International Language Environments Guide for Oracle® Solaris 11.3

Part No: E54757
November 2016

ORACLE®

International Language Environments Guide for Oracle Solaris 11.3

Part No: E54757

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Référence: E54757

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité à la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Describes the internationalization features in the Oracle Solaris OS.
- **Audience** – Software developers and system administrators who design and support global applications in the Oracle Solaris OS.
- **Required knowledge** – The readers of this guide should have knowledge of the C programming language.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Introduction

This chapter introduces the concepts of internationalization and localization in the Oracle Solaris operating system. The following topics are covered in this chapter:

Oracle Solaris and the Global Market

The Oracle Solaris OS is an internationalized operating system based on the latest international standards, such as Unicode 6.0. Users around the world can work with the system in their language and with their regional preferences. Oracle Solaris includes support for over 60 languages in about 100 different territories, totalling about 250 different locales. About 140 locales are based on Unicode, and support for various legacy locales based on ISO8859, EUC, and other codesets is still available. Unicode and legacy locales are described in detail in Chapter 2, "Unicode and `UTF-8` Locale Support". The following sections provide an overview of internationalization and localization, and explains the concept of locales.

Internationalization and Localization Overview

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

Internationalized software works with different native languages and customs and can be ported from one locale to another without rewriting the software. The Oracle Solaris system is internationalized, providing the infrastructure and interfaces you need to create internationalized software.

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 POSIX standard which can be accessed through `http://opengroup.org`.

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

A locale name can be composed of a base language, country (territory) of use, and codeset. For example, German language is `de`, an abbreviation for Deutsch, while Swiss German is `de_CH`, `CH` being an abbreviation for Confederation Helvetica. This convention allows for specific differences by country, such as currency unit notation. In Oracle Solaris 11 the default locale codeset is `UTF-8`, an ASCII compatible 8-bit encoding form of Unicode. The fully defined locale name for Swiss German would thus be `de_CH.UTF-8`.

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.UTF-8` locale (English for the United States), while an English-speaking user in Great Britain can select `en_GB.UTF-8` (English for Great Britain).

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

The locale naming convention is:

`language[_`*territory*`][.`*codeset*`][@`*modifier*`]`

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

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

For more information about the portable character set, refer to *X/Open CAE Specification: System Interface Definitions, Issue 5" (ISBN 1-85912-186-1)*.

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

C Locale

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

You can specify your internationalized programs to run in the C locale in the following two ways:

- Unset all locale environment variables. Runs the application in the C locale.

  ```
  $ unset LC_ALL LANG LC_CTYPE LC_COLLATE LC_NUMERIC LC_TIME LC_MONETARY LC_MESSAGES
  ```

- Explicitly set the locale to C or POSIX.

  ```
  $ export LC_ALL=C
  $ export LANG=C
  ```

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

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

```
$ locale
LANG=C
LC_CTYPE="C"
LC_NUMERIC="C"
LC_TIME="C"
LC_COLLATE="C"
LC_MONETARY="C"
LC_MESSAGES="C"
LC_ALL=
```

Locale Categories

The types of locale categories are as follows:

LC_CTYPE Character classification and case conversion.

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

Core Locales

The following table lists Oracle Solaris 11 core locales:

TABLE 1 Languages and Core locales

Language	Core locale
Chinese - Simplified	`zh_CN.UTF-8`
Chinese - Traditional	`zh_TW.UTF-8`
English	`en_US.UTF-8`
French	`fr_FR.UTF-8`
German	`de_DE.UTF-8`
Italian	`it_IT.UTF-8`
Japanese	`ja_JP.UTF-8`
Korean	`ko_KR.UTF-8`
Portuguese - Brazilian	`pt_BR.UTF-8`
Spanish	`es_ES.UTF-8`

Core locales have better coverage at the level of localized messages than the locales available for additional installation. Oracle Solaris OS components such as Installer or Package Manager are localized only in core locales while localized messages for third-party software such as GNOME or Firefox are often available in more locales.

All locales in the Oracle Solaris environment are capable of displaying localized messages, provided that the localized messages for the relevant language and application are present. Additional locales including all their available localized messages can be added to the system from the installation repository by modification of pkg facet properties. For more information, see “Managing Available Locales” on page 36.

Behavior Affected by Locales

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

- Encoding and processing of text data
- Language identification and encoding of resource files
- Rendering and layout of text strings
- Interchange of text between clients
- Input method selection to meet the codeset and text processing requirements of the chosen script
- Fonts and icon files that are culturally specific
- User Interface Definition (UID) files
- Date and time formats
- Numeric formats
- Monetary formats
- Collation order
- Regular expression handling
- Format for informative and diagnostic messages and interactive responses

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

The following sections describe the differences that exist for locale categories and other differences between languages.

Time Formats

The following table shows some of the ways in which different locales write 11:59 P.M. You can display the time format on your current locale by issuing the following command:

```
$ date +%X
```

TABLE 2 International Time Formats

Locale	Description	Time Format
C	-	23:59:00
en_US.UTF-8	English, U.S.A.	11:59:00 PM
es_US.UTF-8	Spanish, U.S.A.	11:59:00 p.m.
mr_IN.UTF-8	Marathi, India	11-59-00 pm
sq_AL.ISO8859-2	Albanian, Albania	11.59.00.MD
ja_JP.UTF-8	Japanese, Japan	23時59分00秒
ko_KR.UTF-8	Korean, Korea	오후11시 59분 00초
zh_CN.UTF-8	Simplified Chinese, China	23时59分00秒

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

Time zone splits occur between and within countries. Although a time zone can be described in terms of the number of 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 Saving Time (DST) starts and ends on dates that can vary from country to country. Many countries do not implement DST at all. Additionally, Daylight Saving Time can vary within a time zone. In the U.S. for example, the implementation is a state decision.

Date Formats

The following table shows some of the date formats used around the world. Variations can exist even within a country. You can display the date format on your current locale by issuing the following command:

```
$ date +%x
```

TABLE 3 International Date Formats

Locale	Description	Date Format
C	-	07/16/11
en_CA.UTF-8	English, Canada	7/11/16
en_GB.UTF-8	English, United Kingdom	7/16/11
fi_FI.UTF-8	Finnish, Finland	7/16/11
ja_JP.UTF-8	Japanese, Japan	2011年07月16日
ko_KR.UTF-8	Korean, Korea	2011년07월16일

Locale	Description	Date Format
zh_TW.UTF-8	Traditional Chinese, Taiwan	11年07月16日

Number Formats

There are various number formats specified by locales, for example Great Britain and the United States use a period to indicate the decimal place. Many other countries use a comma instead. The decimal separator is also called the *radix* character. Likewise, while Great Britain and the United States use a comma to separate groups of thousands, many other countries use a period instead, and some countries separate thousands groups with a thin space (Unicode character U+2009).

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

The following table shows some commonly used numeric formats. The information on numeric delimiters for current locale can be obtained by issuing the following command:

```
$ locale -ck LC_NUMERIC
```

TABLE 4 International Numeric Conventions

Locale	Description	Number Format
C	-	4294967.00
ar_SA.UTF-8	Arabic, Saudi Arabia	4967967,00
cs_CZ.UTF-8	Czech, Czech Republic	4 294 967,00
de_DE.UTF-8	German, Germany	4.294.967,00
de_CH.UTF-8	German, Switzerland	4'294'967.00
en_US.UTF-8	English, U.S.A.	4,294,967.00
hi_IN.UTF-8	Hindi, India	42,94,967.00

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

International Monetary Formats

Currency units, presentation order, and local and international symbols for currency vary greatly around the world. The monetary formats for current locale can be obtained by issuing the following command:

```
$ locale -ck LC_MONETARY
```

The following table shows monetary formats in some countries.

TABLE 5 International Monetary Conventions

Locale	Currency	Example
C	-	1234.56
da_DK.UTF-8	Danish krone (kr)	1.234,56kr
da_DK.ISO8859-15@euro	Euro (€)	1.234,56€
en_GB.UTF-8	Pound (£)	£1,234.56
en_US.UTF-8	United States dollar ($)	$1,234.56
fr_FR.UTF-8	Euro (€)	1 234,56€
ja_JP.UTF-8	Japanese yen (¥)	¥1,235
th_TH.UTF-8	Thai Baht (#)	#1,234.56
zh_CN.UTF-8	Chinese yuan (¥)	¥1,234.56

The Euro currency is supported in all UTF-8 locales. Legacy locales based on the ISO8859-15 code set are also available and exist with @euro (e.g. da_DK.ISO8859-15@euro) variants for countries which have not adopted Euro as their currency.

Language Word and Letter Differences

This section describes important differences between languages.

Word Delimiters

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

Sort Order

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

Character Sets

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

Western European Alphabets

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

Japanese Text

Japanese text is composed of three different scripts mixed together:

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

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

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

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

Korean Text

Korean text can be written using a phonetic writing system called Hangul. Hangul has more than 11,000 characters, which consist of consonants and vowels known as jamos. About 3000 characters from the entire Hangul vocabulary of characters are usually used in Korean computer

systems. Korean also uses ideographs based on the set invented in China, called Hanja. Korean text requires over 6000 Hanja characters. Hanja is used mostly to avoid confusion when Hangul would be ambiguous. Hangul characters are formed by combining consonants and vowels. After these characters are combined, they can compose one syllable, which is a Hangul character. Hangul characters are often arranged in a square, so that the group takes up the same space as a Hanja character. Arabic numerals, Roman letters, and special symbol characters are also present in Korean text.

Thai Text

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

Chinese Text

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

- In the People's Republic of China there are about 7000 commonly used Hanzi characters in the GB2312 (`zh_CN.EUC` locale), more than 20,000 characters in the GBK charset (`zh_CN.GBK` locale), and about 30,000 characters in the GB18030-2000 charset (`zh_CN.GB18030` locale), including all CJK Unified Ideographs Extension A characters defined in Unicode 6.0.
- In Taiwan, the most frequently used charsets are the CNS11643-1992 (`zh_TW.EUC` locale) and the Big5 (`zh_TW.BIG5` locale). They share about 13,000 Hanzi characters.
- In Hong Kong, 4702 characters have been added into the Big5 charset to become the Big5-HKSCS charset (`zh_HK.BIG5HK` locale).

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

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

Several phonetic systems represent Chinese. In the People's Republic of China the most common is *pinyin*, which uses Roman characters and is widely employed in the West for place

names such as Beijing. The Wade-Giles system is an older phonetic system, formerly used for place names such as Peking. In Taiwan *zhuyin* (or *bopomofo*), a phonetic alphabet with unique letter forms, is often used instead.

Hebrew Text

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

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

Hindi Text

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

Keyboard Differences

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

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

Differences in Paper Sizes

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

Internationalized applications should not make assumptions about the page sizes available to them. The Oracle Solaris system does not provide any support for tracking the output page size. This tracking is the responsibility of the application program. The following table shows common international page sizes.

TABLE 6 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

Unicode and UTF-8 Locale Support

Text strings in a computer are represented as sequence of character codes. A character set is a mapping between a character and the character code. The character set used for encoding strings is one of the most important characteristics of a locale. This chapter describes character sets used in Oracle Solaris locales and also the data source used for their creation. It covers the following topics:

Using the wrong character set to display a string often results in broken output. Therefore sometimes it is necessary to convert strings in order to use a different character set. Such conversions are described in "Migrating From Non-UTF-8 Locales to UTF-8 Locales" on page 26.

Unicode Overview

Unicode is the universal character encoding standard used for representation of text for computer processing. Unicode provides a consistent way of encoding multilingual text and facilitates exchanging of international text files.

The standard for coding multilingual text is *ISO/IEC 10646*. Although the ISO/IEC 10646 and Unicode standards contain all the same characters and encoding points, the Unicode standard provides additional information about the characters and their use.

