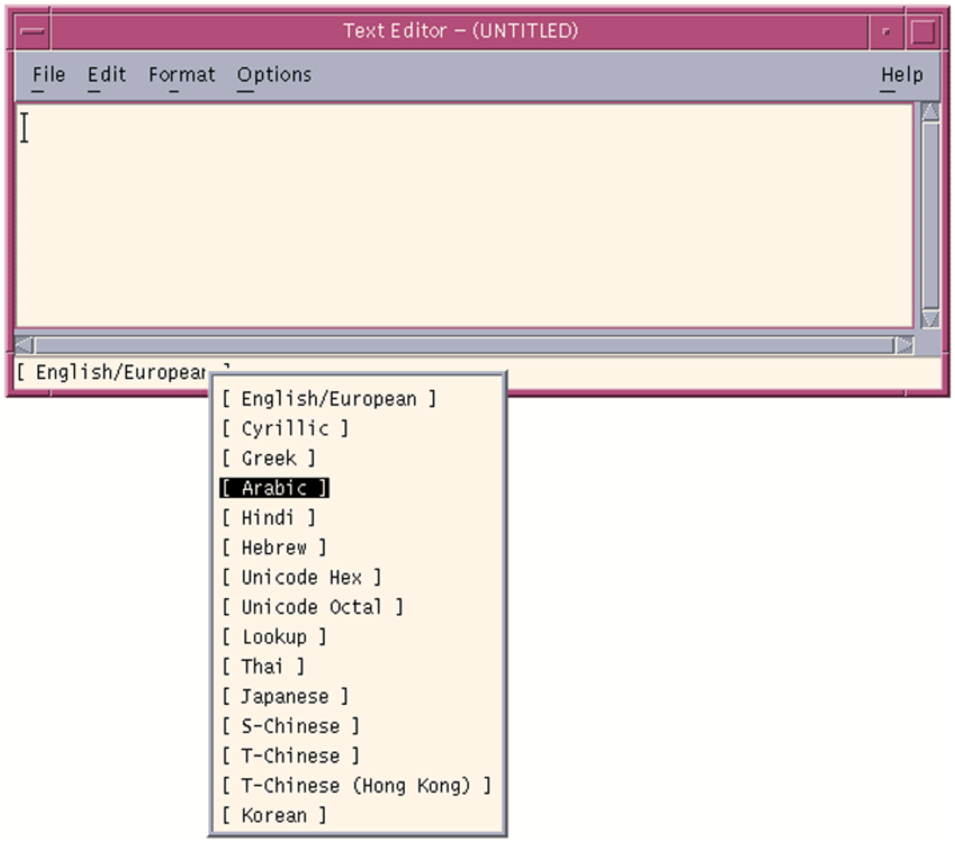


FIGURE 5–1 Input Mode Selection Window

Input Mode Switch Key Sequences

Users can also change their the current input mode to a new input mode by using the key sequences listed in the following table. There is no restriction for using these key sequences, except that if you are in one of the Asian input modes, you need to switch back to `English/European` input mode by pressing Control and Space together.

Once you're in the `English/European` input mode, you can switch freely to any other input mode by using the key sequences.

TABLE 5–1 Input Mode Switch Key Sequences

Input Mode	Input Mode Selection Choice	Key Sequences
English/European	English/European	Control Space
Cyrillic	Cyrillic	Compose c c
Greek	Greek	Compose g g
Arabic	Arabic	Compose a r
Hebrew	Hebrew	Compose h h
Thai	Thai	Compose t t
Hindi	Hindi	Compose h i
Japanese	Japanese	Compose j a
Korean	Korean	Compose k o
Simplified Chinese	Simplified Chinese	Compose s c
Traditional Chinese	Traditional Chinese	Compose t c
Traditional Chinese (Hong Kong)	Traditional Chinese (Hong Kong)	Compose h k
Unicode octal code input method	Unicode Octal	Compose u o
Unicode hexadecimal code input method	Unicode Hex	Compose u h
Table lookup input method	Lookup	Compose l l

Note – When there is no Compose key on your keyboard (for example if you are using a PC keyboard, you can substitute for the Compose key by pressing the Control key, the Shift key and the t keys together.

English/European Input Mode

The English/European input mode includes the English alphabet plus characters with diacritical marks (for example, á, è, î, õ, and ü) and special characters (such as ¡, §, ¿) from European scripts.

This input mode is the default mode for any application. The input mode is displayed at the bottom left corner of the GUI application.

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 sequence, as shown in the following examples:

- For Ä, press and release Compose, then Shift-A, and then "
- For ¿, press and release Compose, then ?, and then ?

When there is no Compose key available on your keyboard, you can substitute by simultaneously pressing the Control key, the Shift key and the t keys together.

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 there is no AltGraph key available on your keyboard, you can substitute the Alt key.

The following tables show the most commonly used Compose Sequences in Latin-1, Latin-2, Latin-3, Latin-4, Latin-5, and Latin-9 script input for the Solaris operating environment.

The following table lists the common Latin-1 compose sequences.

TABLE 5–2 Common Latin-1 Compose 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 ´
,	,	cedilla Ç
"	"	diaeresis ¨
-	^	macron ¯

TABLE 5–2 Common Latin-1 Compose Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
o	o	degree °
x	x	multiplication sign x
+	-	plus-minus ±
-	-	soft hyphen –
-	:	division sign ÷
-	a	ordinal (feminine) ª
-	o	ordinal (masculine) º
-	,	not sign ¬
.	.	middle dot ·
1	2	vulgar fraction ½
1	4	vulgar fraction ¼
3	4	vulgar fraction ¾
<	<	left double angle quotation mark «
>	>	right double angle quotation mark »
?	?	inverted question mark ¿
A	‘	A grave À
A	’	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	,	C cedilla Ç
C	o	copyright sign ©
D	-	Capital eth
E	‘	E grave È
E	’	E acute É
E	"	E diaeresis Ë

TABLE 5–2 Common Latin-1 Compose Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
E	^	E circumflex Ê
I	‵	I grave Ì
I	′	I acute Í
I	"	I diaeresis ˙
I	^	I circumflex Î
L	-	pound sign £
N	~	N tilde Ñ
O	‵	O grave Ò
O	′	O acute Ó
O	/	O slash Ø
O	"	O diaeresis Ö
O	^	O circumflex Ô
O	~	O tilde Õ
R	O	registered mark ®
T	H	Thorn Þ
U	‵	U grave Ù
U	′	U acute Ú
U	"	U diaeresis Ü
U	^	U circumflex Û
Y	′	Y acute ý
Y	-	yen sign ¥
a	‵	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

TABLE 5–2 Common Latin-1 Compose Sequences *(Continued)*

Press Compose, then Press and Release	**Press and Release**	**Result**
c	,	c cedilla ç
c	/	cent sign ¢
c	o	copyright sign ©
d	-	eth
e	‘	e grave è
e	′	e acute é
e	"	e diaeresis ë
e	^	e circumflex ê
i	‘	i grave ì
i	′	i acute í
i	"	i diaeresis ï
i	^	i circumflex î
n	~	n tilde ñ
o	‘	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
t	h	thorn Þ
u	‘	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 and Latin-4 compose sequences.

TABLE 5–3 Common Latin-2 Compose Sequences

Press Compose, then Press and Release	Press and Release	Result
a	[Spacebar]	ogonek
u	[Spacebar]	breve
v	[Spacebar]	caron
"	[Spacebar]	double acute
A	a	A ogonek
A	u	A breve
C	′	C acute
C	v	C caron
D	v	D caron
-	D	D stroke
E	v	E caron
E	a	E ogonek
L	′	L acute
L	-	L stroke
L	>	L caron
N	′	N acute
N	v	N caron
O	>	O double acute
S	′	S acute
S	v	S caron
S	,	S cedilla
R	′	R acute
R	v	R caron
T	v	T caron
T	,	T cedilla
U	*	U ring above

TABLE 5–3 Common Latin-2 Compose Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
U	>	U double acute
Z	′	Z acute
Z	v	Z caron
Z	.	Z dot above
k	k	kra
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

TABLE 5–3 Common Latin-2 Compose Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
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

The following table lists the common Latin-3 compose sequences.

TABLE 5–4 Common Latin-3 Compose 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

TABLE 5–4 Common Latin-3 Compose Sequences *(Continued)*

Press Compose, then Press and Release	Press and Release	Result
g	.	g dot above
h	>	h circumflex
j	>	j circumflex
s	>	s circumflex
u	u	u breve

The following table lists the common Latin-5 compose sequences.

TABLE 5–5 Common Latin-5 Compose 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 Sequences.

TABLE 5–6 Common Latin-9 Compose Sequences

Press Compose, then Press and Release	Press and Release	Result
o	e	Diphthong oe
O	E	Diphthong OE
Y	"	Y diaeresis

Cyrillic Input Mode

To switch to Cyrillic/Russian input mode, either press Compose c c at your keyboard, or select Cyrillic from the Input Mode Selection Window. (For information on accessing the input mode selection window, see Figure 5–1.)

The Cyrillic (Russian) keyboard layout appears in the following figure.

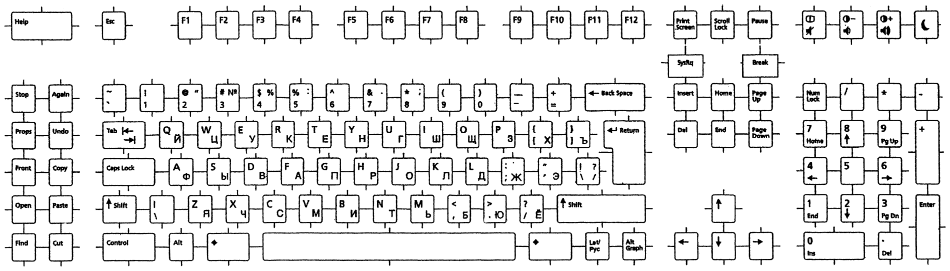


FIGURE 5–2 Cyrillic Keyboard

After you switch to Cyrillic input mode, you cannot enter English or European text. To switch back to the English/European input mode, type Control + Space from your keyboard, or select English/European input mode from the Input Mode Selection Window by clicking in the status area. See Figure 5–1

You can also switch into other input modes by typing the corresponding input mode switch key sequence.

Greek Input Mode

To switch to Greek input mode, either press Compose g g at your keyboard, or select Greek, from the input mode selection window. (For information on accessing the input mode selection window, see Figure 5–1 .)

After you switch to Greek input mode, you cannot enter English or European text. To switch back to the English/European input mode, type Control + Space from your keyboard, or select English/European input mode from the Input Mode Selection Window by clicking in the status area. The Greek Euro keyboard layout appears in the following figure.

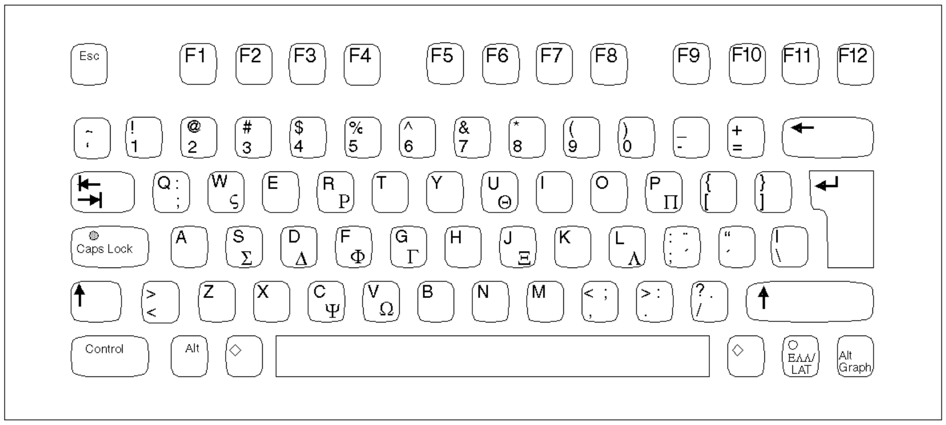


FIGURE 5–3 Greek Euro Keyboard

The following figure is for the Greek UNIX keyboard.

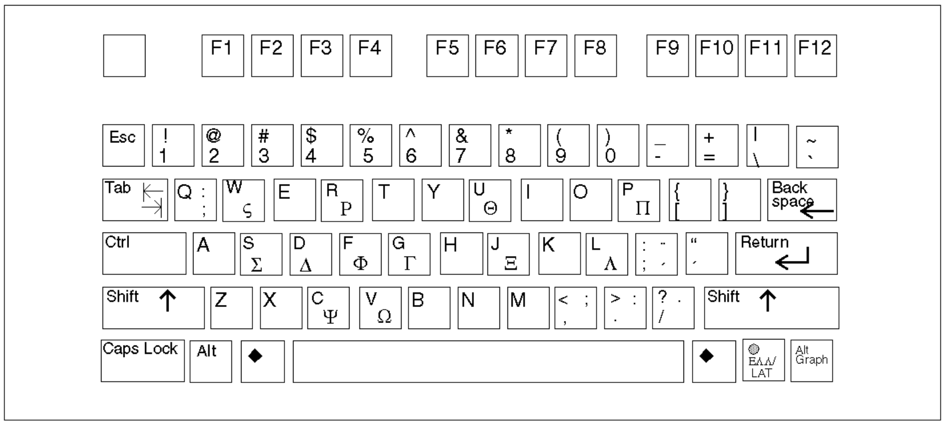


FIGURE 5–4 Greek UNIX Keyboard

Arabic Input Mode

To switch to Arabic input mode, either type Compose a r at your keyboard, or select Arabic from the input mode selection window. (For information on accessing the input mode selection window, see "Script Selection and Input Modes" on page 103

The following figure shows the Arabic keyboard layout.

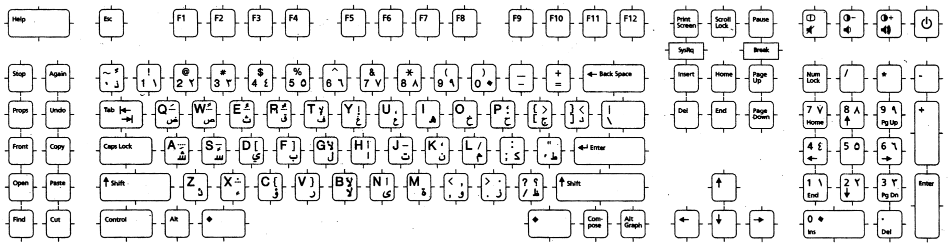


FIGURE 5–5 Arabic Keyboard

Hebrew Input Mode

To switch into Hebrew input mode, , either press Compose h h at your keyboard, or select Hebrew from the input mode selection window. (For information on accessing the input mode selection window, see "Script Selection and Input Modes" on page 103 .

The following figure shows the Hebrew keyboard layout.

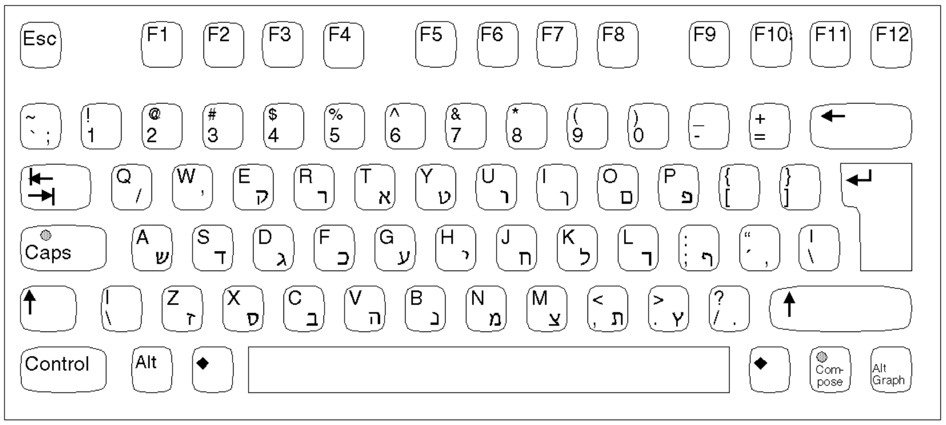


FIGURE 5–6 Hebrew Keyboard

Thai Input Mode

To switch to Thai input mode, either press Compose + T at your keyboard, or select Thai, from the input mode selection window. (For information on accessing the input mode selection window, see "Script Selection and Input Modes" on page 103.)

After you switch to the Thai input mode, you have to switch back to English/European input mode to enter English/European characters by typing Control and Space together.

The Thai keyboard layout is shown in the following figure.

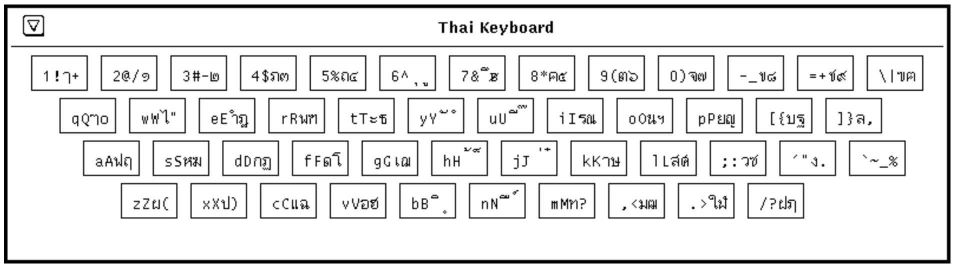


FIGURE 5–7 Thai Keyboard

Hindi Input Mode

To switch to Hindi input mode, either press Compose h i at your keyboard, or select Hindi from the input mode selection window. (For more information on accessing the input mode selection window, see "Accessing Input Mode" on page 104. After you switch to the Hindi input mode, you have to switch back to English/European input mode to enter English/European characters by typing Control and Space together.

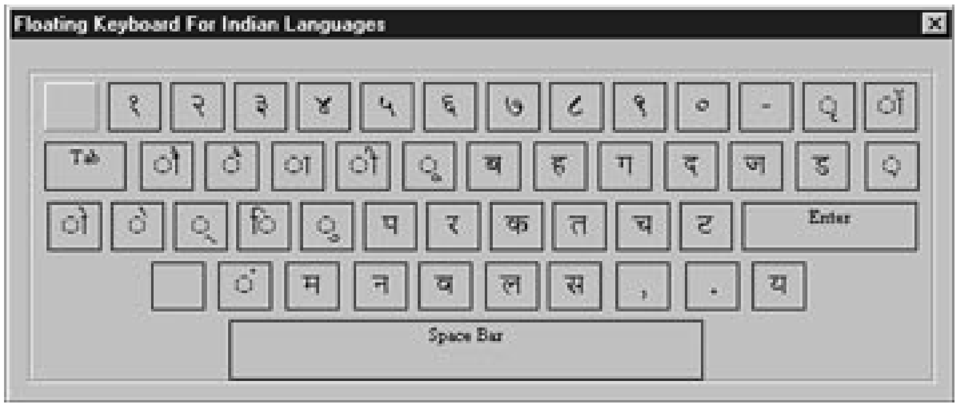


FIGURE 5–8 Hindi Keyboard

Additionally, there is a keyboard for Hindi-Shift.

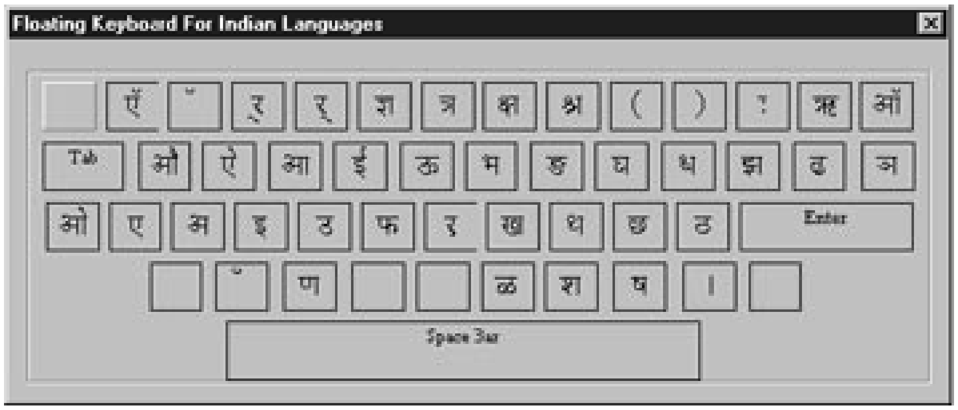


FIGURE 5–9 Hindi-Shift Keyboard

Japanese Input Mode

To switch to Japanese input mode, either press Compose j a at your keyboard, or select Japanese from the input mode selection window. (For information on accessing the input mode selection window, see “Script Selection and Input Modes” on page 103.)

To use the native Japanese input system, you need to install the Japanese locale and rebvoot the system. After you install the Japanese locale, you can use ATOK12 in all `UTF-8` locales. Wnn6 is not available in `UTF-8` locales except `ja_JP.UTF-8`.

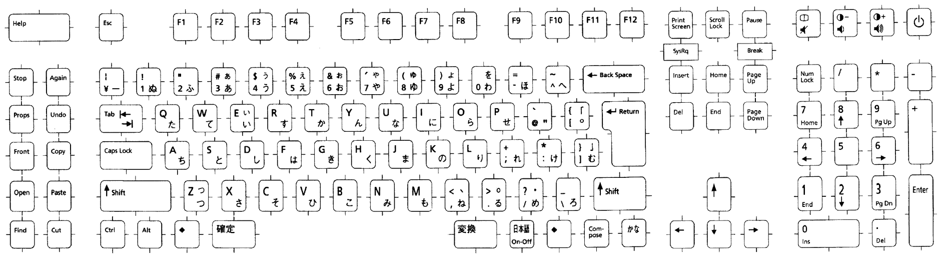


FIGURE 5–10 Japanese Keyboard

Korean Input Mode

To switch to Korean input mode, either press Compose k o at your keyboard, or select Korean from the input mode selection window. (For information on accessing the input mode selection window, see “Script Selection and Input Modes” on page 103.)

To have the native Korean input system, you need to install one or more Korean locales on your system. Once you install the Korean locale, you can use the native Korean input system. For more details on how to use the Korean Input System, refer to *Korean Solaris User's Guide*.

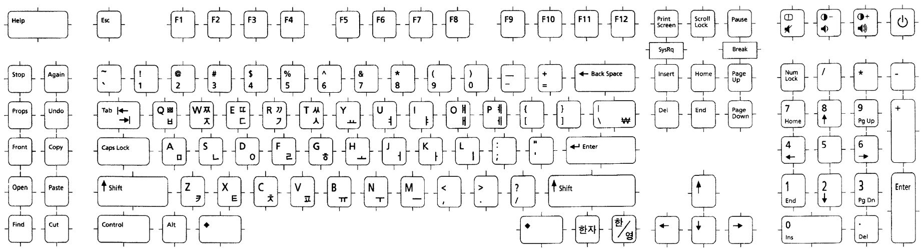


FIGURE 5–11 Korean Keyboard

Simplified Chinese Input Mode

To switch to Simplified Chinese input mode, either press Compose s c at your keyboard, or select S-Chinese from the input mode selection window. (For information on accessing the input mode selection window, see "Script Selection and Input Modes" on page 103 .)

To use the native Simplified Chinese input system, you need to install one or more Simplified Chinese locales on your system. After you install the Simplified Chinese locales, you can use the native Simplified Chinese input system. For more details on how to use Simplified Chinese Input System, refer to *Simplified Chinese Solaris User's Guide*

Traditional Chinese Input Mode

To switch to Traditional Chinese input mode, either press Compose t c at your keyboard, or select T-Chinese from the input mode selection window. (For information on accessing the input mode selection window, see ."Script Selection and Input Modes" on page 103)

To have the native Traditional Chinese input system, you need to install one or more Traditional Chinese locales at your system. After you install the Traditional Chinese locales, you can use the native Traditional Chinese input system. For more details on how to use the Traditional Chinese Input System, refer to *Traditional Chinese Solaris User's Guide*.

Traditional Chinese (Hong Kong) Input Mode

To switch to Traditional Chinese input mode, either press Compose h k at your keyboard, or select T-Chinese (Hong Kong) from the input mode selection window. (For information on accessing the input mode selection window, see “Script Selection and Input Modes” on page 103 .)

To have the native Traditional Chinese (Hong Kong) input system, you need to install one or more Traditional Chinese (Hong Kong) locales at your system. After you install the Traditional Chinese (Hong Kong) locales, you can use the native Traditional Chinese (Hong Kong) input system.

Unicode Hexadecimal and Octal Code Input Method Input Modes

To switch to Unicode hexadecimal code input method input mode, either press Compose u h at your keyboard, or select Unicode Hex from the input mode selection window. To switch to the octal number system, press Compose u o or select Unicode Octal. (For information on accessing the input mode selection window, see “Script Selection and Input Modes” on page 103.)

To use these input modes, you need to know about either the hexadecimal or the octal code point values of the characters. Refer to *The Unicode Standard, Version 3.0* for the mapping between code point values and characters.

If you are in the Unicode hexadecimal code input method input mode, for example, to input a character you would type four hexadecimal digits. Some sample hexadecimal values are:

- 00a1 for Inverted Exclamation Mark
- 03b2 for Greek Small Letter Beta
- ac00 for a Korean Hangul Syllable
- 30a1 for Japanese Katakana Letter A
- 4e58 for a Unified Han character

You can use both uppercase and lowercase letters of A, B, C, D, E, and, F for hexadecimal digits. If you prefer the octal number system instead of hexadecimal numbers, you can input octal digits, 0 to 7. If you mistype a digit or two, you can delete the digits by using the Delete or Backspace key.

Table Lookup Input Method Input Mode

To switch to table lookup input method input mode, either press Compose l l at your keyboard, or select Lookup from the input mode selection window. (For information on accessing the input mode selection window, see “Script Selection and Input Modes” on page 103 .)

The following figure shows the available lookup character groups.

The second lookup window shows candidates for the group only display 80 candidates at a time at a maximum. Press Control+n for the next set of candidates and also Control+p for previous set of candidates.

System Environment

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

Locale Environment Variable

To use the `en_US.UTF-8` locale desktop environment, choose the locale first. Be sure you have the `en_US.UTF-8` locale installed on your system.

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 other categories are not set (or are set to `en_US.UTF-8`). because the `LANG` environment variable has a lower priority than other environment variables (such as `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_NUMERIC`, `LC_MONETARY`, and `LC_TIME`) have at setting the locale. See the `setlocale`(3C) man page for more details about the hierarchy of environment variables.

To check current locale settings in various categories, use the `locale(1)` 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=
```

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`.

```
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=
```

TTY Environment Setup

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

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

The following table shows STREAMS modules supported by the `en_US.UTF-8` locale in the terminal environment.

TABLE 5–7 32–bit STREAMS Modules Supported by `en_US.UTF-8`

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

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

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

64-bit STREAMS module	Description
`/usr/kernel/strmod/sparcv9/u8lat1`	Code conversion STREAMS module between `UTF-8` and ISO8859-1 (Western European)

TABLE 5–8 64–bit STREAMS Modules Supported by `en_US.UTF-8` *(Continued)*

64-bit STREAMS module	Description
`/usr/kernel/strmod/sparcv9/u8lat2`	Code conversion STREAMS module between `UTF-8` and ISO8859-2 (Eastern European)
`/usr/kernel/strmod/sparcv9/u8koi8`	Code conversion STREAMS module between `UTF-8` and KOI8-R (Cyrillic)

Loading a STREAMS Module at Kernel

To load a STREAMS module at kernel, first become root:

```
system% su
Password:
```

To determine whether you are running a 64-bit Solaris or 32-bit Solaris system, use the `isainfo(1)` utility as follows:

```
system# isainfo -v
64-bit sparcv9 applications
32-bit sparc applications
```

If the command returns this information, you are running the 64-bit Solaris system. If you are running the 32-bit Solaris system, the utility shows the following:

```
system# isainfo -v
32-bit sparc applications
```

Use `modinfo`(1M) to be certain that your system has not already loaded the STREAMS module:

```
system# modinfo | grep u8lat1 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)
```

If the module is already installed, you do not need to load it. However, if the module has not yet been loaded, use `modload`(1M) as follows:

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

This loads the 32–bit `u8lat1` STREAMS module at the kernel so you can push it onto a stream. If you are running the 64–bit Solaris product, use `modload`(1M) as follows:

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

The STREAMS module is loaded at the kernel and you can now push it onto a stream.

To unload a module from the kernel, use `modunload`(1M), as shown below. In this example, the `u8lat1` module is being unloaded.

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

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

Unlike in previous releases of the Solaris operating environment, the `dtterm`(1) Terminal and any other terminals that support input and output of the `UTF-8` codeset do not need to have any other additional STREAMS module in their stream. `ldterm`(7M) module is now codeset independent and supports Unicode/`UTF-8` as well.

For the proper terminal environment setup for the Unicode locales, use the `stty(1)` utility as follows:

```
system% stty defeucw
```

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.

To set up the STREAMS configuration shown above, use *strchg(1M)*, as follows:

```
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). To see the current configuration, use `strconf`(1), as follows:

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

To reset the original configuration, set the STREAMS configuration as follows:

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

Setting Terminal Options

To set up the `UTF-8` text line edit behavior on TTY, you must first set proper terminal options by using `stty`(1), as follows:

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

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

You can also query the current settings using `stty`(1) with the `-a` option, as shown below:

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

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/mystream$$
     /bin/rm -f /tmp/mystream$$
     /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. Note that the second `_EOF` should start from the first column of the file.

You can also query the current settings using:`stty`(1) with the `-a` option, as shown below:

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

Code Conversions

Unicode locale support adds various code conversions among major codesets of many countries through `iconv`(1), `iconv`(3C), and *sdtconvtool(1)*.

Note – In the Solaris 9 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 detail on this utility, refer to `geniconvtbl` (1) and `geniconvtbl`(4) man pages.

The available `fromcode` and `tocode` names that can be applied to `iconv(1)`, `iconv_open(3C)`, and `sdtconvtool(1)` are shown in the tables in Appendix A. For more details on `iconv` code conversion, see the `iconv`(1), `iconv_open`(3C), `iconv` (3) , `iconv_close` (3C) `geniconvtbl` (1) `geniconvtbl` (4) and *sdtconvtool(1)* man pages. For more information on available code conversions, see the *iconv_en_US.UTF-8(5), iconv_ja(5), iconv_ko(5),iconv_zh(5),* and *iconv_zh_TW(5)* man pages. Also see Appendix A.

Note – UCS-2, UCS-4, UTF-16 and UTF-32 are all fixed-width Unicode/ ISO/IEC 10646 representation forms that recognize Byte Order Mark (BOM) characters defined in the Unicode 3.1 and ISO/IEC10646-1:2000 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 all 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. They do not recognize the BOM character.

Note – For associated scripts/languages of ISO8859-* and KOI8-*, see http://czyborra.com/charsets/iso8869.html.

DtMail

As a result of increased coverage in scripts, Solaris 9 DtMail running in the `en_US.UTF-8` locale supports the following MIME character sets.

- US-ASCII (7-bit US ASCII)
- UTF-8 (UCS Transmission Format 8 of Unicode)
- UTF-7 (UCS Transmission Format 7 of Unicode)
- 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-8859-15 (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 of Unicode)
- UTF-16BE (UTF-16 Big-Endian of Unicode)
- UTF-16LE (UTF-16 Little-Endian of Unicode)
- 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 of Unicode)
- UTF-32BE (UTF-32 Big-Endian of Unicode)
- UTF-32LE (UTF-32 Little-Endian of Unicode)

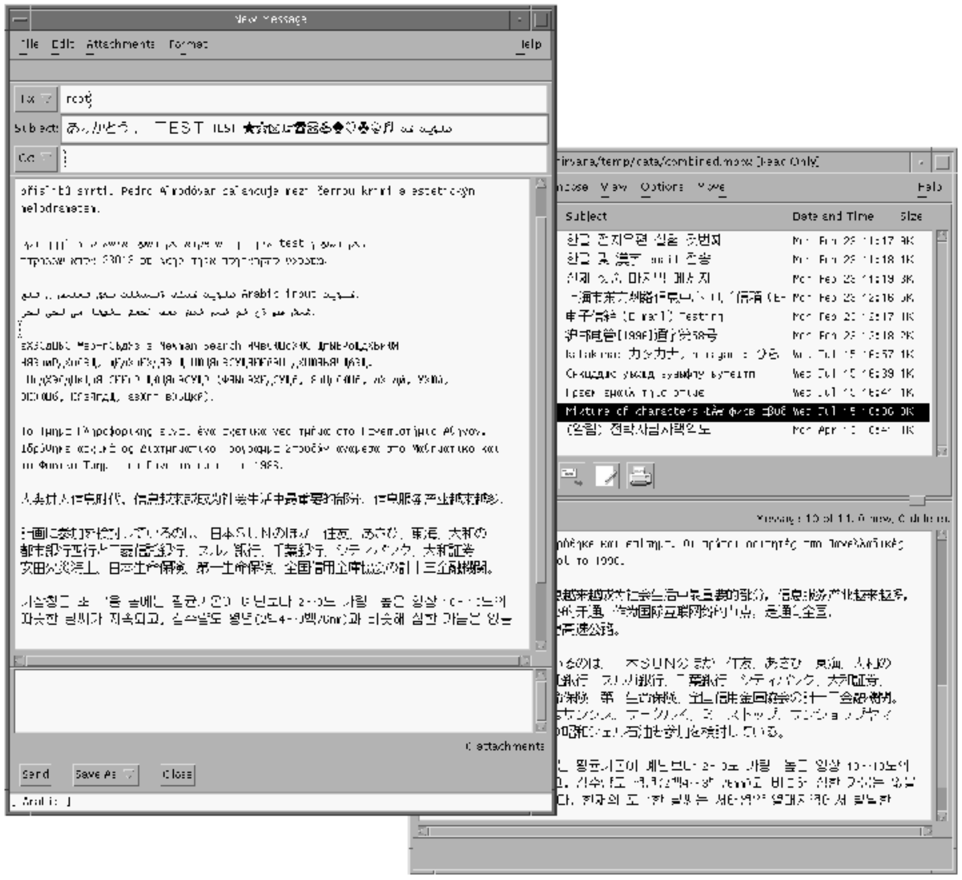
This support allows users to view virtually any kind of email encoded in various MIME character sets from any region of the world in a single instance of DtMail. The decoding of received email is done by DtMail, which looks at the MIME character set and content transfer encoding provided with the email. Windows-125x MIME charsets are supported.

However, in case of sending, you need to specify a MIME character set that is understood by the recipient mail user agent (in other words, mail client), unless you want to use the default MIME character set provided by the `en_US.UTF-8` locale. To switch the character set of outgoing email, at the New Message window, type either CONTROL+Y, or click the Format menu button and then click again on the Change Char Set button. The next available character-set name displays at left bottom corner on 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 MIME character set to the `UTF-8` that can represent any character.