Oracle Solaris 11 provides system-level support for the Unicode Standard Version 6.0 and ISO/IEC 10646:2011.

Each Unicode character is mapped to a code point, which is an integer between 0 and 1,114,111. Unicode code points are referred to using notation in the form U+nnnn, where nnnn is the code

point's hexadecimal number, or by a text string describing the code point. For example, the lower case letter “a” can be represented by U+0061or the text string "LATIN SMALL LETTER A".

Code points can be encoded using different character encoding schemes. In Oracle Solaris Unicode locales, the UTF-8 form is used. UTF-8 is a variable-length encoding form of Unicode that preserves ASCII character code values transparently (see “UTF-8 Overview” on page 24).

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

- The Unicode Standard, Version 6.0 from the Unicode Consortium
- ISO/IEC 10646:2011, Information Technology-Universal Multiple-Octet Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane
- The Unicode Consortium web site

UTF-8 Overview

UTF-8 is a variable-length encoding form of Unicode. This form is used in Oracle Solaris Unicode locales.

The advantage of this form is that it is backward compatible with the ASCII encoding scheme and avoids the complications of endianness and byte order. Unicode code points are in UTF-8 represented by one to four 8-bit bytes. The following table specifies the bit distribution for UTF-8, showing the ranges of Unicode code points corresponding to one-byte, two-byte, three-byte, and four-byte sequences.

TABLE 7 Bit Distribution of UTF-8

Code Point Range	Code Point (binary)	1st Byte	2nd Byte	3rd Byte	4th Byte
U+0000..U+007F	0xxxxxxx	0xxxxxxx			
U+0080..U+07FF	00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
U+0800..U+FFFF	zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
U+010000..U+10FFFF	000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

For more details about the UTF-8 encoding form, refer to the following sources:

- The Unicode Standard, Version 6.0, Chapter 3 (http://www.unicode.org/versions/Unicode6.0.0/ch03.pdf), Section 3.9 “Unicode Encoding Forms”, pp. 93 - 94
- The Unicode Consortium web site

Common Locale Data Repository

The Common Locale Data Repository (CLDR) Project , is a project of the Unicode Consortium to provide extensive locale data. CLDR contains locale-specific information that an operating system will typically provide to applications. This information includes number formats, date, time and currency strings, character classification (lower, upper, printable, and the like), string collation rules and others. For more information, see “Behavior Affected by Locales” on page 15.

All Oracle Solaris 11 locale data is aligned to the CLDR data version 1.9.0, except the non-`UTF-8` forms of the `ja_JP`, `ko_KR`, `th_TH`, `zh_CN`, `zh_HK` and `zh_TW` locales and the `ja_JP.UTF-8` locale. These locales are based on other sources and do not use the CLDR data. There is a CLDR-based variant of Japanese `UTF-8` available under the name `ja_JP.UTF-8@cldr`.

For more details on CLDR, refer to The Unicode Common Locale Data Repository project site.

Locales With Non-`UTF-8` Character Sets

To avoid conversion issues, Oracle Solaris locales use the `UTF-8` encoding form described in “`UTF-8` Overview” on page 24 of the Unicode character set. All supported languages have a `UTF-8` locale as the preferred and supported form.

For historical, technical, and legal reasons, non-`UTF-8` locales are also available in Oracle Solaris - the `C` locale, legacy single-byte (8–bit) ISO locales for EMEA languages, and traditional locales for APAC languages.

Single-byte character sets were popular in the past because they used just one byte (8 bits) to represent one character. But due to the limited size of the sets (a maximum of 256 characters), different languages have to use different character sets. This introduces many problems - a file created in one character set is often unreadable in another character set, representing a multilanguage document is an issue, and also many languages have more characters than can be represented by a single byte, and the like. For these languages, such as Chinese, different traditional multibyte character sets were created.

The non-`UTF-8` locales, also called *legacy* or *traditional* locales, have limited support in Oracle Solaris 11. These limited support locales are not available in the GDM login dialog and are not installed by default. Localization that exists for a `UTF-8` locale might not be available in the non-`UTF-8` locale variant. Some of the limited support locales might be removed from future Oracle Solaris releases.

The legacy locales are not installed by Oracle Solaris installer. To enable these locales, you must install the `system/locale/extra` package manually, for example,

```
# pkg install system/locale/extra
```

Locale facets also need to be set correctly. For more information, see "Locale Facets" on page 37.

Migrating From Non-UTF-8 Locales to UTF-8 Locales

When migrating to UTF-8 locales, the method used for importing or exporting data depends on the file type.

Plain Text Files

Plain text files do not have an explicit identification of the files' character encoding. If the files are not in the UTF-8 encoding, conversion is needed. For example, you would run the following command to convert a plain text file encoded in Traditional Chinese big5 to UTF-8:

```
$ iconv -f big5 -t UTF-8 inputfilename > outputfilename
```

The Text Editor application can read and write character encoding text automatically, or you can specify an encoding explicitly when opening or saving a file. To start the Text Editor, choose Launch → Applications → Accessories → Text Editor.

File Names and Directory Names

File systems like UFS or ZFS store file and directory names in the character set you use. If you use non-UTF-8 locales or mount a non-UTF-8 file system and move to a UTF-8 locale, you might see garbage characters in the file names. To fix this problem, use convmv(1) can be used to convert a single file name, a directory tree and the contained files or a whole file system into a different encoding. It only converts the file names, not the contents of the files.

See the convmv(1) man page for more information. The tool works on any file system.

Alternatively, fsexam(1) can be used for this purpose. For more information, see "File Examiner (fsexam)" on page 68.

ZFS

ZFS is the main file system used in Oracle Solaris 11. ZFS uses the locale's character set to store file and directory names like other file systems. For the UTF-8 character set, the

normalization property sets the type of normalization algorithm used by the file system for comparing names to avoid having more than one entity with the same file name in a single directory.

If the utf8only property is enabled, the file system will reject file names that include characters not present in the UTF-8 character set.

See the zfs(1) man page for more information.

NFS

For more information, see "Interoperability with Other Platforms" on page 72

♦ ♦ ♦ CHAPTER 3

Working with Languages and Locales

This chapter describes how to select, install and use languages, and locales in Oracle Solaris 11 system. It covers the following topics:

Managing User's Locale

This section provides information about how to set locale and how to select locale in GNOME display manager.

Setting the Locale in a Terminal Session

You can change the locale in a terminal session by setting the `LANG` variable as follows:

```
$ export LANG=locale
```

For example, to change to the `de_DE.UTF-8` locale, use the following command:

```
$ export LANG=de_DE.UTF-8
```

To verify the locale has been successfully changed, run the `locale` command:

```
$ locale
LANG=de_DE.UTF-8
LC_CTYPE="de_DE.UTF-8"
LC_NUMERIC="de_DE.UTF-8"
LC_TIME="de_DE.UTF-8"
LC_COLLATE="de_DE.UTF-8"
LC_MONETARY="de_DE.UTF-8"
LC_MESSAGES="de_DE.UTF-8"
LC_ALL=
```

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

To obtain the list of locales available in a system, use the following command:

```
$ locale -a
```

For more information about installing more locales, see “Managing Available Locales” on page 36.

Composite Locales

The `LC*` variables, such as `LC_CTYPE` or `LC_MESSAGES`, described in “Locale Categories” on page 13, can also be set in a terminal along with the `LANG` variable. When the `LC*` variable is set, they override the `LANG` setting for the particular category. This type of locale setting is called *composite locale*.

```
$ export LANG=de_DE.UTF-8
$ export LC_MESSAGES=en_US.UTF-8
```

In this example, applications that correctly handle the locale settings would operate in German locale but have their localized output printed in English. The output of `locale` command in this case would be as follows:

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

The `LC_MESSAGES` variable is in this case printed without apostrophes, indicating that the value is explicitly set. The other `LC*` variables have their value inherited from the `LANG` variable. The `LC_ALL` variable can be used to override all of the `LANG` and `LC*` settings. For more information, see the `locale(1)` and `setlocale(3C)` man pages.

Caution - Do not use multiple character sets in composite locales. For example, setting `LANG=en_US.UTF-8` and `LC_MESSAGES=ja_JP.eucJP` leads to unpredictable results.

Locale Selection in GNOME Display Manager

Oracle Solaris 11 provides locale selection in GNOME Display Manager (GDM).

▼ How to Select a Locale in GNOME Display Manager

1. **Type your user name and click the Log In button or press Return.**

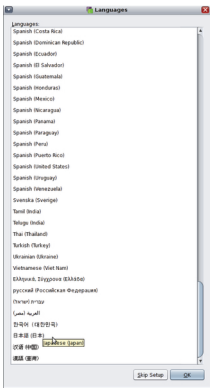
2. **Click the Log In button or press Return.**

   The Locale Selection panel becomes available at the bottom of the GDM window, as shown in the following figure.

   If no locale preference is set during the installation process, the default locale value is set to the `C` locale (also known as the `POSIX` locale).

3. **Choose Other to view a list of all available locales.**

   A window containing a list of all locales installed on the system appears.



4. **Select the desired locale and click OK.**

▼ How to Start a Session in a Different Locale

1. **Type your user name and click the Log In button or press Return.**
2. **Select a Locale from the Locale Selection menu on the login screen.**
3. **Type your password.**
4. **Click the Log In button.**

Persistent Locale Setting for Terminal Sessions

To make a locale setting persistent and apply it to newly opened terminal sessions, place the export statement in the user's shell initialization files. For more information, see "About the User's Work Environment" in *Managing User Accounts and User Environments in Oracle Solaris 11.3*.

Locale Negotiation Over Remote Connections

Applications that provide remote access such as `ssh` or `zlogin` might perform locale negotiation, which is an attempt to match current user locale setting against the potentially different set of locales available on the remote location. This process can have unexpected results, for example:

- User expects to get a shell with the remote system's default system locale but gets a different, negotiated locale.
- User expects to get a negotiated locale but because it is not available, the shell is set up with default system locale.

For more information about the locale negotiation mechanism in Secure Shell, see `ssh`(1) and `sshd`(1M) man pages.

For more information, see *Managing Secure Shell Access in Oracle Solaris 11.3* and Chapter 4, "About Non-Global Zone Login" in *Creating and Using Oracle Solaris Zones*.

Managing Default System Locale

The default system locale is the locale in which the system will boot and run. In earlier releases of Oracle Solaris, the default system locale was configured in the `/etc/default/init` file. You

can use the `nlsadm` command with the `get-system-locale`, `list-locale`, and `set-system-locale` subcommands. For more information, see the `nlsadm(1M)` man page.

To display the current default system locale, use the `get-system-locale` command,

```
# nlsadm get-system-locale
LANG=en_US.UTF-8
LC_CTYPE=
LC_NUMERIC=
LC_TIME=
LC_COLLATE=
LC_MONETARY=
LC_MESSAGES=
LC_ALL=
```

To list available system locales:

```
# nlsadm list-locale
```

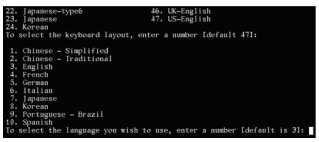
To set the default system locale, use the `set-system-locale` command. For example, to set the default locale to `fr_FR.UTF-8`:

```
# nlsadm set-system-locale fr_FR.UTF-8
```

Locale Selection in the Installer

The initial default system locale is set during the installation. The Oracle Solaris installer is localized, so speakers of any of the core languages can navigate through the installation using their native tongue. The second screen of the Oracle Solaris 11 Live CD enables the user to choose the language that will be used during the installation. Only the ten core languages are available.

FIGURE 1 Language Choices in the Installer



This language selection also determines the default language support and other data formats for the installed system.

Regardless of the default setting, all ten core locales are installed and available to users. You can change the default through the Automated Installer to broaden or narrow the language scope. Alternatively a custom Live CD can be generated by using the `distro_const` command. For more information, see the `distro_const`(1M) man page.