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

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


New Message
File Edit Attachments Format Help
To: root
Subject: ありがとう。 TEST
Cc:
0 attachments
Send Save As Close
/rirsara/temp/cata/combined.mow [Read Only]
Compose View Options Move Help
Subject Date and Time Size
Message 10 of 11, 0 new, 0 deleted

Pedro Almodóvar [illegible] melodrametem.

[illegible] test [illegible]

[illegible] Arabic input [illegible]

Subject	Date and Time	Size
[illegible]	Mon Feb 23 11:17	3K
한글 및 漢字 mail [illegible]	Mon Feb 23 11:18	1K
[illegible]	Mon Feb 23 11:19	3K
[illegible]	Mon Feb 23 12:16	5K
電子信件 (E-mail) Testing	Mon Feb 23 12:17	1K
[illegible]	Mon Feb 23 12:18	2K
katakana: カタカナ, hiragana: ひら	Wed Jul 15 16:57	1K
[illegible]	Wed Jul 15 16:39	1K
Greek [illegible]	Wed Jul 15 16:41	1K
Mixture of characters [illegible]	Wed Jul 15 10:06	3K
[illegible]	Mon Apr 13 09:41	1K

FIGURE 5–12 dtMail

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: 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 Overview" on page 100.

Because the Solaris 9 environment supports the CDE desktop environment, each character set has a guaranteed set of fonts.

The following is a list of the Latin-1 fonts that are supported in the Solaris 9 product:

```
-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);
```

XmFontList Definition as 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 Motif APIs to support writing systems that require complex transformations between logical and physical text representations, such as Arabic, Hebrew, and Thai. CTL Motif provides character shaping, such as ligatures, diacritics, and segment ordering , and supports the transformation of static and dynamic text widgets. It also supports right-to-left and left-to-right text orientation and tabbing for dynamic text widgets . Because text rendering is handled through the rendition layer, other widget libraries can be easily extended to support CTL.

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 it is rendered. Applications that support CTL must include additional resources, as described in the CTL documentation.

Specifically, `XmCTL` 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.
- Bi-directionality.
- Symmetrical swapping.
- Numeral shaping.
- String validation.

Overview of CTL Architecture

The CTL architecture is organized as shown in the following diagram. `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 is built to support new languages by adding a new locale-specific engine. In other words, support for Thai and Vietnamese can be added without altering Motif or `Dt Apps`.

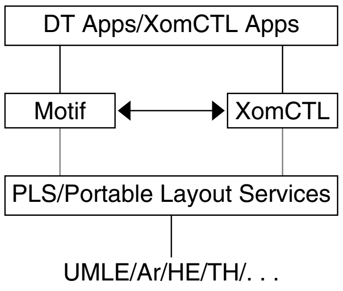


FIGURE 6–1 CTL Architecture

CTL Support for X Library Based Applications

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

New XOC Resources

The following XOC Resources are provided by the Solaris 9 environment:

- XNText.
- Allows user to set the text buffer on which CTL operation needs to be performed.
- XNTextLayoutNumGlyphs.
- Provides number of glyphs for the text in the text buffe.r
- XNTextLayoutModifier.
- Same as XmNLayoutModifier of motif.
- XNTextLayoutProperty.
- Same as PLS Property, Input-to-output and Output-to-input.
- XNTextLayoutMapInpToOut.
- Same as PLS Property, Input-to-output and Output-to-input.
- XNTextLayoutMapOutToInp
- Same as PLS Property, Input-to-output and Output-to-input

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

Changes in Motif to Support CTL Technology

XmDirection

The `XmNlayoutDirection` resource[1] controls object layout. It interacts with the orientation value of the `LayoutObject` in the manner described below.

1 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`.

Layout Direction

First, when `XmNlayoutDirection` is specified as `XmDEFAULT_DIRECTION`, then the widget's layout direction 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:

- Directly, if the component is of type `XmSTRING_COMPONENT_LAYOUT_PUSH` or `XmSTRING_COMPONENT_DIRECTION.`
- Indirectly, if the component is of type `XmSTRING_COMPONENT_LOCALE_TEXT`, `XmSTRING_COMPONENT_WIDECHAR_TEXT`, or `XmSTRING_COMPONENT_TEXT`, from the component's associated `XmRendition`'s and associated `LayoutObject`.

Second, if `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.

For More Information

For more information, see *CAE Specification: Portable Layout Services: Context-dependent and Directional Text*. The Open Group: Feb 1997; ISBN 1-85912-142-X; document number C616.

This document describes a set of portable functions for handling context-dependent and bidirectional text transformations as a logical extension to the existing POSIX locale model. It is intended for system and application programmers who want to provide support for complex-text languages.

`XmStringDirection`

`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, it should look to see if the string was created with an explicit direction (XmStringDirection). If there is 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.

See also XmRendition and XmDirection.

XmRendition

CTL adds the following new pseudo resources to XmRendition:

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 it is not, then the XmNlayoutAttrObject and XmNlayoutModifier resources are ignored.

When the value of this resource is XmFont_IS_XOC, and if 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 LayoutObject associated with the XOC.

XmNlayoutAttrObject — Specifies the layout AttrObject argument to be 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` above 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 associated with the XOC associated with 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 LayoutObject associated with the XOC associated with this Rendition. This is the mechanism for configuring 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`, the user can 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`, `Numerals`

  `Orientation` and `Context` should not be modified dynamically; you can safely modify `ImplicitAlg`, `Swapping`, and `Numerals`.
- Editing behavior: `CheckMode`

XmText, XmTextField

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). All of 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 `CSText`'s in that they 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).

There is also a new resource 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`}, and is listed below.

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` (in the `XmNrenderTable` resource) used for this 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 below or to the existing logical actions.

Character Orientation Action Routines

All of the actions in the following list query the orientation of the character in the direction specified. If the direction is left-to-right, they call the corresponding `next-/forward-` or `previous-/backward-` variants:

- `forward-cell()`
- `backward-cell()`

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, bi-directional 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 of the succeeding characters whose "new cell indicator"[2] is 0.

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. This means that 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 whose "new cell indicator" is 0 (that is, diacritics or ligature fragments).

This means that deletion operates either from the logical/input buffer side, or from the display cell level of the physical/output side. There is no mode for a strict, physical character-by-character deletion, since there is no one-to-one correspondence 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 `XmText` action routines are as follows:

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

`left-character(extend)`	If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and is called without arguments, it moves the insertion cursor back logically by a character. If the insertion cursor is at the beginning of the line, it moves the insertion cursor 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 it is called with an `extend` argument, it moves the insertion cursor, 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 can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.
`left-word(extend)`	If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and is called without any arguments, and the insertion cursor is at the logical starting of the word, it moves the insertion cursor to the logical starting of the logical preceding word, if one exists; otherwise, the insertion cursor position doesn't change. If the insertion cursor is in the word but not at the logical start of the word, it moves the insertion cursor to the logical start of the word. If the insertion cursor is at the logical start of the line, it moves the insertion cursor to the logical start of the logical last word in the previous line, if one exists; otherwise, the insertion cursor position doesn't change.

	If the `XmNeditPolicy` is `XmEDIT_VISUAL` and is called without arguments, it moves the insertion cursor to the first non-white-space character after the first white-space character to the left or after the beginning of the line. If the insertion cursor is already at the beginning of the word, it moves the insertion cursor to the beginning of the previous word. If the insertion cursor is already at the beginning of the line, it moves to the starting of the last word in the previous line. If called with an argument of `extend`, it moves the insertion cursor, 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 it is called with an `extend` argument, this 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 is called without any arguments, it moves the insertion cursor logically forward by a character. If the insertion cursor is at the logical end of the line, it moves the insertion cursor to the logical starting 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`, it 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 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 is called without any arguments, it moves the insertion cursor to the logical starting of the logical succeeding word, if one exists; otherwise, it 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, it moves the cursor to the logical first word in the next line, if one exists; otherwise, it moves to the logical end of the current word.

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

If called with an argument of `extend`, it moves the insertion cursor, 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 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 there is a non-null selection, it deletes the selection; otherwise it deletes the character left of the insertion cursor. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection and `XmNpendingDelete` is set to True, it deletes the selection; otherwise, it deletes the character left of the insertion cursor. This can impact the selection. The `delete-left-character()` action produces calls to the `XmNmodifyVerifyCallback` procedures with reason value `XmCR_MODIFYING_TEXT_VALUE` and the `XmNvalueChangedCallback` procedures with 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 there is a non-null selection, it deletes the selection; otherwise, it deletes the character 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 right of the insertion cursor is deleted. This can impact 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`.
`delete-left-word()`	If the `XmNeditPolicy` is `XmEDIT_VISUAL`, it is equivalent to `delete-prev-word()`. If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then in normal mode, if there is a non-null selection, it deletes the selection; otherwise, it deletes the characters

	left of the insertion cursor to the next space, punctuation character, tab, or beginning-of-line character. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection; otherwise it deletes the characters left of the insertion cursor the right space, tab, or beginning-of-line character. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, the `XmNpendingDelete` is set to True, and the selection is deleted; otherwise, it deletes the character left of the insertion cursor, the right space, tab, or beginning of new-line character. This can impact the selection.
`delete-right-word()`	If the `XmNeditPolicy` is `XmEDIT_VISUAL`, it is equivalent to `delete-right-word()`. If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then in normal mode, if there is a non-null selection, it deletes the selection; otherwise, it deletes the characters right of the insertion cursor to the next space, punctuation character, tab, or end-of-line character. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, `XmNpendingDelete` is set to True, and deletes the selection; otherwise, it deletes the characters right of the insertion cursor to the next space, tab, or end-of-line character. This can impact the selection.
`kill-left-character()`	If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, it is equivalent to `kill-prev-char`. If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then in normal mode, if there is a non-null selection, it deletes the selection; otherwise, it kills the character left of the insertion cursor and stores the character in the cut buffer. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, `XmNpendingDelete` is set to True, and deletes the selection; otherwise, it deletes the character left of the insertion cursor. This can impact the selection.

	The `kill-left-character()` action produces calls to the `XmNmodifyVerifyCallback` procedures with the reason value `XmCR_MODIFYING_TEXT_VALUE`, and produces the `XmNvalueChangedCallback` procedures with the reason value `XmCR_VALUE_CHANGED`.
`kill-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 there is a non-null selection, it deletes the selection; otherwise, it deletes the character right of the insertion cursor and stores it in the cut buffer. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, the `XmNpendingDelete` is set to True and deletes the selection; otherwise, it deletes the character right of the insertion cursor. This can impact the selection. The `kill-right-character()` action produces calls to the `XmNmodifyVerify-Callback` procedures with reason value `XmCR_MODIFYING_TEXT_VALUE`, and produces calls to the `XmNvalue-ChangedCallback` procedures with reason value `XmCR_VALUE_CHANGED`.
`kill-left-word()`	If the `XmNeditPolicy` is `XmEDIT_VISUAL`, it is equivalent to `delete-prev-word().` If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then in normal mode, if there is a non-null selection, it deletes the selection; otherwise, it deletes the characters left of the insertion cursor to the next space, punctuation character, tab, or beginning-of-line character. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection; otherwise it deletes the characters left of the insertion cursor the right space, tab, or beginning-of-line character and stores it in

	the cut buffer. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, `XmNpendingDelete` is set to True and deletes the selection; otherwise it deletes the characters left of the insertion cursor, the right space, tab, or beginning of new-line character. This can impact the selection.
`kill-right-word()`	If the `XmNeditPolicy` is `XmEDIT_VISUAL`, it is equivalent to `delete-right-word()`. If the `XmNeditPolicy` is `XmEDIT_LOGICAL`, then in normal mode, if there is a non-null selection, it deletes the selection; otherwise, it deletes the characters right of the insertion cursor to the next space, tab, or end-of-line character. In add mode, if there is a non-null selection, the cursor is not disjointed from the selection, `XmNpendingDelete` is set to True, and deletes the selection; otherwise, it deletes the characters right of the insertion cursor to the next space, punctuation character, tab, or end-of-line character and stores in the cut buffer. This can impact the selection.

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 follows:

`prev-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, it 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 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 it moves it 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 can produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the *Motif Programmer's Reference* for more information.

XmTextFieldGetLayoutModifier

Purpose

A `TextField` function that returns the layout modifier string that reflects the state of the layout object tied to its rendition.

Synopsis

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

Description

`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 only the changed values.

Return Value

Returns the layout object modifier values in the form of a String value.

Related Information

`XmTextField`

`XmTextGetLayoutModifier`

Purpose

A `Text` function that returns the layout modifier string that reflects the state of the layout object tied to its rendition.

Synopsis

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

Description

`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.

Return Value

Returns the layout object modifier values in the form of a String value.

Related Information

`XmText`

XmTextFieldSetLayoutModifier

Purpose

A `TextField` function that sets the layout modifier values, which changes the behavior of the layout object tied to its rendition.

Synopsis

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

Description

`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.

Related Information

`XmTextField`

XmTextSetLayoutModifier

Purpose

A `Text` function that sets the layout modifier values, which changes the behavior of the layout object tied to its rendition.

Synopsis

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

Description

`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.

Related Information

`XmText`

XmStringDirectionCreate

Synopsis

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

Description

`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, all you need to do is set the appropriate value for the `XmNlayoutDirection` resource. You need not create compound strings with specific direction components. When the application renders an `XmString`, it should look to see if the string was created with an explicit direction (`XmStringDirection`). If there is 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`.

Related Information

See also `XmRendition`, `XmDirection`.

UIL

The following table shows the UIL arguments.

TABLE 6–3 UIL

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

How to Develop CTL Applications

The following sections show how to develop your CTL Applications.

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. Manager and Primitive widgets (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. 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 is alignment, be sure to specify `XmRIGHT_TO_LEFT_TOP_TO_BOTTOM`, which specifies that the components are laid out from right-to-left first and then top-to-bottom, and results in the desired behavior.

Furthermore, the behavior of `XmText` and `TextField` widgets is 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 can be illustrated using the example below.

The input string used in the illustration is:

A B و ض

The XmNlayoutModifier string @ls orientation= setting values for this illustration are shown below, in the left column.

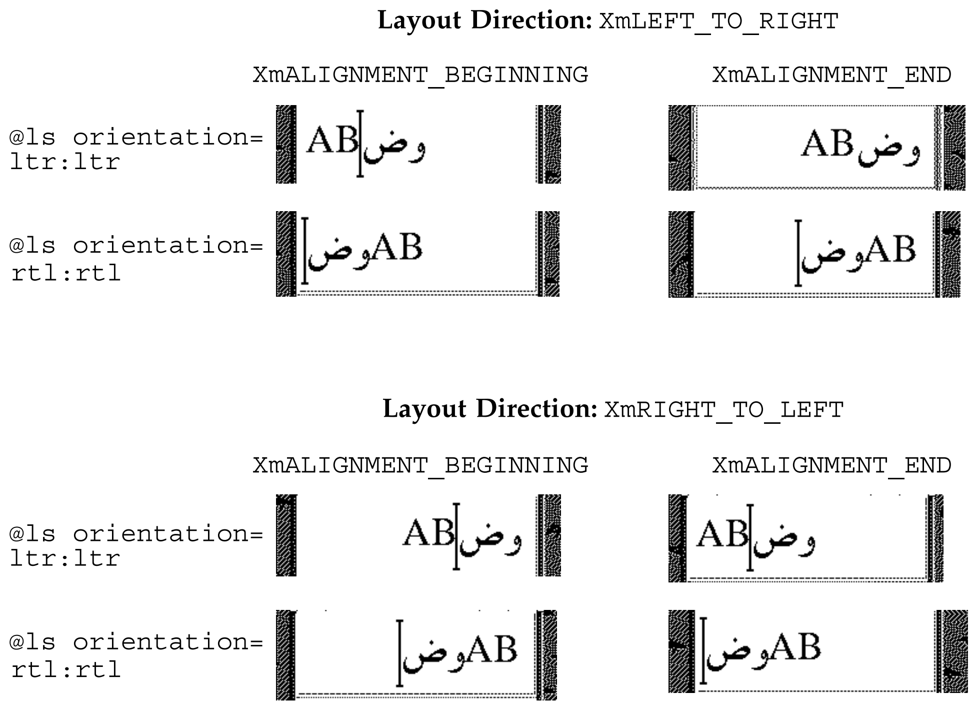


FIGURE 6–2 Layout Direction

As the illustration shows, XmNAlignment dictates whether the text is flush right or left in conjunction with the layout direction. On the other hand, 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.

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 “unicode-3.0.”

The locale-specific layout module transforms its input text (in this example encoded in ISO 8859-6) in an output buffer of physical characters encoded using the 16-bit Unicode 3.0 codeset. Because an explicit layout locale has been specified, this text is rendered properly independent of the runtime locale setting.

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

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 defaults to NULL. This means the LayoutObject 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. It then modifies the Rendition's LayoutObject's ImplicitAlg value 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);
XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
    ....
n = 0;
XtSetArg(args[n], XmNlayoutModifier, "@ls implicitalg=basic");
     n++;
XmRenditionUpdate(rendition, args, n);
```

Related Information

See also XmDirection, XmText.

Creating a Render Table in a Resource File

Renditions and render tables can be specified in resource files. For a properly internationalized application, in fact, this is the preferred method. 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 examples illustrate 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.

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;
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);
```

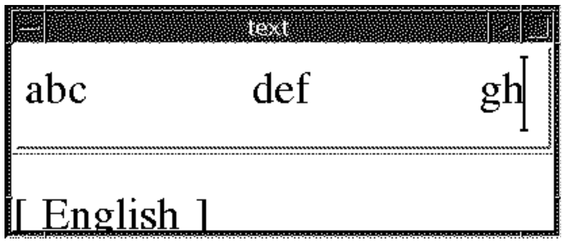
Horizontal Tabs

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

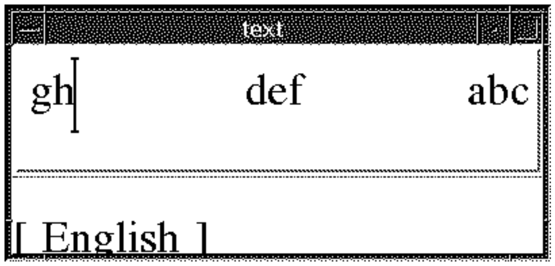
The tab measurement is the distance from the left margin of the compound string display. It is from the right margin, if the layout direction is right-to-left. Regardless of the direction of the text (Arabic right-to-left or English left-to-right), the tab inserts space to the right or left, as specified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



Layout Direction: `XmLEFT_TO_RIGHT`



Layout Direction: `XmRIGHT_TO_LEFT`

FIGURE 6–3 Tabbing Behavior

Mouse Selection

The user makes a primary selection with SELECT (the left mouse button). Pressing SELECT deselects any existing selection and moves the insertion cursor and the anchor to the position in the text where the button is pressed. Dragging SELECT 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 if the text selected is bi-directional, the selected text is not contiguous visually and is a collection of segments. This is because 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 bi-directional text, when rendered does not result 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. So the sequence of selection, deletion, and insertion of bi-directional 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 Shift-arrow keys allows the selection of text.

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 if the text selected is bi-directional, the selected text is not contiguous visually. It is also a collection of segments. This is because the text in the logical buffer does not have one-to-one correspondence with the display. As a result, the contiguous buffer of logical characters of bi-directional text, when rendered, does not result 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. So the sequence of selection, deletion, and insertion of bi-directional text at the same cursor point does not result in the same string.

Text Resources and Geometry

Text has several resources that relate to geometry, including the following:

- 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`, and can also specify the attributes of the layout object, such as `XmNlayoutAttrObject`, usually 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, but the first character in the next line is shaped independently of the previous character in the logical buffer.

Porting Instructions

The new CTL-enabled Motif library can be found in `/usr/dt/lib/libXm.so.4`. If your application links to `libXm.so.3` (`ldd app_name` shows which library the application is linking to), then it does not support Complex Text Layout (CTL). In order 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 should be set during compilation.
2. Recompile the existing application. It automatically links with the CTL-enabled Motif library `libXm.so.4`
3. Add the following 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.

If you want the 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 section describes

Printing for UTF-8

An enhanced `mp`(1) print filter is available in the Solaris 9 environment that can print various input file formats including flat text files written in `UTF-8`. It uses TrueType and Type 1 scalable fonts and X11 bitmap fonts available on the Solaris system.

The output from the utility is standard PostScript and can be sent to any PostScript printer.

To use the utility, type the following:

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

You can also use the utility as a filter, since the utility 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 line 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 filtername -F pathname
```

The command tells LP that a converter (in this case, xutops) is available through the filter description file named pathname. The path name can be determined as follows:

```
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 for more detail.

mp Print Filter Enhancement Overview

The mp print filter is enhanced in the Solaris 9 release. The latest mp can work internally in three different modes to produce the output file in a locale to print international text.

- mp working with locale specific font configuration file mp.conf.
- mp working with locale specific postscript prologue file prolog.ps.
- mp working as a Xprt (X Print Server) client.

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

Using `mp` with the Locale Specific Font Configuration File `mp.conf`

If `-D` or `-P` option is not given in the command line, this printing method is the default method, unless the `prolog.ps` file (which forces `mp` to print using postscript embedded fonts in `prolog.ps` file) is present in either of the`/usr/openwin/lib/locale/$LANG/print` or `/usr/lib/lp/locale/$LANG/mp` directories.

This method makes use of `/usr/lib/lp/locale/$LANG/mp/mp.conf` font configuration file. You may not need to change this file unless you need to print using alternate fonts. This customization is discussed in the nest procedure. This file can be configured with TrueType, Type 1 or `.pcf` fonts. `/usr/lib/lp/locale/C/` contains `.ps` print page layout files common for this mode of printing as well as the next method. A description of how to customize these files is provided in "Customizing Existing `prolog` Files and Adding New `prolog` Files" on page 171.

Using `mp` with the Locale Specific Postscript Prologue Files

If `-D` or `-P` is not given in the command line, and `/usr/openwin/lib/locale/$LANG/print/prolog.ps` exists, then it 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 the next section.

Using `mp` as an Xprt (X Print Server) Client

This support enables `mp` to print output for any printer connected to the network supported by an X Print Server. PostScript and many versions of PCL are also supported with this command.

If `-D` (target printer) or `-P` (target printer) is given in the command line, then `mp` works as a X Print Server Client, provided a properly configured X Print Server is running in any machine in the network, and the `XPDISPLAY` variable in the host machine is set to that print server.

Refer to the *Xprt* man page for instructions on configuring `Xprt`. The `/usr/lib/lp/locale/C/mp` directory contains `.xpr` print page layout files for `mp` working as the `Xprt` client. These are sample files created for 300 dpi printers. If the target printer has a different dpi, these files must be customized to the target printer's resolution as described in the following `.xpr` file customization section.

The following example describes how to print using this mechanism.

If machine 'machine_a' is running `Xprt` (`/usr/openwin/bin/Xprt :1 -fp` < comma separated font paths>). To print from `machine_b`, set the XPDISPLAY in 'machine_b' to `machine_a:1`, and optionally use `mp -D` . The print page is directly spooled to the printer.

Localization of the Configuration File

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

The system default configuration file `/usr/lib/lp/locale/$LANG/mp/mp.conf` where $LANG is a locale environment variable in the locale in which printing occurs. Users can have a personal configuration file that can be specified by 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 that the intermediate code points be PLS output. The default intermediate code generated by the `mp`(1) 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 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 form:

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

keyword
For example,

```
FontNameAlias   prnHelveticaR   Type1   Helvetica
```

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 given the alias name iso88591R.

font type
Specify 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
Give the absolute path name for the font files here. For type1 printer-resident fonts, just specify the font name, such as 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	for this section is FontGroup.
fontgroupname	is the group name for the fonts.
GroupType	is the font type. Create font groups for the same type of fonts only (PCF, Type1, TrueType).
Roman	is the Roman Font name in the font group
Bold	is the Bold Font name in the font group.
Italic	is the Italic Font name in the font group.
BoldItalic	is the BoldItalic Font name in the font group

Note – 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/news articles. If only the Roman font is defined, it is used in place of other fonts.

Mapping

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

keyword	for this section is MapCode2Font.
range_start	is a 4–byte hex value that starts with 0x, which indicates the start of the code range to map to one or more font group.
range_end	indicates the end of the code range to map. It can be a '-' in which case only a single intermediate code point is mapped to the target font.
group	is a Type1, PCF, or TrueType font group, with which the presentation forms are to be printed

Associating

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 fonts encoding. Each line in this section is as follows.

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

Note – The Current TrueType Engine employed by `mp` (1) is capable of dealing only with format 4 and PlatformID 3 `cmap`. That is, 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 Solaris environment are obeying these restrictions, you can map all TrueType fonts in Solaris software within the `mp.conf` file.

When you create a shared object for mapping a font that corresponds to an X Logical Fonts Description (XLFD) , consider the following. If you are mapping a `pcf`/type1 font, then create the 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 iso8859-8, so shared objects have to map between intermediate code and corresponding iso8859-8 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, if a sample intermediate code in the `en_US.UTF-8` locale that corresponds to a Hebrew character (produced by the PLS layer) is 0xe50000e9, you need to consider the following. 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 non-CTL locale and has the value for the keyword is PLSOutput, it is ignored and the mp(1) generates wide-character codes instead.

You can use the following optional keyword/value pairs 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	Default is ORIENTATION_LTR
Numerals	NUMERALS_NOMINAL/ NUMERALS_NATIONAL/ NUMERALS_CONTEXTUAL	Default is NUMERALS_NOMINAL
TextShaping	TEXT_SHAPED/ TEXT_NOMINAL/ TEXT_SHFORM1/ TEXT_SHFORM2/ TEXT_SHFORM3/ TEXT_SHFORM4	Default is TEXT_SHAPED

EXAMPLE 7–1 Adding a Printer-resident Font

The following example illustrates the steps that you need to follow when you add a new PCF, TrueType, or Type1 printer-resident font to the configuration file.

Replace the font for displaying characters in the range 0x00000021 - 0x0000007f with a TrueType font instead of the currently configured PCF font.

Before adding a new font, look at various components in the configuration file that correspond to the currently configured font, as shown next.

```
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
```

Suppose you selected `/usr/openwin/lib/locale/ja/X11/fonts/TT/HG-MinchoL.ttf` as your candidate for doing the mapping in the `en_US.UTF-8` locale. Because this is a Unicode character-mapped TrueType font file, in the mapping function within the `.so` module you only need to have a function that directly returns the incoming ucs-2 code points.

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

Save this in a `ttfjis0201.c` file. Create a shared object as follows.

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

But if you are mapping a PCF file, such as `/usr/openwin/lib/locale/ja/X11/fonts/75dpi/gotmrk20.pcf.Z`, then look in the `fonts.dir` file in the `/usr/openwin/lib/locale/ja/X11/fonts/75dpi/` directory. Become familiar with the encoding, corresponding to XLFD, which is:

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

If `jisx0201` is the encoding, prepare a shared object that maps from ucs-2 to jisx0201. You need to obtain the mapping table for creating the `.so` module (if one is not already provided). For a Unicode locale, find the mappings from the many charsets to Unicode under `ftp.unicode.org/pub/MAPPINGS/`. Follow these mappings(1)(1) in order 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);
                 }
```

When you create a mapping file, include all the UCS-2 to jisx0201 cases.

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

EXAMPLE 7–2 Creating a shared object file

This example creates a shared object file.

Add this font by adding the following lines to the corresponding sections of `mp.conf`. The following example shows how to add the TrueType font. The PCF font follows the same pattern except that you change the keyword to PCF instead of TrueType.

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

# this line needs to be deleted from `mp.conf` before adding.

```
MapCode2Font  0x0020  0x007f  jis0201  CnvCode2Font jis0201R _ttfjis0201 <.so path>
```

where the `.so` path points to the `xu2jis0201.so` file.

Invoking `mp(1)` with the changed `mp.conf` file causes the range 0x0020-0x007f to be printed in the new font. Map the other Japanese character ranges too with the same `.so` file, for example, the range 0x0000FF61 0x0000FF9F.

To maintain backward compatibility, the `/usr/openwin/lib/locale/$LANG/print/prolog.ps` file, if it exists, is used to create output in the current locale, where `$LANG` is one of the locale components. In that situation, no configuration file mechanism is used.

Refer to `/usr/lib/lp/locale/en_US.UTF-8/mp/mp.conf`, which is a sample `mp.conf` file.

Local Dependant `prolog` Files

What Is `prolog.ps`

The purpose of the `prolog.ps` file is to set up non-generic fonts. Applications use these pre-defined 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

These fonts need to be able to print the local character set in the following example usage of those fonts:

```
100 100 moveto
/LC_Times-Roman findfont 24 scale font setfont
(Any text string in your locale) show
```

The `prolog.ps` *File*

For example, the localization kit provides a sample `prolog.ps` for the Japanese environment. Alternatively, this file is found in the `/usr/openwin/lib/locale/ja/print/` directory.

To add or change Composite fonts in an existing `prolog.ps`, see the following example.

```
%
(Foo-Fine) makecodeset12
(Base-Font) makeEUCfont
%
```

For example, the following defines a composite font called LC_Base-Font:

LC_Base-Font is a composite font of Foo-Fine and a base font called Base-Font. Foo-Fine is a font that contains the local character set. You do not need any in-depth PostScript knowledge to add or change a font.

To Create a `prolog.ps` File, the best way is to study the example version. In the example `prolog.ps`, two routines need to be written, `makecodeset12` and `makeEUCfont`. `Makecodeset12` sets up local font encoding information. This routine might differ from locale to locale. `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`.

`prolog.ps` file support is kept for backward compatibility only. Do not create a new `prolog.ps` file for the printing needs of a locale. Use `mp.conf` instead.

To find `prolog.ps`, the path is