Note - Localization for the text installer is available only through the serial console and not through the physical console.

Locale Selection in the Automated Installer

This section describes possible adjustments to Automated Installer manifests that alter the locale, keyboard, and timezone preferences.

Selecting Locales to Be Installed

The selection of locales to be installed is controlled by facets in the AI manifest. The <software><image> element contains the element `facet`, which has the following syntax:

```
<facet set="true|false">facet-name</facet>
```

The following sample manifest uses the `facet` elements to make sure that only the German (in Germany) and English (in the United States) locales and translations are installed on the target system.

```
<!DOCTYPE auto_install SYSTEM "file:///usr/share/install/ai.dtd.1">
<auto_install>
 <ai_instance auto_reboot="true" name="ai-german">
...
 <software type="IPS">
  <destination>
   <image>
    <!-- deselect all locales -->
    <facet set="false">facet.locale.*</facet>
    <!-- specify specific locales to install -->
    <!-- install German and English only -->
    <facet set="true">facet.locale.de</facet>
    <facet set="true">facet.locale.de_DE</facet>
    <facet set="true">facet.locale.en</facet>
    <facet set="true">facet.locale.en_US</facet>
   </image>
  </destination>
...
 </software>
 </ai_instance>
```

```
</auto_install>
```

You can obtain a list of available locale facets from the `system/locale` package by issuing a command similar to the following example:

```
$ pkg contents -m system/locale | /usr/gnu/bin/grep -o facet.locale.[^\ ]* | sort -u
```

For more information about manifest creation, see the `ai_manifest`(4) man page.

Installing Non `UTF-8` Locales

Although non `UTF-8` (legacy) locales are not part of the default installation, they are still available in the `system/locale/extra` package. For example, to install the `de_DE.ISO8859-1` locale, the package `system/locale/extra` has to be added to the AI manifest too.

```
...
 <software_data action="install">
  <name>pkg:/entire@release</name>
  <name>pkg:/group/system/solaris-desktop</name>
  <name>pkg:/system/locale/extra</name>
 </software_data>
...
```

Setting the Default System Locale, Keymap, and Timezone

You specify the default locale of a system installed through the Automated Installer in the System Configuration (SC) profile, as described in the `service_bundle`(4) and `installadm`(1M) man pages. For more information, see "Providing Configuration Profiles" in *Installing Oracle Solaris 11.3 Systems*.

The following example shows a system configuration profile that sets the default system locale to German.

```
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type="profile" name="system configuration">
...
 <service name='system/environment' version='1'>
  <instance name='init' enabled='true'>
   <property_group name='environment'>
    <propval name='LANG' value='de_DE.UTF-8'/>
   </property_group>
  </instance>
 </service>
...
```

```
</service_bundle>
```

Other international environment settings that were transitioned to the Service Management Facility (SMF) can be set in a similar fashion. The following example shows how to use the UK-English keyboard layout and the GMT timezone.

```
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type="profile" name="system configuration">
...
 <service name='system/keymap' version='1'>
  <instance name='default' enabled='true'>
   <property_group name='keymap'>
    <propval name='layout' value='UK-English'/>
   </property_group>
  </instance>
 </service>

 <service name='system/timezone' version='1'>
  <instance name='init' enabled='true'>
   <property_group name='timezone'>
    <propval name='localtime' value='GMT'/>
   </property_group>
  </instance>
 </service>
...
</service_bundle>
```

Managing Available Locales

In a default installation of Oracle Solaris, the following core locales are available, with only one locale per language:

- Chinese - Simplified (`zh_CN.UTF-8`)
- Chinese - Traditional (`zh_TW.UTF-8`)
- English (`en_US.UTF-8`)
- French (`fr_FR.UTF-8`)
- German (`de_DE.UTF-8`)
- Italian (`it_IT.UTF-8`)
- Japanese (`ja_JP.UTF-8`)
- Korean (`ko_KR.UTF-8`)
- Portuguese - Brazilian (`pt_BR.UTF-8`)
- Spanish (`es_ES.UTF-8`)

This section describes the different ways to add support for additional languages.

Adding or Removing Locales by Using `nlsadm`

You can use the `install-locale`, `list-locale`, and `uninstall-locale` subcommands of the `nlsadm` command to administer locales and manage national language properties. For more information, see the `nlsadm`(1M) man page.

To install a locale and any available translations, use the `install-locale` subcommand. For example, to install the Danish locale, you would use the following command:

```
# nlsadm install-locale da_DK.UTF-8
```

To list the locales available for installation based on the current image, use the `list-locale` command.

```
# nlsadm list-locale
LOCALE            LANG  TERRITORY  CODESET  MODIFIER  FLAGS
af_ZA.UTF-8        af    ZA         UTF-8    -         -
...
```

On a default installation, this command lists only the locales available from the installed packages. More locales are available in the package repository, which you can display by using the `-a` option.

```
# nlsadm list-locale -a
Reading package information from IPS publisher's repository (it could take couple of
 minutes) ...
LOCALE             LANG  TERRITORY  CODESET  MODIFIER  FLAGS
af_ZA.UTF-8         af    ZA         UTF-8      -         -
...
```

To uninstall a locale (Danish) and all related translations, use the following command:

```
# nlsadm uninstall-locale da_DK.UTF-8
```

Note - Due to the structure of packaging, sometimes the requested locale changes cannot be made and additional locales will be installed or removed as a side-effect. In such cases, `nlsadm` command will provide a solution with minimal impact and inform you about the additional changes needed before committing them.

Locale Facets

To understand the packaging of language support related components in Oracle Solaris 11, you must be familiar with the concept of locale facets.

What Is a Facet?

In earlier releases of Oracle Solaris, the optional components such as documentation, localization, or debug files were split into separate packages. The Image Packaging System in Oracle Solaris 11 allows Oracle to keep the optional components in the same package by using special tags called *facets*. Facets make the packaging simpler, while keeping disk space usage low if you do not need the additional features. For more information about facets, see "Package Facets and Variants" in *Adding and Updating Software in Oracle Solaris 11.3* and Chapter 5, "Allowing Variations" in *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.3*.

The locale facets are used to mark files or actions that are language or locale specific. For example, in the manifest of the `web/wget` package, the file `/usr/share/locale/ja/LC_MESSAGES/wget.mo` is tagged with `locale.ja=`*true*. This tag indicates that the file, which contains Japanese translations of the `wget` messages, will be installed only when support for Japanese is enabled by setting the `locale.ja` facet to `true`.

Structure of Locale Facets

There is no fixed format for the locale facets. The following convention is used in the Oracle IPS repositories:

`locale.{`*language*`}[`*_territory*`]`

language is a two-letter language code from the ISO 639 standard, and *territory* is a two-letter territory code from ISO 3166.

Adding or Removing Locales by Using `pkg`

To add language support, set the pertinent locale facet with the `pkg change-facet` command by using the `pkg` command. For example, you can use the following command to install and add support for French as spoken in France.

- To install files common for all French locales

```
# pkg change-facet locale.fr=True
```

- To install files specific to French in France

```
# pkg change-facet locale.fr_FR=True
```

You would use the following commands to add support to all available French variants.

- To install files common for all French locales

```
# pkg change-facet locale.fr=True
```

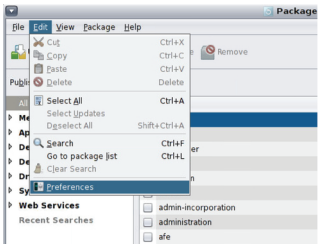
- To install all files specific to all of the French variants

```
# pkg change-facet locale.fr_*=True
```

How to Add Language Support in Package Manager

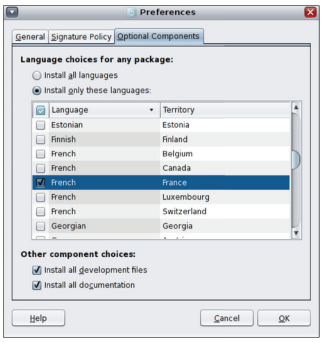
The Package Manager manages only languages present in the `system/locale` package. To add or remove support for any other language, use the command line.

1. In the Package Manager window, choose Edit → Preferences.



2. **In the Preferences window, select the Optional Components tab.**

3. **Select or deselect the desired language.**



4. **Click OK.**

All the optional components of the installed packages will be updated automatically.

Legacy Locales

Although the default system locales in Oracle Solaris use `UTF-8` encoding, a number of legacy locales can be used as well. You can install or remove legacy locales by using the `nlsadm` command, as described in "Adding or Removing Locales by Using `nlsadm`" on page 37.

Note - The GNOME desktop environment supports only `UTF-8` locales.

Legacy locale support is contained in the `system/locale/extra` package. The `nlsadm` command installs this package automatically when it is needed. When adding or removing locales by using `pkg` facet modification, install the package by using the following command:

```
# pkg install pkg:/system/locale/extra
```

To enable support for a specific language, set the corresponding locale facet to `true`. For example, to install the `da_DK.ISO8859-1` locale and all of the `da_DK` locales, you would use the following commands:

```
# pkg install pkg:/system/locale/extra

# pkg change-facet locale.da=True

# pkg change-facet locale.da_DK=True
```

Locale Aliasing

While the most common locales are usually well supported across the major operating systems, their names are different in many cases. For example, although Oracle Solaris uses `fr_FR.UTF-8` as the locale name for French as spoken in France using `UTF-8` encoding, IBM AIX uses `FR_FR`, and HP-UX 11.11 and RHEL 5.4 use `fr_FR.utf8`. This inconsistency can be burdensome in a heterogeneous environment or when migrating to Oracle Solaris.

To address this issue, support for locale aliases was introduced in `libc` in Oracle Solaris 11. Locale name aliases are accepted and mapped to corresponding canonical locale names, if any, during the locale selection process, as specified in the `setlocale`(3C) man page, and message object or message catalog processing, as specified in the `gettext`(1), `catopen`(3C), and `gettext`(3C) man pages.

In addition, to provide better compatibility with earlier Oracle Solaris releases, the messaging functions now look for the message object or catalog using the obsolete Solaris locale names,

such as `fr` or `fr_FR`, as additional locale names to check against. For more information, see "Short Form Locales" in the "Localization" section on the Oracle Solaris 11 - End of Feature Notices page.

For example, typical use case is a predominantly Linux environment where the Linux style locale name is used in the locale announcement in the user's shell initialization file. For example, the command `setenv LANG ja_JP.utf8` is included in `$HOME/.login`, and the home directory is NFS-mounted. In this network environment, when a user logs into an Oracle Solaris 11 system, the locale alias support mechanism will internally and transparently map the locale name into the corresponding Oracle Solaris locale name, which is `ja_JP.UTF-8`, and will honor and support the user-specified locale name. In the same way, when non-Solaris locale names are passed to a remote Oracle Solaris 11 system through `ssh(1)`, they will be recognized, honored, and supported.

For more information about local aliasing, see the `locale_alias(5)` man page, which also has full lists of locale name mappings.

♦ ♦ ♦ CHAPTER 4

Desktop Keyboard Preferences and Input Methods

There are many different options for configuring the keyboard in Oracle Solaris 11 desktop. However most desktop users would need to do only minimal configuration in order to correctly set the keyboard. This chapter describes the Keyboard preferences and Input Methods, and the process to correctly configure the keyboard according to your needs.

About Keyboard Layout and Input Method Settings

You can configure Keyboard Layout preferences such as keyboard layout, keyboard model, and so on. The two main areas are:

- Input Methods (IM) – Choose this method if you regularly use languages that have large number of characters or complex characters, for example Asian languages like Chinese, Japanese or Korean.
- GNOME Keyboard Layout Preferences – Choose this method if you regularly use languages that have smaller number of characters, for example languages using Latin characters as English, Spanish, French, German, etc.