```
/usr/openwin/lib/locale/$LANG/print/prolog.ps
```

Customizing Existing `prolog` Files and Adding New `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 are in three main categories.

- Common prolog file
- Print layout prolog files

Common PostScript prolog Files

The common prolog file is:

- mp.common.ps

This file is the commonprolog file which every other page layout prologue file needs to include.

This file, which resides in the /usr/lib/lp/locale/C/mp/ directory, contains a PostScript routine to re-encode a font from the Standard Encoding to ISOLatin1 Encoding. The .reencodeISOroutine is called from the print layout prolog files to change encoding of the fonts. Usually this prolog file does not need any customization. If the users are creating their own prolog files, 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 giving 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.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, 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`(1) 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.

Following are the functions implemented in print layout `prolog` files. All 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 only 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 routine.
endcol	usage : page_number col_number endcol

Display header and footer information. Move to the new print position, and so forth.

For adding 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 File Customization

These files are located, by default, at `/usr/lib/lp/locale/C/mp/`. An `.xpr` file corresponds to each PostScript `prolog` layout file, except for `mp.common.ps.` 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 in the following description. Three main sections for each `.xpr` file are bound by the following keyword pairs:

- STARTCOMMON/ENDCOMMON
- STARTPAGE/ENDPAGE
- STARTCOLUMN/ENDCOLUMN

Certain keyword/value pairs can be used in these three areas. Each area is described next.

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 "/".

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-integer unsigned-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 string 1' to 'font string n' are X Logical Font Descriptions. The Token which separates the keyword EXTRAHDNGFONT from the comma separated font name list is '"', not spaces or tabs. These fonts are given preference over the built-in fonts when printing of the heading occurs. Usually, EXTRABODYFONT is used to assign printer-resident fonts that are configured in `/usr/openwin/server/etc/XpConfig/C/print/` `models/<model name>/fonts` directory. The `fonts.dir` contains the XLFD of the printer-resident fonts. Usually a font is specified as "-monotype-Gill Sans-Regular-r-normal- -*-%d-*-*-p-0-iso8859-2" in the .xpr file. "%d", if present, is replaced by the `mp(1)` to the point size of the current heading fonts in the `.xpr` file. The x resolution and y resolution are specified "*" and the average width field is set as "0" to indicate selection of scalable font, if possible. You can give more specific font names also.

EXTRABODYFONT *font string 1, font string 2, ... font string n*	This is the same as EXTRAHDNGFONT, except that these fonts are used to print the page body.
XDISPLACEMENT *signed/unsigned int*	Gives 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*	This parameter is the same as x displacement except that the shifting happens in the y direction. These two keywords are useful when you find that some printers have nonstandard margin widths and you need to shift the printed contents in a page.

STARTPAGE/ENDPAGE

The keyword value pairs in this section are bound by STARTPAGE and ENDPAGE keywords. This section contains drawings 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. 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 /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 ints 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.

STARTCOLUMN/ENDCOLUMN

All keywords are the same as the previous STARTPAGE/ENDPAGE section except that the entries in this section are applied to 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 are three times per page.

Creating a New `.xpr` File

The following are the `mp`(1) program defaults for different keyword values if these values are not specified in the `.xpr` file for the STARTCOMMON/ENDCOMMON section.

- ORIENTATION 0
- PAGELENGTH 60
- LINELENGTH 80
- YTEXTBOUNDARY 3005
- NUMCOLS 01
- HDNGFONTSIZE 120
- BODYFONTSIZE 90
- PROLOGDPI 300
- STARTTEXT 135 280
- PAGESTRING 0

No default values are needed for the other two sections bound by STARTPAGE/ENDPAGE and STARTCOLUMN/ENDCOLUMN.

When you create a new `.xpr prolog` file, you need to specify only the values that differ from the default.

To create a page with no decoration, use four logical pages per physical page, in portrait format.

- STARTCOMMON
- NUMCOLS 04
- LINELENGTH 20
- ENDCOMMON

In this situation, you do not need the other 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 default unless you are not specifying the PROLOGDPI keyword. If target printer resolution is different, the `.xpr` file is scaled to fit into that resolution by the program.

When you create a `.xpr` file, you must know the paper dimensions beforehand. 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, they put some margin around the physical paper. That means that even if you try to print from 0,0 the printing won't be in the top left corner of the page. You need to consider this limitation when you create a new `.xpr` file.

APPENDIX **A**

`iconv` Code Conversions

The following table lists the available Unicode-related code conversion modules.

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment

From Code (Symbol)	To Code (Symbol)
646 (ISO 646)	UTF-8
646 (ISO 646)	UCS-2
646 (ISO 646)	USC-2BE
646 (ISO 646)	UCS-2LE
646 (ISO 646)	USC-4
646 (ISO 646)	USC-4BE
646 (ISO 646)	USC-4LE
646 (ISO 646)	UTF-16
646 (ISO 646)	UTF-16BE
646 (ISO 646)	UTF-16LE
646 (ISO 646)	UTF-8
8859–11	UTF-8
8859-1 (ISO8859-1)	UCS-2
8859-1 (ISO8859-1)	UCS-2BE
8859-1 (ISO8859-1)	UCS-2LE
8859-1 (ISO8859-1)	UCS-4
8859-1 (ISO8859-1)	UCS4-BE

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
8859-1 (ISO8859-1)	UCS-4LE
8859-1 (ISO8859-1)	UTF-16
8859-1 (ISO8859-1)	UTF-16BE
8859-1 (ISO8859-1)	UTF-16LE
8859-1 (ISO8859-1)	UTF-8
8859-10 (ISO8859-10)	UCS-2
8859-10 (ISO8859-10)	UCS-2BE
8859-10 (ISO8859-10)	UCS-2LE
8859-10 (ISO8859-10)	UCS-4
8859-10 (ISO8859-10)	UCS4-BE
8859-10 (ISO8859-10)	UCS-4LE
8859-10 (ISO8859-10)	UTF-16
8859-10 (ISO8859-10)	UTF-16BE
8859-10 (ISO8859-10)	UTF-16LE
8859-10 (ISO8859-10)	UTF-8
8859-13 (ISO8859-13)	UCS-2
8859-13 (ISO8859-13)	UCS-2BE
8859-13 (ISO8859-13)	UCS-2LE
8859-13 (ISO8859-13)	UCS-4
8859-13 (ISO8859-13)	UCS4-BE
8859-13 (ISO8859-13)	UCS-4LE
8859-13 (ISO8859-13)	UTF-16
8859-13 (ISO8859-13)	UTF-16BE
8859-13 (ISO8859-13)	UTF-16LE
8859-13 (ISO8859-13)	UTF-8
8859-14 (ISO8859-14)	UCS-2
8859-14 (ISO8859-14)	UCS-2BE
8859-14 (ISO8859-14)	UCS-2LE

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
8859-14 (ISO8859-14)	UCS-4
8859-14 (ISO8859-14)	UCS4-BE
8859-14 (ISO8859-14)	UCS-4LE
8859-14 (ISO8859-14)	UTF-16
8859-14 (ISO8859-14)	UTF-16BE
8859-14 (ISO8859-14)	UTF-16LE
8859-14 (ISO8859-14)	UTF-8
8859-15 (ISO8859-15)	UCS-2
8859-15 (ISO8859-15)	UCS-2BE
8859-15 (ISO8859-15)	UCS-2LE
8859-15 (ISO8859-15)	UCS-4
8859-15 (ISO8859-15)	UCS4-BE
8859-15 (ISO8859-15)	UCS-4LE
8859-15 (ISO8859-15)	UTF-16
8859-15 (ISO8859-15)	UTF-16BE
8859-15 (ISO8859-15)	UTF-16LE
8859-15 (ISO8859-15)	UTF-8
8859-2 (ISO8859-2)	UCS-2
8859-2 (ISO8859-2)	UCS-2BE
8859-2 (ISO8859-2)	UCS-2LE
8859-2 (ISO8859-2)	UCS-4
8859-2 (ISO8859-2)	UCS4-BE
8859-2 (ISO8859-2)	UCS-4LE
8859-2 (ISO8859-2)	UTF-16
8859-2 (ISO8859-2)	UTF-16BE
8859-2 (ISO8859-2)	UTF-16LE
8859-2 (ISO8859-2)	UTF-8
8859-3 (ISO8859-3)	UCS-2

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
8859-3 (ISO8859-3)	UCS-2BE
8859-3 (ISO8859-3)	UCS-2LE
8859-3 (ISO8859-3)	UCS-4
8859-3 (ISO8859-3)	UCS4-BE
8859-3 (ISO8859-3)	UCS-4LE
8859-3 (ISO8859-3)	UTF-16
8859-3 (ISO8859-3)	UTF-16BE
8859-3 (ISO8859-3)	UTF-16LE
8859-3 (ISO8859-3)	UTF-8
8859-4 (ISO8859-4)	UCS-2
8859-4 (ISO8859-4)	UCS-2BE
8859-4 (ISO8859-4)	UCS-2LE
8859-4 (ISO8859-4)	UCS-4
8859-4 (ISO8859-4)	UCS4-BE
8859-4 (ISO8859-4)	UCS-4LE
8859-4 (ISO8859-4)	UTF-16
8859-4 (ISO8859-4)	UTF-16BE
8859-4 (ISO8859-4)	UTF-16LE
8859-4 (ISO8859-4)	UTF-8
8859-5 (ISO8859-5)	UCS-2
8859-5 (ISO8859-5)	UCS-2BE
8859-5 (ISO8859-5)	UCS-2LE
8859-5 (ISO8859-5)	UCS-4
8859-5 (ISO8859-5)	UCS4-BE
8859-5 (ISO8859-5)	UCS-4LE
8859-5 (ISO8859-5)	UTF-16
8859-5 (ISO8859-5)	UTF-16BE
8859-5 (ISO8859-5)	UTF-16LE

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
8859-5 (ISO8859-5)	UTF-8
8859-6 (ISO8859-6)	UCS-2
8859-6 (ISO8859-6)	UCS-2BE
8859-6 (ISO8859-6)	UCS-2LE
8859-6 (ISO8859-6)	UCS-4
8859-6 (ISO8859-6)	UCS4-BE
8859-6 (ISO8859-6)	UCS-4LE
8859-6 (ISO8859-6)	UTF-16
8859-6 (ISO8859-6)	UTF-16BE
8859-6 (ISO8859-6)	UTF-16LE
8859-6 (ISO8859-6)	UTF-8
8859-7 (ISO8859-7)	UCS-2
8859-7 (ISO8859-7)	UCS-2BE
8859-7 (ISO8859-7)	UCS-2LE
8859-7 (ISO8859-7)	UCS-4
8859-7 (ISO8859-7)	UCS4-BE
8859-7 (ISO8859-7)	UCS-4LE
8859-7 (ISO8859-7)	UTF-16
8859-7 (ISO8859-7)	UTF-16BE
8859-7 (ISO8859-7)	UTF-16LE
8859-7 (ISO8859-7)	UTF-8
8859-8 (ISO8859-8)	UCS-2
8859-8 (ISO8859-8)	UCS-2BE
8859-8 (ISO8859-8)	UCS-2LE
8859-8 (ISO8859-8)	UCS-4
8859-8 (ISO8859-8)	UCS4-BE
8859-8 (ISO8859-8)	UCS-4LE
8859-8 (ISO8859-8)	UTF-16

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
8859-8 (ISO8859-8)	UTF-16BE
8859-8 (ISO8859-8)	UTF-16LE
8859-8 (ISO8859-8)	UTF-8
8859-9 (ISO8859-9)	UCS-2
8859-9 (ISO8859-9)	UCS-2BE
8859-9 (ISO8859-9)	UCS-2LE
8859-9 (ISO8859-9)	UCS-4
8859-9 (ISO8859-9)	UCS4-BE
8859-9 (ISO8859-9)	UCS-4LE
8859-9 (ISO8859-9)	UTF-16
8859-9 (ISO8859-9)	UTF-16BE
8859-9 (ISO8859-9)	UTF-16LE
8859-9 (ISO8859-9)	UTF-8
eucJP	UTF-8
gb2312	UTF-8
iso2022	UTF-8
ko_KR-cp933	UTF-8
ko_KR-euc	UTF-8
ko_KR-iso2022–7	UTF-8
ko_KR-johap	UTF-8
ko_KR-johap92	UTF-8
zh_TW-euc	UTF-8
zh_TW-cp937	UTF-8
zh_TW-iso2022–7	UTF-8
GBK	UTF-8
FujitsuJEF-ascii	UTF-8
FujitsuJEF-ascii-face	UTF-8
FujitsuJEF-Kana	UTF-8

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
FujitsuJEF-Kana-face	UTF-8
HitachiKEIS78	UTF-8
HitachiKEIS83	UTF-8
ISO-2022–JP	UTF-8
KOI8-R	UCS-2
KOI8-R	UCS-2BE
KOI8-R	UCS-2LE
KOI8-R	UCS-4
KOI8-R	UCS4-BE
KOI8-R	UCS-4LE
KOI8-R	UTF-8
KOI8-R	UTF-16
KOI8-R	UTF-16BE
KOI8-R	UTF-16LE
KOI8-R	UTF-8
KOI8-U	UCS-2
KOI8-U	UCS-2BE
KOI8-U	UCS-2LE
KOI8-U	UCS-4
KOI8-U	UCS-4BE
KOI8-U	UCS-4LE
KOI8-U	UTF-16
KOI8-U	UTF-16BE
KOI8-U	UTF-16LE
KOI8-U	UTF–8
NECJIPS	UTF-8
NECJIPSE	UTF-8
PCK	UTF-8

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UCS-2	646 (ISO 646)
UCS-2	8859-1 (ISO8859-1)
UCS-2	8859-10 (ISO8859-10)
UCS-2	8859-13 (ISO8859-13)
UCS-2	8859-14 (ISO8859-14)
UCS-2	8859-15 (ISO8859-15)
UCS-2	8859-2 (ISO8859-2)
UCS-2	8859-3 (ISO8859-3)
UCS-2	8859-4 (ISO8859-4)
UCS-2	8859-5 (ISO8859-5)
UCS-2	8859-6 (ISO8859-6)
UCS-2	8859-7 (ISO8859-7)
UCS-2	8859-8 (ISO8859-8)
UCS-2	8859-9 (ISO8859-9)
UCS-2	KOI8-R
UCS-2	KOI8-U
UCS-2	UCS-4
UCS-2	UCS-4BE
UCS-2	UCS-4LE
UCS-2	UTF-7
UCS-2	UTF-8
UCS-2BE	646 (ISO 646)
UCS-2BE	8859-1 (ISO8859-1)
UCS-2BE	8859-10 (ISO8859-10)
UCS-2BE	8859-13 (ISO8859-13)
UCS-2BE	8859-14 (ISO8859-14)
UCS-2BE	8859-15 (ISO8859-15)
UCS-2BE	8859-2 (ISO8859-2)

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UCS-2BE	8859-3 (ISO8859-3)
UCS-2BE	8859-4 (ISO8859-4)
UCS-2BE	8859-5 (ISO8859-5)
UCS-2BE	8859-6 (ISO8859-6)
UCS-2BE	8859-7 (ISO8859-7)
UCS-2BE	8859-8 (ISO8859-8)
UCS-2BE	8859-9 (ISO8859-9)
UCS-2BE	KOI8-R
UCS-2BE	KOI8-U
UCS-2BE	UCS-4
UCS-2BE	UCS-4BE
UCS-2BE	UCS-4LE
UCS-2BE	UTF-8
UCS-4	UTF-8
UCS-4LE	646 (ISO 646)
UCS-4LE	8859-1 (ISO8859-1)
UCS-4LE	8859 -10 (ISO8859-10)
UCS-4LE	8859-13 (ISO8859-13)
UCS-4LE	8859-14 (ISO8859-14)
UCS-4LE	8859-15 (ISO8859-15)
UCS-4LE	8859-2 (ISO8859-2)
UCS-4LE	8859-3 (ISO8859-3)
UCS-4LE	8859-4 (ISO8859-4)
UCS-4LE	8859-5 (ISO8859-5)
UCS-4LE	8859-6 (ISO8859-6)
UCS-4LE	8859-7 (ISO8859-7)
UCS-4LE	8859-8 (SO 8859-8)
UCS-4LE	8859-9 (ISO8859-9)

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UCS-4LE	KOI8-R
UCS-4LE	KOI8-U
UCS-4LE	UCS-2
UCS-4LE	UCS-2BE
UCS-4LE	UCS-2LE
UCS-4LE	UTF-16
UCS-4LE	UTF-16BE
UCS-4LE	UTF-16LE
UCS-4LE	UTF-8
UTF-7	UTF-8
UTF-8	646
UTF-8	8859–1
UTF-8	8859–2
UTF-8	8859–3
UTF-8	8859–4
UTF-8	8859–5
UTF-8	8859–6
UTF-8	8859–7
UTF-8	8859–8
UTF-8	8859–9
UTF-8	8859–10
UTF-8	8859–11
UTF-8	8859–15
UTF-8	eucJP
UTF-8	gb2312
UTF-8	iso2022
UTF-8	ko_KR-euc
UTF-8	ko_KR-johap

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-8	ko_KR-johap92
UTF-8	ko_KR-iso2022–7
UTF-8	zh_TW-euc
UTF-8	zh_TW-iso2022–7
UTF-8	zh_TW-cp937
UTF-8	FujitsuJEF-ascii
UTF-8	FujitsuJEF-ascii-face
UTF-8	FujitsuJEF-kana
UTF-8	FujitsuJEF-kana-face
UTF-8	GBK
UTF-8	HitachiKEIS78
UTF-8	HitachiKEIS83
UTF-8	ISO-2022–JP
UTF-8	KOI8–R
UTF-8	NECJIPS
UTF-8	NECJIPSE
UTF-8	PCK
UTF-8	UCS-2
UTF-8	UCS-4
UTF-8	UTF-7
UTF-8	UTF-16
UTF-16	646 (ISO 646)
UTF-16	8859-1 (ISO8859-1)
UTF-16	8859-10 (ISO8859-10)
UTF-16	8859-13 (ISO8859-13)
UTF-16	8859-14 (ISO8859-14)
UTF-16	8859-15 (ISO8859-15)
UTF-16	8859-2 (ISO8859-2)

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-16	8859-3 (ISO8859-3)
UTF-16	8859-4 (ISO8859-4)
UTF-16	8859-5 (ISO8859-5)
UTF-16	8859-6 (ISO8859-6)
UTF-16	8859-7 (ISO8859-7)
UTF-16	8859-8 (ISO8859-8)
UTF-16	8859-9 (ISO8859-9)
UTF-16	KOI8-R
UTF-16	KOI8-U
UTF-16	UCS-4
UTF-16	UCS-4BE
UTF-16	UCS-4LE
UTF-16	UTF-8
UTF-16BE	646 (ISO 646)
UTF-16BE	8859-1 (ISO8859-1)
UTF-16BE	8859-10(ISO8859-10)
UTF-16BE	8859-13 (ISO8859-13)
UTF-16BE	8859-14 (ISO8859-14)
UTF-16BE	8859-15 (ISO8859-15)
UTF-16BE	8859-2 (ISO8859-2)
UTF-16BE	8859-3 (ISO8859-3)
UTF-16BE	8859-4 (ISO8859-4)
UTF-16BE	8859-5 (ISO8859-5)
UTF-16BE	8859-6 (ISO8859-6)
UTF-16BE	8859-7 (ISO8859-7)
UTF-16BE	8859-8 (ISO8859-8)
UTF-16BE	8859-9 (ISO8859-9)
UTF-16BE	KOI8-R

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-16BE	KOI8-U
UTF-16BE	UCS-4
UTF-16BE	UCS-4BE
UTF-16BE	UCS-4LE
UTF-16BE	UTF-8
UTF-16LE	646 (ISO 646)
UTF-16LE	8859-1 (ISO8859-1)
UTF-16LE	8859-10 (ISO8859-10)
UTF-16LE	8859-13 (ISO8859-13)
UTF-16LE	8859-14 (ISO8859-14)
UTF-16LE	8859-15 (ISO8859-15)
UTF-16LE	8859-2 (ISO8859-2)
UTF-16LE	8859-3 (ISO8859-3)
UTF-16LE	8859-4 (ISO8859-4)
UTF-16LE	8859-5 (ISO8859-5)
UTF-16LE	8859-6 (ISO8859-6)
UTF-16LE	8859-7 (ISO8859-7)
UTF-16LE	8859 -8 (ISO8859-8)
UTF-16LE	8859-9 (ISO8859-9)
UTF-16LE	KOI8-R
UTF-16LE	KOI8-U
UTF-16LE	UCS-4
UTF-16LE	UCS-4BE
UTF-16LE	UCS-4LE
UTF-16LE	UTF-8
UTF-7	UCS-2
UTF-7	UCS-4
UTF-7	UCS-8

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-8	646 (ISO 646)
UTF-8	8859-1 (ISO8859-1)
UTF-8	8859-10 (ISO8859-10)
UTF-8	8859-13 (ISO8859-13)
UTF-8	8859-14 (ISO8859-14)
UTF-8	8859-15 (ISO8859-15)
UTF-8	8859-2 (ISO8859-2)
UTF-8	8859-3 (ISO8859-3)
UTF-8	8859-4 (ISO8859-4)
UTF-8	8859-5 (ISO8859-5)
UTF-8	8859-6 (ISO8859-6)
UTF-8	8859-7 (ISO8859-7)
UTF-8	8859-8 (ISO8859-8)
UTF-8	8859-9 (ISO8859-9)
UTF-8	KOI8-R
UTF-8	KOI8-U
UTF-8	UCS-2
UTF-8	UCS-2BE
UTF-8	UCS-2LE
UTF-8	UCS-4
UTF-8	UCS-4BE
UTF-8	UCS-4LE
UTF-8	UTF-16
UTF-8	UTF-16BE
UTF-8	UCS-16LE
UTF-8	UTF-7
UTF-32	UTF-8
UTF-32	UCS-2

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-32	UCS-2BE
UTF-32	UCS-2LE
UTF-32	UCS-4
UTF-32	UCS-4BE
UTF-32	UCS-4LE
UTF-32	UTF-16
UTF-32	UTF32–BE
UTF-32	UTF-16LE
UTF-32	US-ASCII
UTF-32	ISO8859–1
UTF-32	ISO8859–2
UTF-32	ISO8859–3
UTF-32	ISO8859–4
UTF-32	ISO8859–5
UTF-32	ISO8859–6
UTF-32	ISO8859–7