Although the IM provides a more complex mechanism for composing characters, it can be used also for Latin languages. Similarly, GNOME Keyboard Layout Preferences can be also used for non-Latin languages.

Note - Although you can set a variety of Keyboard Layout preferences, you don't need to configure all areas related to the keyboard. For example, if you use only one or two Latin keyboard layouts (For example, US/English and French), rather than activating IM. You can just set keyboard layouts in GNOME Keyboard Preferences. On the other hand, if you use Chinese or Japanese keyboard layouts, IM would probably be more suitable.

Keyboard Layout Selection in GNOME Desktop Manager

Oracle Solaris 11 provides Keyboard Selection support in Gnome Desktop Manager (GDM).

▼ How to Select the Keyboard Layout in GNOME Desktop Manager

Keyboard layout for the GNOME session can be selected in GDM, however it is not mandatory. If you do not select any keyboard layout, the default keyboard layout is selected. You can select a different layout later in the GNOME session or the next time you log in.

1. **You type the user name.**

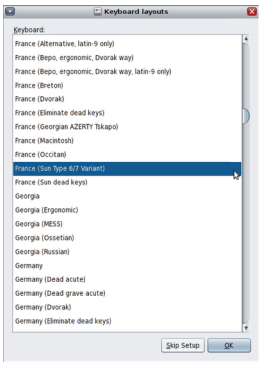
2. **Click the Log In button or presses Return.**

   The "Keyboard Layout Selection" panel becomes visible. This panel is located at the bottom of the GDM window, as shown in the following figure.

   The default value of the keyboard layout is set to USA.

3. **To view a list of all the available keyboard layouts, choose the Other.**

A window containing a list of all the available keyboard layout variants appears as shown in the following figure.



4. **Select the desired layout and click OK.**

Selecting Keyboard Layout in GDM

If a keyboard layout is selected in GDM, the value of the selected keyboard layout is stored in the file `$HOME/.dmrc` and it will be preselected on the next login.

Note - The keyboard layout selection in GDM works only when both client and server are on the same physical machine, it does not work for remote connections. When GDM is accessed remotely, the default value of the keyboard layout is set to USA regardless of the contents of the file `$HOME/.dmrc`.

Input Methods

Input Method (IM) is a mechanism to input specific characters which are not provided on input devices like a keyboard to the various desktop applications. IM is required for some languages such as Chinese, Indic, Japanese, Korean, and Thai because these languages contain a much larger set of characters than what is available on the input devices. IM converts combinations of keystrokes from input devices to language-specific characters, and sends the information back to the focused application.

Input Method has two main components, the IM Framework and the IM Language Engine. The IM Framework is a software component providing functions that enable cooperation between the IM Language Engine and user applications. The IM Language Engine is a software

component which takes keystroke combinations from the IM Framework and converts them to specific language characters to send them back to the IM Framework.

IBus is the default IM Framework on the Oracle Solaris desktop system. *IIIMF* is available in the installation repository as a secondary IM Framework.

By default, Input Method is activated only when you log in into the following languages:

- Chinese (Simplified)
- Chinese (Traditional)
- Japanese
- Korean
- Indic
- Thai

For other languages, if necessary, IM can be activated manually through the Input Method Framework Selector.

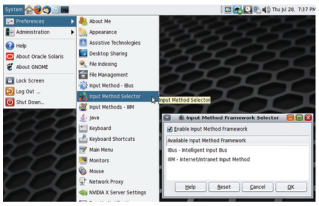
How to Activate and Deactivate Input Methods

The Input Method Framework Selector (`imf-selector`) is a configuration tool used for selecting the preferred Input Method Framework and enabling or disabling them for the desktop session.

1. **Choose System → Preferences → Input Method Selector.**

   The Input Method Framework Selector window appears.

2. **Select the Enable Input Method Framework option and select the preferred Input Method Framework from the list**



3. **Click the OK button to save selection.**

About IBus

Intelligent Input Bus (IBus) for the Linux and Unix operating systems is a powerful multilingual Input Method Framework working with many open-source IM language engines. IBus uses bus-like architecture to process communication between the IBus IM Framework and the IM language engines. This process runs per user session. There is no shared process between different user desktop sessions.

IBus Configuration

IBus is configured per user by the IBus Preference tool (`ibus-setup`). To access this tool, choose System→ Preferences→ Input Method - IBus. The following tasks describe some major configuration changes you can make.

▼ How to Add New Language Engine

1. **Click the Input Method tab in the IBus Preferences window.**
2. **Select the preferred IM language engine.**
3. **Click Add.**

▼ How to Add New Input Method Trigger Key

1. **Click the General tab in IBus Preferences window.**
2. **Click the ... button on the Enable or disable label. Then click ... button on Key code label and information dialogue pops up. Now press the preferred key combination for trigger key and click Close button on information dialogue. Then press Add and OK button on Select Keyboard shortcut for trigger window.**
3. **Trigger key is used to activate IBus on focused application. If IBus is not activated, characters from input devices are sent straight to the focused application. The default trigger keys are shown in the Enable or disable label.**

▼ How to Configure the Language Panel Location

1. **In the IBus Preference tool, select a configuration from the Show Language Panel menu.**

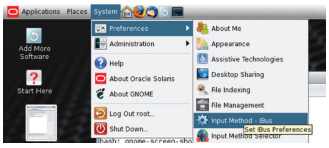
There are three possible locations for language panel position.

- Embedded in menu (default) - Language panel is embedded in notification area of GNOME panel
- When active - Language panel is shown as independent window only when Input Method is activated
- Always - Language panel is always shown as independent window.

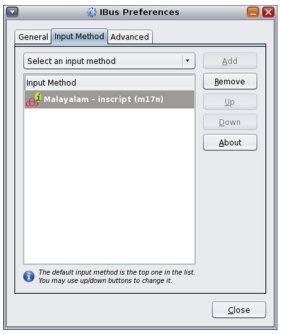
2. **Click OK.**

▼ How to Add a New Language Engine in IBus

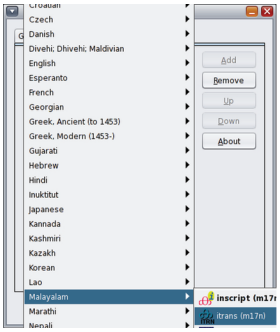
1. **Choose System → Preferences → Input methods – IBus.**



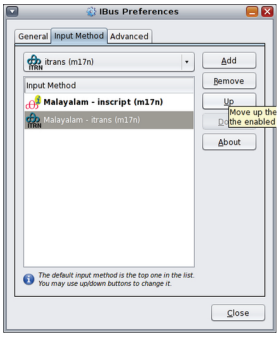
2. **IBus Preferences window opens, click on the Input Method tab**



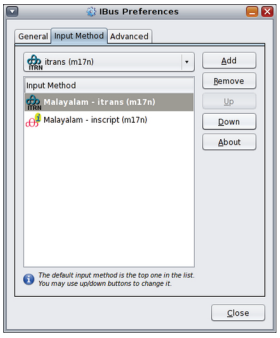
3. **Select the Select Input Method and select the desired input method.**



4. **Click the Add button to add it to the selected list.**



5. **To make the new input method the default, click the Up button. Move the selection to the top of the list.**



6. **Log out and log back in to use the currently selected input method as the default.**

Language Engines Available on the IBus Input Method Framework

The following Language Engines are available on the IBus Input Method Framework.

- Installed by default during installation process for the desktop
  - Anthy - Japanese
  - Chewing - Traditional Chinese
  - Hangul - Korean
  - IBus-Sayura -- Sinhala
  - IBus-XKBC - Keyboard Emulation IM Engine framework
  - SunPinyin - Simplified Chinese
  - Various Language Engines based on multilingualization.
- Optional Language Engines
  - Pinyin - Simplified Chinese Language Engine
  - Various Language Engines based on IBus table framework

Input Method for Indic Languages

The following table summarizes the available input methods for Indic languages.

TABLE 8 Input Method for Indic Languages

Locale	Language	Input Methods
as_IN.UTF-8	Assamese	Inscript layout, ITRANS transliteration, phonetic layout
bn_IN.UTF-8	Bengali	Inscript layout, ITRANS transliteration, probhat layout, Unijoy keyboard layout
en_IN.UTF-8	English	No input method necessary
gu_IN.UTF-8	Gujarati	Inscript layout, ITRANS transliteration, phonetic layout
hi_IN.UTF-8	Hindi	Inscript layout, ITRANS transliteration, phonetic layout, Remington typewriter layout, input method with `typewriter' method
kn_IN.UTF-8	Kannada	Inscript layout, ITRANS transliteration, KGP method

Locale	Language	Input Methods
ks_IN.UTF-8	Kashmiri	Input method simulating Kashmiri keyboard.
ml_IN.UTF-8	Malayalam	Inscript layout, ITRANS transliteration, Mozhi input method, Swanalekha input method
mr_IN.UTF-8	Marathi	Inscript layout, ITRANS transliteration, phonetic layout
or_IN.UTF-8	Oriya	Inscript layout, ITRANS transliteration, phonetic layout
pa_IN.UTF-8	Punjabi	inscript layout, ITRANS transliteration, jhelum layout, phonetic layout
sa_IN.UTF-8	Sanskrit	Input method with Harvard-Kyoto convention.
ta_IN.UTF-8	Tamil	Inscript layout, ITRANS transliteration, phonetic layout, Renganathan layout. tamil99 layout, typewriter layout, "vutam" Type_As_You_Write layout.
te_IN.UTF-8	Telugu	Apple keyboard layout for Telugu, inscript layout, ITRANS transliteration, pothana Telugu input method Version 2.0, RTS method

IBus XKBC IM Engine

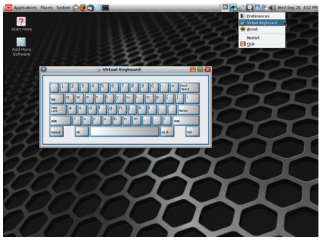
IBus XKBC emulates keyboard layout using the XKeyboard Configuration Database. All keyboard layouts available in XkeyboardConfig, including keyboard layout variants, can be emulated on the IBus IM Framework. IBus XKBC is available in the Other language category in the IBus Preference tool. The IBus XKBC help provides detailed configuration information.

IBus Virtual Keyboard (`ibus-keyboard`)

IBus Virtual Keyboard is a graphical keyboard emulator based on theIBus XKBC Language Engine. It displays a graphical keyboard in a window, and enables the user to input characters to the focused application by clicking key icons. As with IBus XKBC, the IBus Virtual Keyboard supports various keyboard layouts and variants for emulation. To launch the IBus Virtual Keyboard, choose Virtual Keyboard from the IBus menu in the GNOME panel, as shown in the following figure.

To display the configuration menu for the virtual keyboard, right-click on the Virtual Keyboard application window.

FIGURE 2 Virtual Keyboard



Internet Intranet Input Method Framework

The Internet Intranet Input Method Framework (IIIMF) is another IM Framework in Oracle Solaris 11, which has been supported since Solaris 9. IIIMF runs per user instead of as a shared system-wide process. The configuration tool for the IIIM is the Input Method Preference Editor (`iiim-properties`). To access the tool, choose System → Preferences → Input Method - IIIM.

IIIMF has two very powerful Japanese language engines, ATOK and Wnn. The following procedure describes how to use the ATOK or Wnn language engines with IIIMF.

Note - For more information about this tool, see the *ATOK for Oracle Solaris User Guide* and *Wnn8 User's Guide* provide more information for these language engines. These documents are available in Japanese only.

▼ How to Use the ATOK and Wnn Language Engines With IIIMF

1. **Install the following packages by using the Package Manager GUI or the `pkg` command:**

   - IIIMF core package: `system/input-method/iiim`
   - ATOK package: `system/input-method/iiim/atok`
   - Wnn package: `system/input-method/iiim/wnn`

   To list the IIIMF Language Engines available for installation, run the following command:

   ```
   $ pkg list -a system/input-method/iiim/*
   ```

2. **Select IIIMF in your desktop session using the Input Method Framework Selector.**

IIIMF will be used for your desktop session at the next login. (The desktop session needs to be restarted.)

3. **If you are using the Wnn language engine is used,**

   a. **Invoke the `jserver` and `dpkeyserv` servers by issuing the following command. (Service Management privilege needed.)**

      ```
      # svcadm enable wnn8/server
      ```

   b. **Select the Wnn IM Language Engine in the Input Method Preference Editor (`iiim-properties`) tool.**

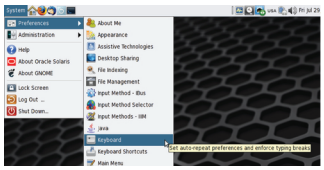
GNOME Keyboard Preferences

The Oracle Solaris 11 Desktop has more than 400 keyboard layouts available for more than 100 languages. Multiple keyboard layout variants are available for almost every language. Regardless of the keyboard layout model and physical layout, you can always configure and use any keyboard layout available in the Oracle Solaris 11 Desktop. Use the GNOME Keyboard Preferences tool to set and customize keyboard preferences such as keyboard model, layout, variant, and so on.

Note - The keyboard layout preferences window is available only when both client and server are on the same physical machine, it is not available for remote connections.

▼ How to Set Keyboard Preferences and Customization

1. **Choose System → Preferences → Keyboard.**



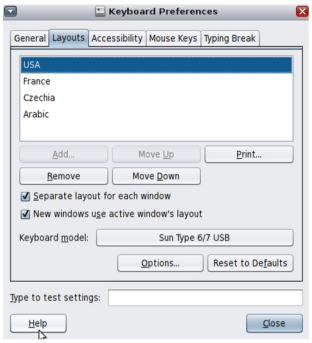
2. **Click the Layouts tab.**

The Layouts tab provides all necessary tools to set and customize the keyboard layout. You can add up to 4 different keyboard layouts and easily switch between them by using the Keyboard Layout Indicator menu in the panel or by setting a keyboard shortcut to switch between them.

In the Layouts tab you can also customize different settings of your keyboard, such as keyboard model, specific keyboard layout variants or you can set special behavior for specific keys such as **Shift**, **Alt**, **Meta**, **Super**, **Hyper**, **CapsLock**, etc.

The layout at the top of the list in the Layouts tab is the default layout. You can move layouts up and down in the list by using the Move Up and Move Down buttons.

The following illustration shows the Layouts tab.



Keyboard Layout Options and Default Behavior

The following list shows the most common transactions performed in the Layouts tab in the Keyboard preferences window:

▼ How to Add and Remove Keyboard Layouts

1. **Select the desired keyboard layout by Country or by Language.**
2. **Click the Add button.**
3. **To remove a layout, select the keyboard layout you want to delete from the list.**
4. **Click the Remove button.**

▼ How to Add EuroSign to Certain Keys

1. **Click the Options button.**

2. **Choose Adding EuroSign to Certain Keys.**
3. **Choose the desired key or keys.**
4. **Click the Close button.**

▼ How to Set Key Sequence to Kill the X Server

1. **Click the Options button.**
2. **Choose Key Sequence to Kill the X Server.**
3. **Choose Control + Alt + Backspace.**
4. **Click the Close button.**

▼ How to Set a Keyboard Shortcut to Switch Between Selected Keyboard Layouts

In case you use more than one keyboard layout, you can assign a shortcut to switch between them easily.

1. **Click the Options button.**
2. **Choose Key(s) to Change Layout.**
3. **Choose the desired key or keys for the shortcut.**
4. **Click the Close button.**

GNOME Keyboard Layout Indicator (Keyboard Selector)

Keyboard Layout Indicator is automatically activated and visible on the panel when two or more keyboard layouts are selected as shown in the following figure. When only one keyboard is selected (default behavior), the Keyboard Layout Indicator is not visible on the panel.

FIGURE 3 Keyboard Layout Indicator

When the Keyboard Layout Indicator menu is activated and visible on the panel you can switch between selected keyboard layouts by using the mouse. Single click to immediately switch to the next keyboard layout. Right-click to open a menu with three options:

- Groups – Contains a list of selected keyboard layouts.
- Keyboard Preferences – Launches the Keyboard Layout Preferences window
- Show Current Layout – Launches a window with an interactive application showing the current keyboard layout.

Keyboard Layout Settings Using the X Keyboard Extension (setxkbmap)

You can use the `setxkbmap` command to set and customize all keyboard layout settings in the X Server from the command line rather than using the GNOME Keyboard Layout Preference tool.

The `setxkbmap` command maps the keyboard to use the keyboard layout determined by the options specified on the command line. Configurable options of the `setxkbmap` command include geometry, keyboard model, layout symbols, layout variant, rules, and the like.

The following example shows how to set two keyboard layouts (US/English and French) from the command line:

```
$ /usr/bin/setxkbmap us,fr
```

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

Keyboard Layout in the Console

Although you will probably not need to do so often, you can change the keyboard layout in the console by using the `nlsadm` command with the `get-console-keymap`, `list-console-keymap`, and `set-console-keymap` subcommands. The console keyboard layouts for an Oracle Solaris console include ASCII characters only. For more information, see the `nlsadm`(1M) man page.

To view the current console keyboard layout setting:

```
# nlsadm get-console-keymap
keymap=US-English
```

To list available values for the console keyboard layout setting:

```
# nlsadm list-console-keymap
```

```
Arabic
...
```

To set the console keyboard layout to Japanese:

```
# nlsadm set-console-keymap Japanese
```

Note - The `/usr/share/lib/keytables/type_6/kbd_layouts` file on an installed system provides a list of available keyboard layouts.

For more information, see Appendix A, “Available Locales and Supported Character Sets”.

♦ ♦ ♦ CHAPTER 5

Configuring Fonts

This chapter describes the underlying library for font configuration and lists the available fonts for selected Asian languages. Font configuration for the GNOME desktop environment is described in detail in the "Font Preferences" in *Oracle Solaris 11.3 Desktop User's Guide*.

This chapter covers the following topics:

`fontconfig` Library

The Oracle Solaris desktop environment uses the `fontconfig` library for configuring and customizing font access. `fontconfig` maintains a list of all fonts available on the system, using automatic discovery based on the configuration in `fonts.conf`(4) file. When an application is searching for a specific font by name, and optionally other attributes such as size, font-weight, or language, `fontconfig` provides the application with the specific font path. `fontconfig` does not lay out or render the fonts into text.

Adjusting Font Configuration

Sometimes the default fonts in a specific locale are not satisfactory. You can alter the font priorities by modifying the `fontconfig` configuration files in the `/etc/fonts`. `fontconfig` directory. System-wide configuration is modularized into individual `xml` files in the `/etc/fonts/conf.avail` directory. The `/etc/fonts/conf.d` directory then includes symbolic links to selected configuration files in the `/etc/fonts/conf.avail` directory. You can alter the system-wide configuration by adding or removing these symbolic links.

Use the `fc-match`(1) and `fc-list`(1) commands to obtain information about the default font and other available fonts. For example, the following command will display information about

the default monospaced font for Japanese. (Monospaced fonts, also called fixed-width fonts, are typically used in text editors.)

```
$ fc-match Monospace:lang=ja
ipag.otf: "IPAGothic" "Regular"
```

You might want to disable the use of bitmap fonts when alternative TrueType fonts are available and preferred, create a symbolic link to the configuration file `70-no-bitmaps.conf`, as shown in the following example:

```
# cd /etc/fonts/conf.d
# ln -s ../conf.avail/70-no-bitmaps.conf
```

`fontconfig` monitors the configuration directory `/etc/fonts/conf.d` and will automatically adjust its in-memory configuration to include the changes. For more information, see the `fc-cache`(1) man page. Per-user configuration can also be maintained in `$HOME/.fonts.conf` for each user. `$HOME/.fonts` is scanned for users' custom font files. For more information on `fontconfig` and its configuration, see the `fonts.conf`(4), `fc-match`(1), and `fc-list`(1) man pages, or the project page at `http://www.freedesktop.org/wiki/Software/fontconfig`.

Available Fonts

This Oracle Solaris 11 release includes numerous updates to fonts available in the system and features a significant move towards using TrueType or OpenType font formats.

Many fonts come from open-source projects and are updated regularly, such as the DejaVu fonts, which are based on the well known Vera fonts. DejaVu fonts are the system's default fonts in many locales, as they are designed for use with Unicode.

Fonts are added to the system automatically upon the modification of locale facets. For more information, see “Locale Facets” on page 37. For installation of additional fonts using Package Manager, fonts are categorized in the `System/Fonts`.

Indic Fonts

Indic fonts are described in the following table.

TABLE 9 Indic Fonts

Locale	Language	Font
`as_IN.UTF-8`	Assamese	Mukti Narrow

Locale	Language	Font
bn_IN.UTF-8	Bengali	Mukti Narrow
en_IN.UTF-8	English	DejaVu Sans
gu_IN.UTF-8	Gujarati	Lohit Gujarati
hi_IN.UTF-8	Hindi	Lohit Hindi
kn_IN.UTF-8	Kannada	Lohit Kannada
ks_IN.UTF-8	Kashmiri	Lohit Kashmiri
ml_IN.UTF-8	Malayalam	Lohit Malayalam
mr_IN.UTF-8	Marathi	Lohit Marathi
or_IN.UTF-8	Oriya	Lohit Oriya
pa_IN.UTF-8	Punjabi	Lohit Punjabi
sa_IN.UTF-8	Sanskrit	Lohit Sanskrit
ta_IN.UTF-8	Tamil	Lohit Tamil
te_IN.UTF-8	Telugu	Lohit Telugu

Japanese Fonts

Japanese TrueType fonts are described in the following table.

TABLE 10 Japanese Fonts

Full Family Name	Subfamily	Format	Style	Vendor	Character Set
IPAGothic	Regular	TrueType	Sans serif, fixed width	IPA	JIS X 0201: 1976, JIS X 0213:2004
IPAPGothic	Regular	TrueType	Sans serif, proportional	IPA	JIS X 0201: 1976, JIS X 0213:2004
IPAMincho	Regular	TrueType	Serif, fixed width	IPA	JIS X 0201: 1976, JIS X 0213:2004
IPAPMincho	Regular	TrueType	Serif, proportional	IPA	JIS X 0201: 1976, JIS X 0213:2004
Sun-Gothic	Regular, Bold	PCF (12,14,16,20,24)	Sans serif, fixed width	SCR	JIS X 0201: 1976, JIS X 0208:1990
Sun-Minchou	Regular, Bold	PCF(16,20,24)	Serif, fixed	SCR	JIS X 0201: 1976, JIS X 0208:1990

Japanese fonts are packaged as follows:

- IPAGothic, IPAPGothic – `system/font/truetype/ipafont`
- IPAMincho, IPAPMincho – `system/font/truetype/ipafont-mincho`
- Sun-Gothic, Sun-Minchou – `system/font/sun-ja-bitmap-unicode` (Unicode indexed font) and `system/font/sun-ja-bitmap` (JIS indexed font)

Unicode indexed fonts are provided to use bitmap fonts in GTK applications. For example, to use bitmap fonts in a Japanese GNOME environment, install `system/font/sun-ja-bitmap-unicode`.

To use Japanese bitmap fonts used in the CDE environment in the previous release, install `system/font/sun-ja-bitmap`.

Korean Fonts

Korean TrueType fonts are described in the following table.