UTF-32	ISO8859–8
UTF-32	ISO8859–9
UTF-32	ISO8859–10
UTF-32	ISO8859—13
UTF-32	ISO8859–14
UTF-32	ISO8859–15
UTF-32	ISO8859–16
UTF-32	KOI8–R
UTF-32	KOI8–U
UTF-32BE	UTF-8
UTF-32BE	UCS-2
UTF-32BE	UCS-2BE

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-32BE	UCS-2LE
UTF-32BE	UCS-4
UTF-32BE	UCS-4BE
UTF-32BE	UCS-4LE
UTF-32BE	UTF-16
UTF32–BE	UTF-16BE
UTF-32 BE	UTF-16LE
UTF-32BE	US-ASCII
UTF-32BE	ISO8859–1
UTF-32BE	ISO8859–2
UTF-32BE	ISO8859–3
UTF-32BE	ISO8859–4
UTF-32BE	ISO8859–5
UTF-32BE	ISO8859–6
UTF-32BE	ISO8859–7
UTF-32BE	ISO8859–8
UTF-32BE	ISO8859–9
UTF-32BE	ISO8859–10
UTF-32BE	ISO8859–13
UTF-32BE	ISO8859–14
UTF-32BE	ISO8859–15
UTF-32BE	ISO8859–16
UTF-32BE	KOI8–R
UTF-32BE	KOI8–U
UTF-32LE	UTF-8
UTF-32LE	UCS-2
UTF-32LE	UCS-2BE
UTF-32LE	UCS-2LE

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-32LE	UCS-4
UTF-32LE	UCS-4BE
UTF-32LE	UCS-4LE
UTF32–LE	UTF-16
UTF32–LE	UTF-16BE
UTF-32LE	UTF-16LE
UTF-32LE	US-ASCII
UTF-32LE	ISO8859–1
UTF-32LE	ISO8859–2
UTF-32LE	ISO8859–3
UTF-32LE	ISO8859–4
UTF-32LE	ISO8859–5
UTF-32LE	ISO8859–6
UTF-32LE	ISO8859–7
UTF-32LE	ISO8859–8
UTF-32LE	ISO8859–9
UTF-32LE	ISO8859–10
UTF-32LE	ISO8859–13
UTF-32LE	ISO8859–14
UTF-32LE	ISO8859–15
UTF-32LE	ISO8859–16
UTF-32LE	KOI8–R
UTF-32LE	KOI8–U
ISO8859–16	UTF-8
ISO8859–16	UCS-2
ISO8859–16	UCS-2BE
ISO8859–16	UCS-2LE
ISO8859–16	UCS-4

TABLE A–1 Available Unicode Related `iconv` Code Conversion Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
ISO8859–16	UCS-4BE
ISO8859–16	UCS-4LE
ISO8859–16	UTF-16
ISO8859–16	UTF-16BE
ISO8859–16	UTF-16LE
ISO8859–16	UTF-32
ISO8859–16	UTF-32BE
ISO8859–16	UTF-32LE

Note – UTF-EBCDIC is a new IBM codepage name. Starting with the Solaris 9 environment, Sun also supports bidirectional UTF-8 <—> UTF-EBCDIC conversion.

The following table lists the available Unicode and IBM/Microsoft EBCDIC and PC `iconv` code conversion modules.

TABLE A–2 Available Unicode and IBM/Microsoft EBCDIC and PC Code Page Related `iconv` Code Conversions Modules in the Solaris 9 Environment

From Code (Symbol)	To Code (Symbol)
UTF-8	IBM-037
UTF-8	IBM-273
UTF-8	IBM-277
UTF-8	IBM-278
UTF-8	IBM-280
UTF-8	IBM-284
UTF-8	IBM-285
UTF-8	IBM-297
UTF-8	IBM-420
UTF-8	IBM-424
UTF-8	IBM-500

TABLE A–2 Available Unicode and IBM/Microsoft EBCDIC and PC Code Page Related `iconv` Code Conversions Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-8	IBM-850
UTF-8	IBM-852
UTF-8	IBM-855
UTF-8	IBM-856
UTF-8	IBM-857
UTF-8	IBM-862
UTF-8	IBM-864
UTF-8	IBM-866
UTF-8	IBM-869
UTF-8	IBM-870
UTF-8	IBM-871
UTF-8	IBM-875
UTF-8	IBM-880
UTF-8	IBM-1025
UTF-8	IBM-1026
UTF-8	IBM-1112
UTF-8	IBM-1122
UTF-8	IBM-921
UTF-8	IBM-922
UTF-8	IBM-1046
UTF-8	IBM-1140
UTF-8	IBM-1141
UTF-8	IBM-1142
UTF-8	IBM-1143
UTF-8	IBM-1144
UTF-8	IBM-1145
UTF-8	IBM-1146
UTF-8	IBM-1147

TABLE A–2 Available Unicode and IBM/Microsoft EBCDIC and PC Code Page Related `iconv` Code Conversions Modules in the Solaris 9 Environment *(Continued)*

From Code (Symbol)	To Code (Symbol)
UTF-8	IBM-1148
UTF-8	IBM-1149
UTF-8	CP850
UTF-8	CP852
UTF-8	CP855
UTF-8	CP857
UTF-8	CP862
UTF-8	CP864
UTF-8	CP866
UTF-8	CP869
UTF-8	CP874
UTF-8	CP1250
UTF-8	CP1251
UTF-8	CP1252
UTF-8	CP1253
UTF-8	CP1254
UTF-8	CP1255
UTF-8	CP1256
UTF-8	CP1257
UTF-8	CP1258

The following table lists the available `iconv` code conversions IBM and Microsoft EBCDIC/PC code pages to UTF-8.

TABLE A–3 Available `iconv` Code Conversions - IBM and Microsoft EBCDIC/PC Code Pages to UTF-8

UTF-EBCDIC	UTF-8
IBM-037	UTF-8
IBM-273	UTF-8
IBM-277	UTF-8

TABLE A–3 Available `iconv` Code Conversions - IBM and Microsoft EBCDIC/PC Code Pages to UTF-8 *(Continued)*

UTF-EBCDIC	UTF-8
IBM-278	UTF-8
IBM-280	UTF-8
IBM-284	UTF-8
IBM-285	UTF-8
IBM-297	UTF-8
IBM-420	UTF-8
IBM-424	UTF-8
IBM-500	UTF-8
IBM-850	UTF-8
IBM-852	UTF-8
IBM-855	UTF-8
IBM-856	UTF-8
IBM-857	UTF-8
IBM-862	UTF-8
IBM-864	UTF-8
IBM-866	UTF-8
IBM-869	UTF-8
IBM-870	UTF-8
IBM-871	UTF-8
IBM-875	UTF-8
IBM-880	UTF-8
IBM-921	UTF-8
IBM-922	UTF-8
IBM-1025	UTF-8
IBM-1026	UTF-8
IBM-1046	UTF-8
IBM-1112	UTF-8
IBM-1122	UTF-8

TABLE A–3 Available `iconv` Code Conversions - IBM and Microsoft EBCDIC/PC Code Pages to UTF-8 *(Continued)*

UTF-EBCDIC	UTF-8
IBM-1140	UTF-8
IBM-1141	UTF-8
IBM-1142	UTF-8
IBM-1143	UTF-8
IBM-1144	UTF-8
IBM-1145	UTF-8
IBM-1146	UTF-8
IBM-1147	UTF-8
IBM-1148	UTF-8
IBM-1149	UTF-8
CP850	UTF-8
CP852	UTF-8
CP855	UTF-8
CP857	UTF-8
CP862	UTF-8
CP864	UTF-8
CP866	UTF-8
CP869	UTF-8
CP874	UTF-8
CP1250	UTF-8
CP1251	UTF-8
CP1252	UTF-8
CP1253	UTF-8
CP1254	UTF-8
CP1255	UTF-8
CP1256	UTF-8
CP1257	UTF-8
CP1258	UTF-8

APPENDIX **B**

Partial Localization Package Names on OS CD

The following table lists the packages and contents of the OS CD.

TABLE B–1 List of Partial Locales

Package Name	Description
SUNWauaox	Australasia 64-bit OS Support.
SUNWauadt	Australasia CDE Support.
SUNWauaos	Australasia OS Support.
SUNWauaow	Australasia OW Support.
SUNWcamox	Central America 64-bit OS Support.
SUNWcamdt	Central America CDE Support.
SUNWcamos	Central America OS Support.
SUNWcamow	Central America OW Support.
SUNWceuox	Central Europe 64-bit OS Support.
SUNWceudt	Central Europe CDE Support.
SUNWceuos	Central Europe OS Support.
SUNWceuow	Central Europe OW Support.
SUNWalex	Common files shared by Chinese, Japanese and Korean locales. This package is required to run Asian Language Environment (64-bit).
SUNWeeuox	Eastern Europe 64-bit OS Support.
SUNWeeudt	Eastern Europe CDE Support.
SUNWeeuos	Eastern Europe OS Support.

TABLE B–1 List of Partial Locales *(Continued)*

Package Name	Description
SUNWeeuow	Eastern Europe OW Support.
SUNWfris	French install software localization.
SUNWdeis	German install software localization.
SUNWindt	Indic localizations for CDE Desktop Login Environment
SUNWinfnt	Indic (UTF-8) X Windows Platform Required Fonts
SUNWinis	Indic package contains Indic language environment specific files. It is a required package for installing Indic language environment, but it is not installed for the user environment
SUNWinleu	Indic package contains Indic language environment specific files. It is a required package for running Indic language environment.
SUNWinlex	Indic package contains Indic language environment specific files. It is a required package for running the Indic language environment (64–bit).
SUNWinplt	Indic X Window System platform software
SUNWinttf	Indic TrueType Fonts
SUNWitis	Italian install software localization.
NSCPjacom	Japanese (common) localization of Netscape Communicator 4.7 supporting International security.
SUNWjedt	Japanese (EUC) Localization for CDE DESKTOP LOGIN ENVIRONMENT.
SUNWjeuc	Japanese (EUC) Feature Package specific files for `usr`; it is a required package to support EUC environment.
SUNWjeucx	Japanese (EUC) Feature Package specific 64-bit files for `usr`; it is a required package to run JFP environment.
SUNWjexpl	Japanese (EUC) Localizations for X Window System platform software.
SUNWjexpx	Japanese (EUC) Localizations for X Window System platform software (64-bit).
SUNWjfpr	Stream modules for Japanese Feature Package (JFP); it is a required package to run JFP environment.
SUNWjfpu	Japanese Feature Package (JFP) specific files for `usr`; it is a required package to run JFP environment.
SUNWjfpux	Japanese Feature Package (JFP) specific 64-bit files for `usr`; it is a required package to run JFP environment.

TABLE B–1 List of Partial Locales *(Continued)*

Package Name	Description
SUNWjman	Japanese Feature Package Man Pages to see English man pages for `SUNWjfpr` and `SUNWjfpu`.
SUNWjpck	Japanese (PCK - PC Kanji Code) Feature Package specific files; it's a required package to support PCK environment.
SUNWjpckx	Japanese (PCK) Feature Package specific 64-bit files for `usr`; it is a required package to run JFP environment.
SUNWjpdt	Japanese (PCK) Localization for CDE DESKTOP LOGIN ENVIRONMENT.
SUNWjpxpl	Japanese (PCK) Localizations for X Window System platform software.
SUNWjpxpx	Japanese (PCK) Localizations for X Window System platform software (64-bit).
SUNWju8	Japanese (UTF-8) Feature Package specific files; it's a required package to support Japanese UTF-8 environment.
SUNWju8x	Japanese (UTF-8) Feature Package specific 64-bit files for `usr`; it is a required package to run JFP environment.
SUNWjudt	Japanese (UTF-8) Localization for CDE DESKTOP LOGIN ENVIRONMENT.
SUNWjuxpl	Japanese (UTF-8) Localizations for X Window System platform software.
SUNWjxcft	Japanese JIS X 0212–1990 TrueType and bitmap fonts.
SUNWkleux	Korean (EUC) Language Environment specific files. It is a required package to run Korean Language Environment (64-bit).
SUNWkulex	Korean (UTF-8) Language Environment specific files. It is a required package to run Korean Language Environment (64-bit).
SUNWkdt	Korean Localizations for CDE Desktop Login Environment.
NSCPcpcom	Simplified Chinese partial version of Netscape Communicator 4.7 supporting International security.
SUNWsamox	Southern America 64-bit OS Support.
SUNWsamdt	Southern America CDE Support.
SUNWsamos	Southern America OS Support.
SUNWsamow	Southern America OW Support.
SUNWseuox	Southern Europe 64-bit OS Support.
SUNWseudt	Southern Europe CDE Support.

TABLE B–1 List of Partial Locales *(Continued)*

Package Name	Description
SUNWseuos	Southern Europe OS Support.
SUNWseuow	Southern Europe OW Support.
SUNWfrspl	Spell Checking Engine - French Dictionary.
SUNWdespl	Spell Checking Engine - German Dictionary.
SUNWitspl	Spell Checking Engine - Italian Dictionary.
SUNWesspl	Spell Checking Engine - Spanish Dictionary.
SUNWsvspl	Spell Checking Engine - Swedish Dictionary.
SUNWjfpr	Stream modules for Japanese Feature Package (JFP), it is a required package to run JFP environment.
SUNWsvis	Swedish install software localization.
SUNWkleu	This package contains Korean Language Environment specific files. It is a required package to run Korean Language Environment.
SUNWkuleu	This package contains Korean UTF-8 Language Environment specific files. It is a required package to run Korean Language Environment.
SUNWcleu	This package contains Simplified Chinese (EUC) Language Environment specific files. It is a required package to run Simplified Chinese (EUC) Language Environment.
SUNWgleu	This package contains Simplified Chinese (EUC) LLanguage Environment specific files. It is a required package to run Simplified Chinese (GBK) Language Environment.
SUNWculeu	This package contains Simplified Chinese (UTF-8) Language Environment specific files. It is a required package to run Simplified Chinese (UTF-8) Language Environment.
SUNWtleu	This package contains Thai Language Environment specific files. It is a required package to run Thai Language Environment.
SUNWhuleu	This package contains Traditional Chinese (UTF-8) Language Environment specific files. It is a required package to run Traditional Chinese UTF-8 Language Environment.
SUNW5leu	This package contains Traditional Chinese Language Environment specific files. It is a required package to run Traditional Chinese BIG5 Language Environment.
SUNWhleu	This package contains Traditional Chinese Language Environment specific files. It is a required package to run Traditional Chinese Language Environment.

TABLE B–1 List of Partial Locales *(Continued)*

Package Name	Description
SUNWale	This package contains common files shared by Chinese, Japanese and Korean locales. It is a required package to run Asian Language Environment.
SUNWaled	This package contains man pages shared by Chinese, Japanese, and Korean locales.
SUNW5leux	Traditional Chinese (BIG5) Language Environment user files (64-bit).
SUNW5xplx	Traditional Chinese (BIG5) X Windows Platform Software Package (64-bit).
SUNWhleux	Traditional Chinese (EUC) Language Environment specific files. It is a required package to run Traditional Chinese Language Environment (64-bit).
SUNWhulex	Traditional Chinese (UTF-8) Language Environment user files (64-bit).
SUNW5xplt	Traditional Chinese BIG5 X Windows Platform Software Package.
SUNW5dt	Traditional Chinese Localizations for CDE Desktop Login Environment.
SUNWhdt	Traditional Chinese Localizations for CDE Desktop Login Environment.
SUNWhicd	Traditional Chinese Solaris Install CD localization source files.
SUNW5ttf	Traditional Chinese TrueType Fonts Package.
SUNWhttf	Traditional Chinese TrueType Fonts Package.
SUNWhuplt	Traditional Chinese UTF-8 X Windows Platform Software Package.
SUNWhxplt	Traditional Chinese X Windows Platform Software Package.
SUNWhxfnt	Traditional Chinese X Windows Platform required Fonts Package.
SUNWhkdt	Traditional Chinese (Hong Kong) localization for CDE Desktop Login Environment
SUNWhkfnt	Traditional Chinese BIG5 (Hong Kong) X Windows Platform required Fonts Package
SUNWhkleu	Traditional Chinese (Hong Kong) Language Environment user files
SUNWhklex	Traditional Chinese (Hong Kong BIG5) language Environment user files (64-bit)

TABLE B–1 List of Partial Locales *(Continued)*

Package Name	Description
SUNWhkplt	Traditional Chinese (Hong Kong) BIG5 X Windows Platform Software Package
SUNWhkplx	Traditional Chinese (Hong Kong BIG5) X Windows Platform Software Package (64-bit)
NSCPhpcom	Traditional Chinese partial version of Netscape Communicator 4.7 supporting International security.
SUNWweuox	Western Europe 64-bit OS Support.
SUNWweudt	Western Europe CDE Support.
SUNWweuos	Western Europe OS Support.
SUNWweuow	Western Europe OW Support.
SUNWi1cs	X11 ISO8859-1 Codeset Support.
SUNWi13cs	X11 ISO8859-13 Codeset Support.
SUNWi15cs	X11 ISO8859-15 Codeset Support.
SUNWi2cs	X11 ISO8859-2 Codeset Support.
SUNWi5cs	X11 ISO8859–5 Codeset Support.
SUNWi7cs	X11 ISO8859–7 Codeset Support.
SUNWi9cs	X11 ISO8859–9 Codeset Support.
SUNWi2of	X11 fonts for ISO-8859-2 character set (optional fonts).
SUNWi4of	X11 fonts for ISO-8859-4 character set (optional fonts).
SUNWi5of	X11 fonts for ISO-8859-5 character set (optional fonts).
SUNWi7of	X11 fonts for ISO-8859-7 character set (optional fonts).
SUNWi9of	X11 fonts for ISO-8859-9 character set (optional fonts).
SUNWkoi8f	X11 fonts for KOI8–R character set.

APPENDIX **C**

Languages CD Packages List

The following table lists the Simplified Chinese language packages and their contents.

TABLE C–1 Simplified Chinese

Package Name	Description
NSCPccom	Simplified Chinese Localization of Netscape Communicator 4.7 supporting International security.
NSCPcucom	zh.UTF-8 Localization of Netscape Communicator 4.7 supporting International security.
NSCPgcom	zh.GBK Localization of Netscape Communicator 4.7 supporting International security.
SUNWcacx Simplified Chinese AccessX client program	
SUNWcadis	Simplified Chinese (EUC) Localizations for admintool and GUI install.
SUNWcadma	Simplified Chinese (EUC) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWhadis packages for Simplified Chinese (EUC) Localization.
SUNWcbcp	Simplified Chinese (EUC) Language Environment binary compatibility files.
SUNWcdab	Simplified Chinese (EUC) Localizations for CDE Desktop Application Builder
SUNWcdbas	Simplified Chinese (EUC) Localizations for CDE Base functionality
SUNWcdcl	Simplified Chinese Localizations for Solaris Diskless Client Management Application
SUNWcddst	Simplified Chinese (EUC) Localizations for CDE Desktop Applications
SUNWcddte	Simplified Chinese (EUC) Localizations for CDE Desktop Login Environment

TABLE C–1 Simplified Chinese *(Continued)*

Package Name	Description
SUNWcdezt	Simplified Chinese (EUC) Localizations for Desktop Power Pack Applications
SUNWcdft	Simplified Chinese (EUC) Localizations for CDE Fonts
SUNWcdhcm	Simplified Chinese Localizations for DHCP Manager
SUNWcdhe	Simplified Chinese (EUC) Localizations for CDE Help Runtime environment
SUNWcdhev	Simplified Chinese (EUC) CDE Help Volumes
SUNWcdhez	Simplified Chinese (EUC) (Common) Desktop Power Pack Help Volumes
SUNWcdicn	Simplified Chinese (EUC) Localizations for CDE Icons
SUNWcdim	Simplified Chinese (EUC) Localizations for CDE Imagetool
SUNWcdwm	Simplified Chinese (EUC) Localizations for CDE Desktop Window Manager
SUNWcepmw	Simplified Chinese (EUC) Localization for Power Management OW Utilities
SUNWcfdl	Simplified Chinese Solaris Font Downloader for Adobe Postscript (tm) TCP/IP printers
SUNWcj2p	Simplified Chinese Localization of Java Plug-in 1.2.2
SUNWcj2rt	Java virtual machine and core class libraries (Simplified Chinese supplement)
SUNWckcsr	Simplified Chinese (EUC) KCMS Runtime Environment
SUNWcleex	Simplified Chinese Language Environment specific files. It is a required package to run Simplified Chinese Language Environment (64-bit) - Extension
SUNWcleue	Simplified Chinese (EUC) Language Environment specific files. It is a required package to run Simplified Chinese (EUC) Language Environment
SUNWcmga	Simplified Chinese Solaris Management Applications
SUNWcoaud	Simplified Chinese (EUC) OPENLOOK Audio Applications Package
SUNWcorte	Simplified Chinese (EUC) OPENLOOK Toolkits Runtime Environment Package
SUNWcos	This package contains Simplified Chinese Language Environment specific files. It is a required package to run Simplified Chinese Language Environment
SUNWcpdas	Simplified Chinese Localization for tools to synchronize desktop applications with the Palm Pilot PDA
SUNWcrdm	Simplified Chinese (EUC) OILBN ReadMe Directory
SUNWcreg	Simplified Chinese (EUC) Localizations for Solaris User Registration
SUNWcsadl	Simplified Chinese (EUC) Localizations for Solstice Admintool launcher and associated libraries.

TABLE C–1 Simplified Chinese *(Continued)*

Package Name	Description
SUNWcscgu	Simplified Chinese Localizable Solaris Smart Card Administration - Grahical User Interface component
SUNWcsmc	Simplified Chinese Solaris Management Console 2.0
SUNWctltk	Simplified Chinese (EUC) ToolTalk Runtime Package
SUNWcttfe	Simplified Chinese (EUC) True Type Fonts
SUNWcudc	Simplified Chinese (EUC) Localizations for User Defined Character tool for Solaris CDE environment
SUNWcwbc	Simplified Chinese Localisations for Solaris WBEM Services
SUNWcwbcp	Simplified Chinese (EUC) OpenWindows Binary Compatibility Package
SUNWcwsr	Simplified Chinese (EUC) prodreg 2.0 Localizable text resources
SUNWcwdev	Simplified Chinese Localizations for Solaris WBEM Services
SUNWcwsr2	Simplified Chinese Localizations for Solaris Product Registry
SUNWcwsrv	Simplified Chinese Localizations for Solaris Product Registry Viewer
SUNWcxe	Simplified Chinese (EUC) X Windows Platform Software Package
SUNWcxfnt	Simplified Chinese (EUC) X Windows Platform Required Fonts
SUNWcxman	Simplified Chinese (EUC) X Windows Online User Man Pages Package
SUNWgttfe	Simplified Chinese (GBK) True Type Fonts

The following table lists the French language packages and their contents.

TABLE C–2 French

Package Name	Description
NSCPfrcdo	
	French Localization of Netscape Communicator 4.7 supporting U.S. security.
NSCPfrcom	French Localization of Netscape Communicator 4.7 supporting International security.
SUNWf8adm	French (UTF-8) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWfris packages for French localization.
SUNWfbcp	French OS Binary Compatibility Package
SUNWfdcl	Diskless Client Management Application French Localization
SUNWfdhcm	French Localizations for DHCP Manager

TABLE C–2 French *(Continued)*

Package Name	Description
SUNWffdl	Localizable strings for the Font Downloader
SUNWfj2rt	Java virtual machine and core class libraries (French supplement)
SUNWfjmfp	Localizable JMF Player for playing audio and video files
SUNWfmgp	Solaris Management Applications French Localization
SUNWfbcp	French OS Binary Compatibility Package
SUNWforte	French OPEN LOOK (R) toolkits runtime environment
SUNWfpdas	French tools to synchronize desktop applications with the Palm Pilot PDA
SUNWfrbas	Base L10N fr CDE functionality to run a CDE application
SUNWfrdst	CDE Desktop Applications
SUNWfrdte	CDE Desktop Environment
SUNWfrhe	CDE Help L10N fr Runtime Environment
SUNWfrhed	CDE L10N fr Help Developer Environment
SUNWfrhev	CDE Help Volumes
SUNWfrim	CDE Desktop apps
SUNWfrj2p	French Localization of Java Plug-in 1.2.2
SUNWfros	Localizable message files for the OS-Networking consolidation
SUNWfrpmw	French (EUC) Localizations for Power Management OW Utilities
SUNWfrreg	Solaris User Registration prompts at desktop login for user registration
SUNWfrsmc	Solaris Management Console French Localization
SUNWfrwbc	French Localisations for Solaris WBEM Services
SUNWfrwm	French CDE Desktop Window Manages Messages
SUNWfrwm	French CDE Desktop Window Manages Messages
SUNWftltk	French ToolTalk binaries and shared libraries
SUNWfwacx	French OPEN LOOK (R) AccessX
SUNWfwbcp	French OpenWindows Binary Compatibility Package
SUNWfxplt	French X Windows platform software
SUNWfwdev	French Sun WBEM SDK resources

The following table lists the German language packages and their contents.

TABLE C–3 German

Package Name	Description
NSCPdecom	German Localization of Netscape Communicator 4.7 supporting International security.
SUNWdbcp	German OS Binary Compatibility Package
SUNWdebas	Base L10N German CDE functionality to run a CDE application
SUNWdedst	CDE Desktop Applications
SUNWdedte	CDE Desktop Login Environment
SUNWdehe	CDE Help L10N German Runtime Environment
SUNWdehed	CDE L10N German Help Developer Environment
SUNWdehev	CDE Help Volumes
SUNWdeim	CDE Desktop apps
SUNWdej2p	German Localization of Java Plug-in 1.2.2
SUNWdeos	Localizable message files for the OS-Networking consolidation
SUNWdepmw	German (EUC) Localizations for Power Management OW Utilities
SUNWdereg	Solaris User Registration prompts at desktop login for user registration
SUNWdesmc	Solaris Management Console German Localization
SUNWdewbc	German Localisations for Solaris WBEM Services
SUNWdewm	German CDE Desktop Window Manages Messages
SUNWdews2	German Localisations Solaris Product Registry
SUNWdewsv	German Localisations for Solaris Product Registry Viewer
SUNWdfdl	Localizable strings for the Font Downloader
SUNWdj2rt	Java virtual machine and core class libraries (German supplement)
SUNWdjmfp	Localizable JMF Player for playing audio and video files
SUNWdmgp	Solaris Management Applications German Localization
SUNWdoaud	German OPEN LOOK® Audio applications
SUNWdobk	German OpenWindows online handbooks
SUNWdorte	German OPEN LOOK® toolkits runtime environment
SUNWdpdas	German tools to synchronize desktop applications with the Palm Pilot PDA
SUNWdscgu	Localizable Solaris Smart Card Administration - Grahical User Interface component

TABLE C–3 German *(Continued)*

Package Name	Description
SUNWdtltk	German ToolTalk binaries and shared libraries
SUNWdwacx	German OPEN LOOK® AccessX
SUNWdwbcp	German OpenWindows Binary Compatibility Package
SUNWdwdev	German Sun WBEM SDK resources
SUNWdxplt	German X Windows platform software

The following table lists the Italian language packages and their contents.

TABLE C–4 Italian

Package Name	Description
SUNWi8adm Italian (UTF-8)	Localizations for Software used to perform system administration tasks. Admintool requires boththis and SUNWitis packages for Italian localization.
NSCPitcom	Italian Localization of Netscape Communicator 4.78 supporting International security.
SUNWi8adm Italian (UTF-8)	Localizations for Software used to perform system administration tasks. Admintool requires boththis and SUNWitis packages for Italian localization.
SUNWi8adm Italian (UTF-8)	Localizations for Software used to perform system administration tasks. Admintool requires boththis and SUNWitis packages for Italian localization.
SUNWibcp	Italian OS Binary Compatibility Package
SUNWidcl	Diskless Client Management Application Italian Localization
SUNWioimt	Italian OPEN LOOK (R) imagetool
SUNWidhcm	Italian Localizations for DHCP Manager
SUNWifdl	Localizable strings for the Font Downloader
SUNWij2rt	Java virtual machine and core class libraries (Italian supplement)
SUNWijmfp	Localizable JMF Player for playing audio and video files
SUNWimgp	Solaris Management Applications Italian Localization
SUNWiorte	Italian OPEN LOOK (R) toolkits runtime environment
SUNWitbas	Base L10N it CDE functionality to run a CDE application
SUNWipdas	Italian tools to synchronize desktop applications with the Palm Pilot PDA
SUNWiscgu	Localizable Solaris Smart Card Administration - Grahical User Interface component
SUNWitdst	CDE it Desktop Applications messages

TABLE C–4 Italian *(Continued)*

Package Name	Description
SUNWitdte	CDE Italian Desktop Login Environment
SUNWithe	CDE Help L10N it Runtime Environment
SUNWithed	CDE L10N it Help Developer Environment
SUNWithev	CDE Help Volumes
SUNWitim	CDE Italian Desktop Image editor
SUNWitj2p	Italian Localization of Java Plug-in 1.2.2
SUNWitltk	Italian ToolTalk binaries and shared libraries
SUNWitos	Localizable message files for the OS-Networking consolidation
SUNWitpmw	Italian (EUC) Localizations for Power Management OW Utilities
SUNWitreg	Solaris User Registration prompts at desktop login for user registration
SUNWitsmc	Solaris Management Console Italian Localization
SUNWitwbc	Italian Localisations for Solaris WBEM Services
SUNWitwm	Italian CDE Desktop Window Manages Messages
SUNWitws2	Italian Localisations Solaris Product Registry
SUNWitwsv	Italian Localisations for Solaris Product Registry Viewer
SUNWiwacx	Italian OPEN LOOK (R) AccessX
SUNWiwbcp	Italian OpenWindows Binary Compatibility Package
SUNWiwdev	Italian Sun WBEM SDK resources
SUNWixplt	Italian X Windows platform software

The following table lists the Japanese language packages and their contents.

TABLE C–5 Japanese

Package Name	Description
JSat12xw	Japanese Input System ATOK12 for Japanese Solaris.
NSCPjecom	Japanese (EUC) Localization of Netscape Communicator 4.7 supporting International security.
NSCPjpcom	Japanese (PCK) Localization of Netscape Communicator 4.7 supporting International security.

TABLE C–5 Japanese *(Continued)*

Package Name	Description
NSCPjucom	Japanese (UTF-8) Localization of Netscape Communicator 4.7 supporting International security.
SUNWjadcl	Japanese Localizations for Solaris Diskless Client Management Application.
SUNWjadis	Japanese (EUC) Localizations for admintool and GUI install.
SUNWjadma	Japanese (EUC) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWjadis packages for Japanese (EUC) Localization.
SUNWjaj2p	Japanese Localization of Java Plug-in 1.2.2
SUNWjbcp	Japanese (EUC) utilities including libc and locale data to provide a binary-compatible execution environment for SunOS 4.x applications.
SUNWjcs3f	Japanese JIS X0212 Type1 fonts for printing
SUNWjdab	Japanese (Common) Localization for CDE Desktop Application Builder
SUNWjdbas	Japanese (Common) Localization for CDE application basic runtime environment
SUNWjddst	Japanese (EUC) Localization for CDE Desktop Applications
SUNWjddte	Japanese (EUC) Localization for Solaris Desktop Login Environment
SUNWjdhcm	Japanese Localizations for DHCP Manager
SUNWjdhe	Japanese (EUC) Localization for CDE Help Runtime environment
SUNWjdhed	Japanese (EUC) Localization for CDE Help Developer Environment
SUNWjdhev	Japanese (Common) Localization for CDE Help Volumes
SUNWjdhez	Japanese (Common) Localizations for Desktop Power Pack Help Volumes
SUNWjdim	Japanese (EUC) Localization for Solaris CDE Image Viewer
SUNWjdma	Japanese Localization for CDE MAN PAGES
SUNWjdoc	Japanese Documentation Tools
SUNWjdwm	Japanese (EUC) Localization for CDE Desktop Window Manager
SUNWjeab	Japanese (EUC) Localization for CDE Desktop Application Builder
SUNWject	Japanese (EUC) Localizations for UTF-8 Code Conversion Tool
SUNWjedev	Japanese (EUC) Development Environment Package specific files
SUNWjeezt	Japanese (EUC) Localizations for Desktop Power Pack Applications
SUNWjej2m	Japanese (EUC) man pages
SUNWjeman	Japanese Feature Package Man Pages to see Japanese (EUC) manpages for SUNWjfpr and SUNWjfpu and Japanese manpages for SUNWman and SUNWaled.

TABLE C–5 Japanese *(Continued)*

Package Name	Description
SUNWjepmm	Japanese (EUC) Power Management OW Utilities Man Pages
SUNWjepmw	Japanese (EUC) Localizations for Power Management OW Utilities
SUNWjeudc	Japanese (EUC) Localizations for User Defined Character tool for Solaris CDE environment
SUNWjfdl	Japanese Localization for Solaris Desktop Font Downloader for Adobe PostScript printers
SUNWjfpre	Stream modules for Japanese Feature Package (JFP). It is a extended package to run JFP environment.
SUNWjfpue	Japanese Feature Package (JFP) specific files for user. It is a extended package to run JFP environment.
SUNWjfxmn	English manpages of Japanese features for X Window System.
SUNWjj2dv	Japanese Java virtual macTools and utilities including javac, jdb, javadoc, rmiregistry
SUNWjj2rt	Japanese Java virtual machine and core class libraries
SUNWjjmfp	Japanese Localization for JMF player
SUNWjkcsr	Japanese (EUC) Localizations for Kodak Color Management System Runtime
SUNWjmane	Japanese Feature Package Man Pages (Extension) to see English manpages for SUNWjfpre and SUNWjfpue.
SUNWjmfrn	Japanese (EUC) Localizations for Motif 1.2.3 RunTime Kit.
SUNWjmga	Japanese Solaris Management Applications
SUNWjorte	Japanese (EUC) Localizations for OPEN LOOK toolkits runtime environment
SUNWjos	Japanese ON message files
SUNWjpdas	Japanese Localization for tools to synchronize desktop applications with the Palm Pilot PDA
SUNWjpj2m	Japanese (PCK) man pages
SUNWjreg	Japanese Localizations for Solaris User Registration
SUNWjsadl	Japanese (EUC) Localizations for Solstice Admintool launcher and associated libraries.
SUNWjscag	Japanese Localization for Solaris Smart Card Administration - Grahical User Interface component
SUNWjsmc	Japanese Solaris Management Console 2.0
SUNWjtlmn	Japanese (EUC) ToolTalk manual pages for ToolTalk programmers, OpenWindows users, and Common Desktop Environment (CDE) users

TABLE C–5 Japanese *(Continued)*

Package Name	Description
SUNWjuezt	Japanese (UTF-8) Localizations for Desktop Power Pack Applications
SUNWjuhe	Japanese (UTF-8) Localization for CDE Help Runtime environment
SUNWjuhed	Japanese (UTF-8) Localization for CDE Help Developer Environment
SUNWjuhev	Japanese (UTF-8) Localization for CDE Help Volumes
SUNWjuhez	Japanese (UTF-8) Localizations for Desktop Power Pack Help Volumes
SUNWjuim	Japanese (UTF-8) Localization for Solaris CDE Image Viewer
SUNWjuj2m	Japanese (UTF-8) man pages
SUNWjujmn	Japanese (UTF-8) JavaVM Manual pages for Java programmers and users
SUNWjukcs	Japanese (UTF-8) Localizations for Kodak Color Management System Runtime
SUNWjulcf	Japanese (UTF-8) Localizations for xutops command
SUNWjuman	Japanese Feature Package Man Pages to see Japanese (UTF-8) man pages for SUNWjfpr and SUNWjfpu and Japanese man pages for SUNWman and SUNWaled.
SUNWjumfr	Japanese (UTF-8) Localizations for Motif 1.2.3 RunTime Kit.
SUNWjupmm	Japanese (UTF-8) Power Management OW Utilities Man Pages
SUNWjupmw	Japanese (UTF-8) Localizations for Power Management OW Utilities