TABLE 11 Korean Fonts

Full Family Name	Subfamily	Format	Vendor	Character Set
RoundedGothic, Kodig, Haeseo, Myeongjo .	Regular	TrueType	HanYang Systems Inc	KSC5601-1992
SunDotumChe. SunDotum	Regular	TrueType	Oracle Corporation	KSC5601-1992
Baekmuk: Batang, Dotum, Gulim, Headline	Regular, Bold	TrueType	Baekmuk Font21 Inc.	KSC5601-1987
unfonts	Regular, Bold	TrueType	KLDP	ISO8859-1, ISO8859-5, KOI8
Unifont	Regular	TrueType	GNU Unifont	KSC5601

Korean bitmap fonts are described in the following table.

TABLE 12 Korean Bitmap Fonts

Full Family Name	Subfamily	Format	Character Set
Gothic	Regular	PCF(16)	KSC5601-1987
Mincho	Regular	PCF(16,24)	KSC5601-1987

Simplified Chinese and Traditional Chinese Fonts

Traditional Chinese and Simplified Chinese TrueType fonts are described in the following table.

TABLE 13 Simplified Chinese and Traditional Chinese Fonts

Full Family Name	Subfamily	Format	Vendor	Character Set
AR PL UMing	Regular	TrueType	Arphic	BIG5, BIG5-HKSCS, CNS11643.1992
AR PL UKai	Regular	TrueType	Arphic	BIG5, BIG5-HKSCS, CNS11643.1992
WenQuanYi Zen Hei	Regular	TrueType	WenQuanYi	BIG5, BIG5-HKSCS, CNS11643.1992
Unifont	Regular	TrueType	GNU Unifont	BIG5, BIG5-HKSCS, GB18030-2000, GB2312-1980
uming	Regular	TrueType	Arphic	GB18030-2000, GB2312-1980, GBK
ukai	Regular	TrueType	Arphic	GB18030-2000, GB2312-1980, GBK
wqy-Zenhei	Regular	TrueType	WenQuanYi	ISO8859-1, ISO8859-5
Droid-Sans	Regular	TrueType	Google Corporation	BIG5, GB2312-1980, KSC5601, JIS0208

Traditional Chinese and Simplified Chinese bitmap fonts are described in the following table.

TABLE 14 Traditional Chinese and Simplified Chinese Bitmap Fonts

Full Family Name	Subfamily	Format	Character Set
Song	Regular	PCF(16,24)	GB2312-1980
FangSong	Regular	PCF16	GB2312-1980
Unifont	Regular	PCF16	BIG5, BIG5-HKSCS, GB18030-2000, GB2312-1980

Thai Fonts

Thai TrueType fonts are described in the following table.

TABLE 15 Thai Fonts

Full Family Name	Subfamily	Format	Vendor	Character Set
Garuda	Book, Bold, Oblique, BoldOblique	TrueType	NECTEC/TLGW	ASCII, ISO8859-1, TIS620

Full Family Name	Subfamily	Format	Vendor	Character Set
Kinnari	Medium, Oblique, Italic, Bold, BoldOblique, BoldItalic	TrueType	National Electronics and Computer Technology Center	ASCII, ISO8859-1, TIS620
Loma	Book, Oblique, Bold, BoldOblique	TrueType	NECTEC	ASCII, ISO8859-1, TIS620
Norasi	Regular, Oblique, Italic, Bold, BoldOblique, BoldItalic	TrueType	Yannis Haralambous, Virach Sornlertlamvanich and Anutara Tantraporn	ASCII, ISO8859-1, TIS620
Purisa	Medium	TrueType	Poonlap Veerathanabutr	ASCII, ISO8859-1, TIS620
Sawasdee	Regular, Oblique, Bold, BoldOblique	TrueType	Pol Udomwittayanukul	ASCII, ISO8859-1, TIS620, MS-cp1252
TlwgMono	Medium, Oblique, Bold, BoldOblique	TrueType	Poonlap Veerathanabutr	ASCII, ISO8859-1, TIS620
TlwgTypewriter	Medium, Oblique, Bold, BoldOblique	TrueType	Poonlap Veerathanabutr	ASCII, ISO8859-1, TIS620
TlwgTypist	Medium, Oblique, Bold, BoldOblique	TrueType	Theppitak Karoonboonyanan	ASCII, ISO8859-1, TIS620
Umpush	Book, Oblique, Light, LightOblique, Bold, BoldOblique	TrueType	Widhaya Trisarnwadhana	ISO8859-1, TIS620
Waree	Book, Oblique, Bold, BoldOblique	TrueType	Bitstream	ASCII, ISO8859-1, TIS620, MS-cp1252
Unifont	Regular	TrueType	GNU Unifont	TIS620, ISO8859-1

♦ ♦ ♦ CHAPTER 6

Advanced Topics

This chapter describes few of the selected, more advanced topics related to code sets, printing, interoperability and locale creation. This chapter covers the following topics:

Code Set Conversion

Support for code set conversion, or character set (`charset`) conversion, is an essential part of the operating system, as most of the applications rely on this capability to function properly.

The current release of Oracle Solaris also includes the International Components for Unicode (ICU), a widely used library and tools for Unicode support, software internationalization and software globalization.

Oracle Solaris 11 includes various tools and libraries for code set conversion. The core code set conversion utility, `iconv`, is built around the `iconv` library in Oracle Solaris `libc`.

`iconv` Utility

The `iconv(1)` command-line utility converts characters or sequences of characters from one code set to another. It supports a wide range of code sets. Because code set names often differ among platforms, many of the code sets are supported under multiple names thanks to an aliasing mechanism in `iconv`. Run the following command to obtain the list of code sets currently available in a system:

```
$ /usr/bin/iconv -l
```

Because multiple packages have `iconv` modules, you can extend the default list by installing additional packages. The default installation includes the `system/library/iconv/utf-8` package, which covers the basic set of `iconv` modules for conversions among `UTF-8` and other Unicode code sets and selected other code sets. Other packages are available in the System/Internationalization category in the Package Manager, or by using the `system/library/iconv/*` name pattern for installation with the `pkg(1)` command.

The `iconv –f` option defines the source code set and the `-t` option defines the target code set. You can use `iconv` to convert a file, or standard input, to standard output as follows:

```
$ /usr/bin/iconv -f eucJP -t UTF-8 file.txt
```

This example would convert *file.txt* filename from the `eucJP` code set (Extended UNIX Code Packed Format for Japanese) and write the result in `UTF-8` to standard output.

In Oracle Solaris 11, `iconv` has been extended to include flags that modify the behavior of the conversion in these special situations:

- Illegal character – The input character is not valid in the declared source code set
- Non-identical character – There is no matching character in the target code set

Flags like `//ILLEGAL_DISCARD`, `//NON_IDENTICAL_DISCARD`, `//IGNORE` and `//TRANSLIT` can also be used at the command line. For more information, see the `iconv_open(3C)` man page.

Note - Some of the `iconv` modules in Oracle Solaris might implement only a subset of the flags described in the `iconv_open(3C)` man page.

For more information on `iconv`, see the `iconv(1)`, `iconv(3C)`, `iconv_open(3C)`, and related man pages.

International Components for Unicode

Oracle Solaris 11 adds the International Components for Unicode (ICU) C/C++ libraries to the available interfaces. ICU is a mature, widely used set of libraries providing Unicode and globalization support for software applications. ICU is portable and gives applications the same results on all platforms and between C/C++ and Java software.

Some of the services provided by ICU include:

- Code Page Conversion – Convert text data to or from Unicode and nearly any other character set or encoding.
- Collation – Compare strings according to the conventions and standards of a particular language, region, or country.

- Formatting – Format numbers, dates, times, and currency amounts according to chosen locale.
- Time Calculations – Multiple types of calendars and a thorough set of timezone calculation APIs are provided.
- Unicode Support – ICU closely tracks the Unicode standard, providing easy access to all of the many Unicode character properties, Unicode normalization, case folding, and other fundamental operations as specified by the Unicode Standard.
- Regular Expression – ICU regular expressions fully support Unicode while providing very competitive performance.
- Bidirectional text (Bidi) – Support for handling text containing a mixture of left-to-right and right-to-left data.
- Text Boundaries – Locate the positions of words, sentences, and paragraphs within a range of text, or identify locations that would be suitable for line wrapping when displaying the text.

ICU on Oracle Solaris 11 is split into two packages: `library/icu` contains just the libraries, while `developer/icu` delivers header files and several utilities like `uconv`(1).

For more information, see the project's web site at `http://site.icu-project.org`. The `libicui18n`(3LIB), `libicuio`(3LIB), `libicudata`(3LIB), `libicule`(3LIB), `libiculx`(3LIB), `libicutu`(3LIB), and `libicuuc`(3LIB) man pages document how to use the libraries in Oracle Solaris.

`uconv` Utility

In addition to `iconv(1)`, the `uconv`(1) command that is a part of the International Components for Unicode (ICU) toolset can also be used to convert text from one encoding to another. `uconv` supports 229 encodings along with more than 1000 aliases.

The tool is a part of the `developer/icu` package that is not installed by default. To install it, issue the following command:

```
# pkg install developer/icu
```

To convert a text in the `cp-1252` encoding to `UTF-8`, you would type:

```
$ uconv -f cp1252 -t UTF-8 -o file_in_utf8.txt file_in_cp1252_encoding.txt
```

Another feature of `uconv` is transliteration - conversion of letters from one script to another without translating the underlying words. The following example converts a piece of Greek text to Latin characters:

```
$ echo “Σολαρις”| uconv -x Greek-Latin -f utf-8 -t utf-8
Solaris
```

For more information about this tool's features, see the uconv(1) man page.

File Examiner (fsexam)

The File Encoding Examiner fsexam utility enables you to convert the name of a file, or the contents of a plain text file, from a legacy character encoding to UTF-8 encoding. The fsexam utility includes the following new features:

- Encoding list customization
- Encoding auto-detection
- Support for dry runs, log files, batch conversion, file filtering, symbolic files, command line, and special file types like compressed files

To add fsexam to your system install the storage/fsexam package. For more information, see the fsexam(1) and fsexam(4) man pages.

Auto Encoding Finder (auto_ef)

Oracle Solaris includes auto_ef(1), a command-line utility to identify the encoding of a file. auto_ef judges the encoding by using the iconv code conversion, determining whether a certain code conversion was successful with the file. It also performs frequency analysis on the character sequences that appear in the file. For example,

```
$ auto_ef test_file
eucJP
```

With the -a option, it displays all possible encodings for the given file:

```
$ auto_ef -a test_file
eucJP           0.89
zh_CN.euc       0.40
ko_KR.euc       0.01
```

To add auto_ef to your system install the text/auto_ef package. For more information, see the auto_ef(1) man page.

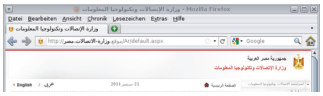
Internationalized Domain Name Support

Internationalized Domain Name (IDN) enables the use of non-English native language names as host and domain names. To use non-English host and domain names, convert these names

into ASCII Compatible Encoding (ACE) encoded names before sending the names to resolver routines as specified in RFC 5890. System administrators are also required to use ACE names in system files and applications where the system administration applications do not support the IDNs.

For more information, see RFC 5890 Internationalizing Domain Names in Applications (IDNA).

FIGURE 4 Example of IDN in Firefox Browser



Oracle Solaris 11 provides two sets of IDN implementations, including libraries and associated utilities: the GNU IDN library and the JPRS (Japan Registry Services co., Ltd.) `idnkit-2` library. Some applications might also have their own IDN implementation. For example, Firefox and Thunderbird have their own IDN service in the networking protocol component called Necko.

GNU IDN Library

`GNU- libidn` is a GNU project, licensed with the GNU Lesser General Public License (LGPL) Version 2.1 or later. `GNU- libidn` is widely adopted by various GNU/Linux distributions. Therefore, Desktop and GNOME applications such as `pidgin`(1) usually leverage `GNU- libidn` for IDN support.

`idn`(1) is the command line interface to the internationalized domain name library. The following example converts the host name in `UTF-8` into ACE encoding. The resulting URL `http://xn-fsqu00a.xn--0zwm56d` can then be used as ACE-encoded equivalent of `http://`例子.测试.

```
$ idn --quiet -a 例子.测试
xn-fsqu00a.xn-0zwm56d
```

`GNU- libidn` is available for installation as the `library/libidn` package. For more information about options, see the `idn`(1) man page.

Note - The current version of `GNU-libidn` shipped in Oracle Solaris 11 only supports IDNA2003 standards.

JPRS idnkit-2 Library

The idnkit-2 library is an open-source IDN implementation with idnkit-2 JPRS Public License. The dedicated idnkit-2 conversion utility *idnconv(1)* provides IDN conversions with various options. For more information on the options to control the conversion details, see the *idnconv(1)* man page.

Oracle Solaris 11 also supports IDN conversions through the iconv(3C) interface by leveraging the conversion routines in libidnkit (3). The iconv(1) utility can also be used for the conversions between ACE and UTF-8, as shown in following table.

Since the IDNA2008 explicitly defines terminologies for two operational modes, lookup and registration, we will also supply corresponding iconv code conversion name aliases, IDNA2008-LOOKUP (an alias to ACE-ALLOW-UNASSIGNED) and IDNA2008-REGIST (an alias to ACE).

TABLE 16 iconv IDN Code Conversions

From Code	To Code
ACE or IDNA2008-REGIST	UTF-8
ACE-ALLOW-UNASSIGNED or IDNA2008-LOOKUP	UTF-8
UTF-8	ACE or IDNA2008-REGIST
UTF-8	ACE-ALLOW-UNASSIGNED or IDNA2008-LOOKUP

The ACE and the ACE-ALLOW-UNASSIGNED iconv code conversion names (and their aliases) have the following meanings:

- ACE or IDNA2008-REGIST

  ACE is a *fromcode* or *tocode* name that can be used in iconv code conversions to refer to the ASCII Compatible Encoding defined in RFC 5890. This conversion uses STD3 ASCII rules. Unassigned characters are not allowed. ACE is typically used to store or set host or domain names.

- ACE-ALLOW-UNASSIGNED or IDNA2008-LOOKUP

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

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

```
$ iconv -f ACE -t UTF-8 hostnames.txt
```

For information about idnkit -2 library and iconv code conversions, see the libidnkit(3) and iconv_en_US.UTF-8(5) man pages.

Printing Enhancement

In Oracle Solaris 11, the legacy LP print service has been replaced by the Common UNIX Printing System (CUPS).

CUPS is a modular, open-source printing system that uses the Internet Printing Protocol (IPP) as the basis for managing printers, print requests, and print queues. CUPS supports network printer browsing and PostScript Printer Description-based printing options. CUPS also provides a common printing interface across a local network.

For more information about CUPS and its configuration, see Chapter 1, "Setting Up and Administering Printers by Using CUPS (Overview)" in *Configuring and Managing Printing in Oracle Solaris 11.3*.

This section describes print enhancement with the `mp` command-line utility.

`mp` utility

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

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

To use the utility, type the following command:

```
$ mp filename | lp
```

Because `mp` accepts `stdin` stream, you can also use the utility as a filter.

```
$ cat filename | mp | lp
```

The `mp` utility is available from the `print/mp` package and depends on select font packages that might not be installed by default. If error messages about missing fonts appear, you can find and install the right package by using the `pkg` commands.

- To find the package with the missing font:

```
# pkg search -rp TlwgTypist.ttf
PACKAGE
```

```
pkg:/system/font/truetype/thai-scalable@...
```

- To install the missing package:

```
# pkg install system/font/truetype/thai-scalable
```

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

Interoperability with Other Platforms

The following sections describe certain considerations for multi-platform environments.

NFS Server Considerations

The NFS version 4 protocol (the default in Oracle Solaris) uses UTF-8 to handle file names and other strings. Therefore, so in most use cases no charset-related adjustments should be necessary. However, note that the charset option can be used if some or all NFS clients are using a specified character set.

For example, to share the /export directory using the ISO8859-1 character set, the following command would be used:

```
# share -o iso8859-1 /export
```

To share a directory using a specific character set for some systems only, the charset=access_list option can be used:

```
# share -o iso-8859-1=isosystem.example.com,koi8-r=koisystem.example.com /export
```

All file and path names created by the clients will be converted to UTF-8 at the server.

For more information, see the share_nfs(1M) man page.

File System Considerations

mount_pcfs(1M) does not support the MS-DOS codepages, so non-ASCII characters on FAT filesystems created by MSDOS, ancient version of MS Windows or the Linux "msdos" driver may be garbled. The later FAT implementations use Unicode for character representation and it's fully supported on Oracle Solaris by default, both for reading and writing.

Archives Containing Non-ASCII Filenames

Archiving files with non-ASCII characters in filenames may cause issues, because support of non-ASCII filenames in the numerous implementations of the particular archive formats differs significantly, although the situation is improving.

Recent tar implementations on UNIX and Unix-like systems support the `POSIX` format specified by `POSIX.1-2001`, so the non-ASCII filenames are handled safely. On the MS Windows platform a number of archival utilities stores the filenames using the current codepage so names of files extracted from such archives can become garbled.

In that case the `convmv`(1) tool can be used to repair them, when the codepage is known:

```
$ convmv -f cp437 -t utf8 my_extracted_filename
```

In Zip files, the original specification sets the encoding of file names and file comments to IBM437. In 2007 PKWare extended the specification to also allow `UTF-8`. In the meantime various zip implementations adopted the strategy of using the current codepage as the filename encoding (usually on the MS Windows platform).

Info-ZIP's Zip 3.0, used in Oracle Solaris 10 and Oracle Solaris 11, stores filenames in `UTF-8`, so if both the compression and decompression utility are of this version, the archive contents would not become corrupted.

When a zip archive using a non-`UTF-8` encoding to store the file names is extracted on Oracle Solaris, the file names might get garbled. You can use the `convmv`(1) tool to repair them, if the codepage is known:

```
$ convmv -f cp437 -t utf8 my-unzipped-filename
```

Configuring National Language Properties

You can list and set the national language properties by using the `nlsadm` command. National language properties include system properties such as, system locale, console keymap and so on. These properties are specific to a locale. Setting these properties enables applications to work properly in different locales. The `nlsadm` command provides a consolidated and convenient way to administer national language properties.

The `nlsadm` command helps you to perform the following tasks:

- Set a system locale, timezone, or console keymap in the system.
- List the values of system locale, timezone, or console keymap set in the system.

- List information about a concrete locale such as, territory, state, language, timezone, codeset, and modifier.
- Set territory and language information for any locale.
- List all timezones, console keymaps, installed locales and locales available for installation from the IPS repositories.

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

The `nlsadm` command uses the `locale_description` file to get the locale information. The `locale_description` file contains national language properties in a key-value pair format. If you create a new locale and want `nlsadm` to list information about this new locale, you must specify the `locale_description` file. For more information, see the `locale_description`(4) man page.

Note - The `nlsadm` command does not depend on the `locale_description` file. If the `locale_description` file does not exist, the `nlsadm` command gets the information from the name of the locale or from similar locales.

Creating a Custom Locale

This section describes how to create a custom locale based on an existing locale delivered with the system using `localedef`(1).

Note - Only the locales provided by Oracle are supported. A badly formed locale can be a source of failures.

Creating a New Locale Based on a System Locale

To customize and thus create a new locale from existing locales, you need to prepare at least three locale definition source files:

- `localedef` source file – Contains necessary definitions for the locale.
- `charmap` source file – Contains mappings between code point values and human-readable symbolic names. The symbolic names are used in the `localedef` source file. The `charmap` source file also contains other definitions, such as the codeset name of the locale, the maximum number of bytes that can be represented in a locale code point, and so on.
- `extension` source file – Contains mappings of standard interfaces such as `strcoll`(3C) and `fgetwc`(3C) to internal locale methods, and other information needed for the proper operation of the locale.

The locales provided by the system have their respective locale definition data files available in `source/locale/localedef` package. These can be useful when only a slight change in existing system locale is required.

▼ How to Create a Custom Locale

1. **Install the necessary packages by typing the following command:**

```
$ sudo pkg install system/header source/locale/localedef
```

2. **Ensure you have the Oracle Solaris Studio C compiler in your `PATH`:**

```
$ export PATH=<oracle-studio-path>/bin:$PATH
```

3. **Prepare the workspace by using the following commands:**

```
$ mkdir mynewlocale
$ cd mynewlocale
$ mkdir amd64
$ cp /usr/lib/locale/common/methods_unicode.so.3 .
$ cp /usr/lib/locale/common/amd64/methods_unicode.so.3 amd64/
$ cp /usr/lib/localedef/src/charmaps/UTF-8.charmap \
/usr/lib/localedef/src/extensions/UTF-8.x \
/usr/lib/localedef/src/locales/fr_FR.UTF-8.src .
```

4. **Make changes to the `localedef`, `charmap`, or `extension` source files as necessary.**

5. **Build the 64-bit locale object.**

```
$ localedef -m lp64 -f UTF-8.charmap -x UTF-8.x -i fr_FR.UTF-8.src \
-L "-R\\\$ORIGIN/../../common/amd64 -Bdirect -M /usr/lib/ld/map.pagealign \
-M /usr/lib/ld/map.noexdata" fr_FR.UTF-8@custom
$ mv fr_FR.UTF-8@custom.so.3 amd64/
```

6. **Build the 32-bit locale object.**

```
$ localedef -m ilp32 -f UTF-8.charmap -x UTF-8.x -i fr_FR.UTF-8.src \
-L "-R\\\$ORIGIN/../common" fr_FR.UTF-8@custom
```

7. **Install the custom locale.**

```
$ sudo mkdir -p /usr/lib/locale/fr_FR.UTF-8\@custom/amd64
$ sudo cp fr_FR.UTF-8\@custom.so.3 /usr/lib/locale/fr_FR.UTF-8\@custom/
$ sudo cp amd64/fr_FR.UTF-8\@custom.so.3 /usr/lib/locale/fr_FR.UTF-8\@custom/amd64/
```

8. **Type the following command to start using the locale:**

```
$ export LANG=fr_FR.UTF-8@custom
```

Creating a Locale From Scratch

Creating a locale from scratch is something that is rarely needed. The same approach as in the “Creating a New Locale Based on a System Locale” on page 74 can be used to create a locale from scratch. Refer the `localedef(1)`, `locale(5)`, `extensions(5)` and `charmap(5)` man pages for more detailed information on locales and options available for the `localedef`, `charmap` and `extension` source files.

♦ ♦ ♦ APPENDIX A

Available Locales and Supported Character Sets

The available locales can be classified as recommended locales and additional locales. The following tables summarize the locales available in Oracle Solaris 11, including details of the supported character sets wherever appropriate.

- Table 17, "Recommended Locales," on page 77 – This table provides the list of locales available from the default package `system/locale`.
- Table 18, "Additional Locales," on page 81 – This table provides the list of traditional (legacy) locales available from the optional package `system/locale/extra`.

For more information about managing installed locales, see "Adding or Removing Locales by Using `nlsadm`" on page 37.