SUNWjurdm	Japanese (UTF-8) OILBN ReadMe Directory
SUNWjurme	Japanese (UTF-8) Localization for Common Desktop Environment (CDE) release documentation
SUNWjurvl	Japanese (UTF-8) Localizations for XIL loadable pipelines for SunVideo capture and compression
SUNWjusal	Japanese (UTF-8) Localizations for Solstice Admintool launcher and associated libraries.
SUNWjutlm	Japanese (UTF-8) ToolTalk manual pages for ToolTalk programmers, OpenWindows users, and Common Desktop Environment (CDE) users
SUNWjutlt	Japanese (UTF-8) Localizations for ToolTalk binaries and shared libraries needed for Common Desktop Environment (CDE), OpenWindows, and all ToolTalk clients
SUNWjuudc	Japanese (UTF-8) Localizations for User Defined Character tool for Solaris CDE environment
SUNWjuwm	Japanese (UTF-8) Localization for CDE Desktop Window Manager
SUNWjuwnu	Japanese Input System, Wnn6 Messages, (UTF-8)
SUNWjuxfa	Japanese (UTF-8) Localizations for Font Administration application for Solaris platforms

TABLE C–5 Japanese *(Continued)*

Package Name	Description
SUNWjuxir	Japanese (UTF-8) Localizations for XIL Runtime Environment
SUNWjwacx	Japanese (EUC) Localizations for AccessX client program
SUNWjwbc	Japanese Localizations for Solaris WBEM Services
SUNWjwbcp	Japanese (EUC) Localizations for Support files, programs, and libraries for OpenWindows Binary Compatibility.
SUNWjwbd	Japanese Localizations for Sun WBEM SDK resources
SUNWjwncr	Japanese Input System, Wnn6 Client, (Root)
SUNWjwncu	Japanese Input System, Wnn6 Client, (Usr)
SUNWjwncx	Japanese Input System, Wnn6 Client X Window System
SUNWjwndt	Japanese Input System, Wnn6 Client for CDE
SUNWjwnsr	Japanese Input System, Wnn6 Server, (Root)
SUNWjws2	Japanese Localizations for Solaris Product Registry
SUNWjwsv	Japanese Localizations for Solaris Product Registry Viewer
SUNWjwnsu	Japanese Input System, Wnn6 Server, (Usr)
SUNWjwsr	Japanese Solaris Product Registry
SUNWjxfa	Japanese (Common) Localizations for Font Administration application for Solaris platforms
SUNWjxfnt	Japanese X Window System Fonts (required fonts), gothic bold fonts and TrueType map files
SUNWjxim	Japanese X Window System X Input Method Server Package
SUNWjxoft	Sun Minchou bitmap fonts
SUNWjxplt	Japanese Localizations for X Window System platform software (Extensions)
SUNWjxpmn	Japanese (EUC) X Window System online programmers man pages
SUNWjxumn	Japanese (EUC) X Window System online user man pages

The following table lists the Korean language packages and their contents.

TABLE C–6 Korean

Package Name	Description
NSCPkocom	Korean Localization of Netscape Communicator 4.7 supporting International security.
SUNWkbcp	This package contains Korean Language Environment binary compatibility files.
SUNWkacx	Korean AccessX client program
SUNWkcoft	Korean/Korean UTF-8 common optional font package
SUNWkdab	Korean Localizations for CDE Desktop Application Builder
SUNWkdbas	Korean Localizations for CDE Base functionality
SUNWkdcl	Korean Localizations for Solaris Diskless Client Management Application
SUNWkdcst	The Localized tools package for Korean.
SUNWkddst	Korean Localizations for CDE Desktop Applications
SUNWkddte	Korean Localizations for CDE Desktop Login Environment
SUNWkdezt	Korean (EUC) Localizations for Desktop Power Pack Applications
SUNWkdft	Fonts for the common desktop environment, Korean L10N CDE
SUNWkdhcm	Korean Localizations for DHCP Manager
SUNWkdhev	Korean CDE Help Volumes
SUNWkdhez	Korean (Common) Localizations for Desktop Power Pack Help Volumes
SUNWkdicn	Korean Localizations for CDE Icons
SUNWkdim	Korean Localizations for CDE Imagetool
SUNWkdwm	Korean Localizations for CDE Desktop Window Manager
SUNWkepmw	Korean (EUC) Localization for Power Management OW Utilities
SUNWkexir	Korean (EUC) XIL Runtime Environment
SUNWkfdl	Korean Solaris Font Downloader for Adobe Postscript (tm) TCP/IP printers
SUNWkj2rt	Java virtual machine and core class libraries (Korean supplement)
SUNWkjmfp	Korean Localization for JMF player
SUNWkkcsr	Korean (EUC) KCMS Runtime Environment
SUNWkleex	Korean Language Environment specific files. It is a required package to run Korean Language Environment (64-bit)

TABLE C–6 Korean *(Continued)*

Package Name	Description
SUNWkler	This package contains the stream modules for Korean Language Environment. It is a required package to run Korean Language Environment
SUNWkleue	This package contains Korean Language Environment specific files. It is a required package to run Korean Language Environment
SUNWkmga	Korean Solaris Management Applications
SUNWkpdas	Korean Localization for tools to synchronize desktop applications with the Palm Pilot PDA
SUNWkoj2p	Korean Localization of Java Plug-in 1.2.2
SUNWkorte	Korean OPENLOOK Toolkits Runtime Environment Package
SUNWkos	This package contains Korean Language Environment specific files. It is a required package to run Korean Language Environment
SUNWkpdas	Korean Localization for tools to synchronize desktop applications with the Palm Pilot PDA
SUNWkttfe	Korean True Type Font Extension
SUNWkuadi	Korean (UTF-8) Localizations for admintool and GUI install.
SUNWkudc	Korean (EUC) Localizations for User Defined Character tool for Solaris CDE environment
SUNWkudhz	Korean (Common) Localizations for Desktop Power Pack Help Volumes
SUNWkudic	Korean/UTF-8 Localizations for CDE Icons
SUNWkudim	Korean/UTF-8 Localizations for CDE Imagetool
SUNWkudwm	Korean/UTF-8 Localizations for CDE Desktop Window Manager
SUNWkuxft	Korean UTF-8 X Windows Platform Required Fonts
SUNWkwbc	Korean Localisations for Solaris WBEM Services
SUNWkwbcp	Korean OpenWindows Binary Compatibility Package
SUNWkwdev	Korean Localizations for Solaris WBEM Services
SUNWkwsr	Korean prodreg 2.0 Localizable text resources
SUNWkwsrv	Korean Localizations for Solaris Product Registry Viewer
SUNWkxe	Korean X Windows Platform Software Package
SUNWkxfte	Korean X Windows Platform Required Fonts
SUNWkxman	Korean X Windows Online User Man Pages Package

The following table lists the Spanish language packages and their contents.

TABLE C–7 Spanish

Package Name	Description
NSCPescom	Spanish Localization of Netscape Communicator 4.7 supporting International security.
SUNWe8adm	Spanish (UTF-8) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWesis packages for Spanish (UTF-8) localization.
SUNWedcl	Diskless Client Management Application Spanish Localization
SUNWedhcm	Spanish Localizations for DHCP Manager
SUNWefdl	Localizable strings for the Font Downloader
SUNWej2rt	Java virtual machine and core class libraries (Spanish supplement)
SUNWejmfp	Localizable JMF Player for playing audio and video files
SUNWemgp	Solaris Management Applications Spanish Localization
SUNWeorte	Spanish OPEN LOOK (R) toolkits runtime environment
SUNWesbas	Base L10N fr CDE functionality to run a CDE application
SUNWescgu	Localizable Solaris Smart Card Administration - Grahical User Interface component
SUNWesdst	CDE Desktop Applications
SUNWesdte	CDE Desktop Login Environment
SUNWeshe	CDE Help L10N es Runtime Environment
SUNWeshed	CDE L10N es Help Developer Environment
SUNWeshev	CDE Help Volumes
SUNWesim	CDE Desktop apps
SUNWesj2p	Spanish Localization of Java Plug-in 1.2.2
SUNWesos	Localizable message files for the OS-Networking consolidation
SUNWespmw	Spanish (EUC) Localizations for Power Management OW Utilities
SUNWesreg	Solaris User Registration prompts at desktop login for user registration
SUNWessmc	Solaris Management Console Spanish Localization
SUNWeswbc	Spanish Localisations for Solaris WBEM Services
SUNWeswm	Spanish CDE Desktop Window Manages Messages
SUNWesws2	Spanish Localisations Solaris Product Registry
SUNWeswsv	Spanish Localisations for Solaris Product Registry Viewer

TABLE C–7 Spanish *(Continued)*

Package Name	Description
SUNWetltk	Spanish ToolTalk binaries and shared libraries
SUNWewacx	Spanish OPEN LOOK (R) AccessX
SUNWewdev	Spanish Sun WBEM SDK resources
SUNWexplt	Spanish X Windows platform software

The following table lists the Swedish language packages and their contents.

TABLE C–8 Swedish

Package Name	Description
NSCPsvcom	Swedish Localization of Netscape Communicator 4.7 supporting International security.
SUNWs8adm	Swedish (UTF-8) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWsvis packages for Swedish localization.
SUNWsdcl	Diskless Client Management Application Swedish Localization
SUNWsdhcm	Swedish Localizations for DHCP Manager
SUNWsfdl	Localizable strings for the Font Downloader
SUNWsj2rt	Java virtual machine and core class libraries (Swedish supplement)
SUNWsjmfp	Localizable JMF Player for playing audio and video files
SUNWsmgp	Solaris Management Applications Swedish Localization
SUNWsorte	Swedish OPEN LOOK (R) toolkits runtime environment
SUNWspdas	Swedish tools to synchronize desktop applications with the Palm Pilot PDA
SUNWsscgu	Localizable Solaris Smart Card Administration - Grahical User Interface component
SUNWstltk	Swedish ToolTalk binaries and shared libraries
SUNWsvbas	Base Swedish CDE functionality messages
SUNWsvdst	Swedish CDE Desktop Applications messages
SUNWsvdte	Swedish CDE Desktop Login Environment messages
SUNWsvhe	Swedish CDE Help Runtime Environment
SUNWsvhed	Swedish CDE Help Developer Environment messages
SUNWsvhev	CDE Help Volumes

TABLE C–8 Swedish *(Continued)*

Package Name	Description
SUNWsvim	Swedish CDE Image editor messages
SUNWsvj2p	Swedish Localization of Java Plug-in 1.2.2
SUNWsvos	Localizable message files for the OS-Networking consolidation
SUNWsvpmw	Swedish (EUC) Localizations for Power Management OW Utilities
SUNWsvreg	Solaris User Registration prompts at desktop login for user registration
SUNWsvsmc	Solaris Management Console Swedish Localization
SUNWsvwm	Swedish CDE Desktop Window Manages Messages
SUNWsvws2	Swedish Localisations Solaris Product Registry
SUNWsvwsv	Swedish Localisations for Solaris Product Registry Viewer
SUNWswacx	Swedish OPEN LOOK (R) AccessX
SUNWsvwsv	Swedish Localisations for Solaris Product Registry Viewer
SUNWsxplt	Swedish X Windows platform software
SUNWvbcp	Swedish OS Binary Compatibility Package
SUNWvwbcp	Swedish OpenWindows Binary Compatibility Package

The following table lists the Traditional Chinese language packages and their contents.

TABLE C–9 Traditional Chinese

Package Name	Description
NSCP5com	zh_TW.BIG5 Localization of Netscape Communicator 4.7 supporting International security.
NSCPhcom	Traditional Chinese Localization of Netscape Communicator 4.7 supporting International security.
NSCPhucom	zh_TW.UTF-8 Localization of Netscape Communicator 4.7 supporting International security.
SUNW5ttfe	Traditional Chinese True Type Fonts Package Extension
SUNW5udc	Traditional Chinese (BIG5) Localizations for User Defined Character tool for Solaris CDE environment
SUNW5xfnt	Traditional Chinese BIG5 X Windows Platform required Fonts Package
SUNWhacx	Traditional Chinese AccessX client program.
SUNWhadis	Traditional Chinese (EUC) Localizations for admintool and GUI install.

TABLE C–9 Traditional Chinese *(Continued)*

Package Name	Description
SUNWhadma	Traditional Chinese (EUC) Localizations for Software used to perform system administration tasks. Admintool requires both this and SUNWhadis packages for Traditional Chinese (EUC) Localization.
SUNWhbcp	This package contains Traditional Chinese Language Environment binary compatibility files.
SUNWhdab	Traditional Chinese Localizations for CDE Desktop Application Builder
SUNWhdbas	Traditional Chinese Localizations for CDE Base functionality
SUNWhdcl	Traditional Chinese Localizations for Solaris Diskless Client Management Application.
SUNWhddst	Traditional Chinese Localizations for CDE Desktop Applications
SUNWhddte	Traditional Chinese Localizations for CDE Desktop Login Environment
SUNWhdezt	Traditional Chinese (EUC) Localizations for Desktop Power Pack Applications
SUNWhdft	Traditional Chinese Localizations for CDE Fonts
SUNWhdhcm	Traditional Chinese Localizations for DHCP Manager
SUNWhdhe	Traditional Chinese Localizations for CDE Help Runtime environment
SUNWhdhev	Traditional Chinese CDE Help Volumes
SUNWhdhez	Traditional Chinese (Common) Localizations for Desktop Power Pack Help Volumes
SUNWhdicn	Traditional Chinese Localizations for CDE Icons
SUNWhdim	Traditional Chinese Localizations for CDE Imagetool
SUNWhdwm	Traditional Chinese Localizations for CDE Desktop Window Manager
SUNWhepmw	Traditional Chinese (EUC) Localization for Power Management OW Utilities
SUNWhfdl	Traditional Chinese Solaris Font Downloader for Adobe Postscript (tm) TCP/IP printers
SUNWhj2p	Traditional Chinese Localization of Java Plug-in 1.2.2
SUNWhj2rt	Java virtual machine and core class libraries (Traditional Chinese supplement)
SUNWhjmfp	Traditional Chinese Localization for JMF player
SUNWhkcsr	Traditional Chinese (EUC) KCMS runtime environment
SUNWhkeex	Traditional Chinese (Hong Kong BIG5) Language Environment specific files. It is a required package to run Traditional Chinese (Hong Kong BIG5) Language Environment (64-bit)

TABLE C–9 Traditional Chinese *(Continued)*

Package Name	Description
SUNWhkeue	This package contains Traditional Chinese (Hong Kong) Language Environment specific files. It is a required package to run Traditional Chinese (Hong Kong) Language Environment
SUNWhkxe	Traditional Chinese(Hong Kong) X Windows Platform Software Package
SUNWhleex	Traditional Chinese Language Environment specific files. It is a required package to run Traditional Chinese Language Environment (64-bit)
SUNWhmga	Traditional Chinese Solaris Management Applications
SUNWhorte	Traditional Chinese OPENLOOK Toolkits Runtime Environment Package
SUNWhos	This package contains Traditional Chinese Language Environment specific files. It is a required package to run Traditional Chinese Language Environment
SUNWhpdas	Traditional Chinese Localization for tools to synchronize desktop applications with the Palm Pilot PDA
SUNWhreg	Traditional Chinese Localizations for Solaris User Registration
SUNWhsadl	Traditional Chinese Localizationsfor Solstice Admintool launcher and associated libraries.
SUNWhscgu	Traditional Chinese Localizable Solaris Smart Card Administration - Grahical User Interface component
SUNWhsmc	Traditional Chinese Solaris Management Console 2.0
SUNWhtltk	Traditional Chinese ToolTalk Runtime Package Package
SUNWhttfe	Traditional Chinese True Type optional Fonts Package Extension
SUNWhuccd	This package contains Traditional Chinese Console Display Environment specific files. It is a required package to run Traditional Chinese Console Display Environment
SUNWhudc	Traditional Chinese (EUC) Localizations for User Defined Character tool for Solaris CDE environment
SUNWhwbcp	Traditional Chinese OpenWindows Binary Compatibility Package
SUNWhwdev	Traditional Chinese Localizations for Solaris WBEM Services
SUNWhwsr2	Traditional Chinese Localizations for Solaris Product Registry
SUNWhwsrv	Traditional Chinese Localizations for Solaris Product Registry Viewer
SUNWhxe	Traditional Chinese X Windows Platform Software Package
SUNWhxman	Traditional Chinese X Windows Online User Man Pages Package

The following table lists the shared packages and their contents.

TABLE C–10 Shared

Package Name	Description
SUNWabcp	Asian common files for SunOS 4.x Binary Compatibility
SUNWerdm	OILBN ReadMe Directory
SUNWudct	User Defined Character tool for Solaris CDE environment

Index