TABLE 17 Recommended Locales

Locale	Code Set	Description
C	US-ASCII	C , POSIX
POSIX	US-ASCII	C, POSIX
af_ZA.UTF-8	Unicode 6.0	Afrikaans, South Africa
ar_AE.UTF-8	Unicode 6.0	Arabic, United Arab Emirates
ar_BH.UTF-8	Unicode 6.0	Arabic, Bahrain
ar_DZ.UTF-8	Unicode 6.0	Arabic, Algeria
ar_EG.UTF-8	Unicode 6.0	Arabic, Egypt
ar_IQ.UTF-8	Unicode 6.0	Arabic, Iraq
ar_JO.UTF-8	Unicode 6.0	Arabic, Jordan
ar_KW.UTF-8	Unicode 6.0	Arabic, Kuwait
ar_LY.UTF-8	Unicode 6.0	Arabic, Libya
ar_MA.UTF-8	Unicode 6.0	Arabic, Morocco
ar_OM.UTF-8	Unicode 6.0	Arabic, Oman
ar_QA.UTF-8	Unicode 6.0	Arabic, Qatar
ar_SA.UTF-8	Unicode 6.0	Arabic, Saudi Arabia
ar_TN.UTF-8	Unicode 6.0	Arabic, Tunisia
ar_YE.UTF-8	Unicode 6.0	Arabic, Yemen

Locale	Code Set	Description
as_IN.UTF-8	Unicode 6.0	Assamese, India
az_AZ.UTF-8	Unicode 6.0	Azerbaijani, Azerbaijan
be_BY.UTF-8	Unicode 6.0	Belarusian, Belarus
bg_BG.UTF-8	Unicode 6.0	Bulgarian, Bulgaria
bn_IN.UTF-8	Unicode 6.0	Bengali, India
bs_BA.UTF-8	Unicode 6.0	Bosnian, Bosnia and Herzegovina
ca_ES.UTF-8	Unicode 6.0	Catalan, Spain
cs_CZ.UTF-8	Unicode 6.0	Czech, Czech Republic
da_DK.UTF-8	Unicode 6.0	Danish, Denmark
de_AT.UTF-8	Unicode 6.0	German, Austria
de_BE.UTF-8	Unicode 6.0	German, Belgium
de_CH.UTF-8	Unicode 6.0	German, Switzerland
de_DE.UTF-8	Unicode 6.0	German, Germany
de_LI.UTF-8	Unicode 6.0	German, Liechtenstein
de_LU.UTF-8	Unicode 6.0	German, Luxembourg
el_CY.UTF-8	Unicode 6.0	Greek, Cyprus
el_GR.UTF-8	Unicode 6.0	Greek, Greece
en_AU.UTF-8	Unicode 6.0	English, Australia
en_BW.UTF-8	Unicode 6.0	English, Botswana
en_CA.UTF-8	Unicode 6.0	English, Canada
en_GB.UTF-8	Unicode 6.0	English, United Kingdom
en_HK.UTF-8	Unicode 6.0	English, Hong Kong SAR China
en_IE.UTF-8	Unicode 6.0	English, Ireland
en_IN.UTF-8	Unicode 6.0	English, India
en_MT.UTF-8	Unicode 6.0	English, Malta
en_NZ.UTF-8	Unicode 6.0	English, New Zealand
en_PH.UTF-8	Unicode 6.0	English, Philippines
en_SG.UTF-8	Unicode 6.0	English, Singapore
en_US.UTF-8	Unicode 6.0	English, U.S.A.
en_ZW.UTF-8	Unicode 6.0	English, Zimbabwe
es_AR.UTF-8	Unicode 6.0	Spanish, Argentina
es_BO.UTF-8	Unicode 6.0	Spanish, Bolivia
es_CL.UTF-8	Unicode 6.0	Spanish, Chile
es_CO.UTF-8	Unicode 6.0	Spanish, Colombia
es_CR.UTF-8	Unicode 6.0	Spanish, Costa Rica
es_DO.UTF-8	Unicode 6.0	Spanish, Dominican Republic
es_EC.UTF-8	Unicode 6.0	Spanish, Ecuador
es_ES.UTF-8	Unicode 6.0	Spanish, Spain

Locale	Code Set	Description
es_GT.UTF-8	Unicode 6.0	Spanish, Guatemala
es_HN.UTF-8	Unicode 6.0	Spanish, Honduras
es_MX.UTF-8	Unicode 6.0	Spanish, Mexico
es_NI.UTF-8	Unicode 6.0	Spanish, Nicaragua
es_PA.UTF-8	Unicode 6.0	Spanish, Panama
es_PE.UTF-8	Unicode 6.0	Spanish, Peru
es_PR.UTF-8	Unicode 6.0	Spanish, Puerto Rico
es_PY.UTF-8	Unicode 6.0	Spanish, Paraguay
es_SV.UTF-8	Unicode 6.0	Spanish, El Salvador
es_US.UTF-8	Unicode 6.0	Spanish, U.S.A.
es_UY.UTF-8	Unicode 6.0	Spanish, Uruguay
es_VE.UTF-8	Unicode 6.0	Spanish, Venezuela
et_EE.UTF-8	Unicode 6.0	Estonian, Estonia
fi_FI.UTF-8	Unicode 6.0	Finnish, Finland
fr_BE.UTF-8	Unicode 6.0	French, Belgium
fr_CA.UTF-8	Unicode 6.0	French, Canada
fr_CH.UTF-8	Unicode 6.0	French, Switzerland
fr_FR.UTF-8	Unicode 6.0	French, France
fr_LU.UTF-8	Unicode 6.0	French, Luxembourg
gu_IN.UTF-8	Unicode 6.0	Gujarati, India
he_IL.UTF-8	Unicode 6.0	Hebrew, Israel
hi_IN.UTF-8	Unicode 6.0	Hindi, India
hr_HR.UTF-8	Unicode 6.0	Croatian, Croatia
hu_HU.UTF-8	Unicode 6.0	Hungarian, Hungary
hy_AM.UTF-8	Unicode 6.0	Armenian, Armenia
id_ID.UTF-8	Unicode 6.0	Indonesian, Indonesia
is_IS.UTF-8	Unicode 6.0	Icelandic, Iceland
it_CH.UTF-8	Unicode 6.0	Italian, Switzerland
it_IT.UTF-8	Unicode 6.0	Italian, Italy
ja_JP.UTF-8	Unicode 6.0	Japanese, Japan JIS X 0201:1976, JIS X 0208:1990, JIS X 0212:1990, JIS X 0213:2004, Vendor-defined characters (VDC), User-defined characters (UDC)
ka_GE.UTF-8	Unicode 6.0	Georgian, Georgia
kk_KZ.UTF-8	Unicode 6.0	Kazakh, Kazakhstan
kn_IN.UTF-8	Unicode 6.0	Kannada, India
ko_KR.UTF-8	Unicode 6.0	Korean, Korea
ks_IN.UTF-8	Unicode 6.0	Kashmiri, India

Locale	Code Set	Description
ku_TR.UTF-8	Unicode 6.0	Kurdish, Turkey
ku_TR.UTF-8@sorani	Unicode 6.0	Kurdish (Sorani), Turkey
ky_KG.UTF-8	Unicode 6.0	Kirghiz, Kyrgyzstan
lt_LT.UTF-8	Unicode 6.0	Lithuanian, Lithuania
lv_LV.UTF-8	Unicode 6.0	Latvian, Latvia
mk_MK.UTF-8	Unicode 6.0	Macedonian, Macedonia
ml_IN.UTF-8	Unicode 6.0	Malayalam, India
mr_IN.UTF-8	Unicode 6.0	Marathi, India
ms_MY.UTF-8	Unicode 6.0	Malay, Malaysia
mt_MT.UTF-8	Unicode 6.0	Maltese, Malta
nb_NO.UTF-8	Unicode 6.0	Bokmal, Norway
nl_BE.UTF-8	Unicode 6.0	Dutch, Belgium
nl_NL.UTF-8	Unicode 6.0	Dutch, Netherlands
nn_NO.UTF-8	Unicode 6.0	Nynorsk, Norway
or_IN.UTF-8	Unicode 6.0	Oriya, India
pa_IN.UTF-8	Unicode 6.0	Punjabi, India
pl_PL.UTF-8	Unicode 6.0	Polish, Poland
pt_BR.UTF-8	Unicode 6.0	Portuguese, Brazil
pt_PT.UTF-8	Unicode 6.0	Portuguese, Portugal
ro_RO.UTF-8	Unicode 6.0	Romanian, Romania
ru_RU.UTF-8	Unicode 6.0	Russian, Russia
ru_UA.UTF-8	Unicode 6.0	Russian, Ukraine
sa_IN.UTF-8	Unicode 6.0	Sanskrit, India
sk_SK.UTF-8	Unicode 6.0	Slovak, Slovakia
sl_SI.UTF-8	Unicode 6.0	Slovenian, Slovenia
sq_AL.UTF-8	Unicode 6.0	Albanian, Albania
sr_ME.UTF-8	Unicode 6.0	Serbian, Montenegro
sr_ME.UTF-8@latin	Unicode 6.0	Serbian, Montenegro (Latin)
sr_RS.UTF-8	Unicode 6.0	Serbian, Serbia
sr_RS.UTF-8@latin	Unicode 6.0	Serbian, Serbia (Latin)
sv_SE.UTF-8	Unicode 6.0	Swedish, Sweden
ta_IN.UTF-8	Unicode 6.0	Tamil, India
te_IN.UTF-8	Unicode 6.0	Telugu, India
th_TH.UTF-8	Unicode 6.0	Thai, Thailand
tr_TR.UTF-8	Unicode 6.0	Turkish, Turkey
uk_UA.UTF-8	Unicode 6.0	Ukrainian, Ukraine
vi_VN.UTF-8	Unicode 6.0	Vietnamese, Vietnam
zh_CN.UTF-8	Unicode 6.0	Simplified Chinese, China

Locale	Code Set	Description
zh_HK.UTF-8	Unicode 6.0	Traditional Chinese, Hong Kong SAR China
zh_SG.UTF-8	Unicode 6.0	Chinese, Singapore
zh_TW.UTF-8	Unicode 6.0	Traditional Chinese, Taiwan

TABLE 18 Additional Locales

Locales	Code Set	Description
ar_EG.ISO8859-6	ISO8859-6	Arabic, Egypt
bg_BG.ISO8859-5	ISO8859-5	Bulgarian, Bulgaria
bs_BA.ISO8859-2	ISO8859-2	Bosnian, Bosnia and Herzegovina
ca_ES.ISO8859-1	ISO8859-1	Catalan, Spain
ca_ES.ISO8859-15	ISO8859-15	Catalan, Spain
cs_CZ.ISO8859-2	ISO8859-2	Czech, Czech Republic
cs_CZ.UTF-8@euro	Unicode 6.0	Czech, Czech Republic (Euro)
da_DK.ISO8859-1	ISO8859-1	Danish, Denmark
da_DK.ISO8859-15	ISO8859-15	Danish, Denmark
da_DK.ISO8859-15@euro	ISO8859-15	Danish, Denmark (Euro)
de_AT.ISO8859-1	ISO8859-1	German, Austria
de_AT.ISO8859-15	ISO8859-15	German, Austria
de_CH.ISO8859-1	ISO8859-1	German, Switzerland
de_DE.ISO8859-1	ISO8859-1	German, Germany
de_DE.ISO8859-15	ISO8859-15	German, Germany
el_GR.ISO8859-7	ISO8859-7	Greek, Greece
en_AU.ISO8859-1	ISO8859-1	English, Australia
en_CA.ISO8859-1	ISO8859-1	English, Canada
en_GB.ISO8859-1	ISO8859-1	English, United Kingdom
en_GB.ISO8859-15	ISO8859-15	English, United Kingdom
en_GB.ISO8859-15@euro	ISO8859-15	English, United Kingdom (Euro)
en_IE.ISO8859-1	ISO8859-1	English, Ireland
en_IE.ISO8859-15	ISO8859-15	English, Ireland
en_NZ.ISO8859-1	ISO8859-1	English, New Zealand
en_US.ISO8859-1	ISO8859-1	English, U.S.A.
en_US.ISO8859-15	ISO8859-15	English, U.S.A.
en_US.ISO8859-15@euro	ISO8859-15	English, U.S.A. (Euro)
es_AR.ISO8859-1	ISO8859-1	Spanish, Argentina
es_BO.ISO8859-1	ISO8859-1	Spanish, Bolivia
es_CL.ISO8859-1	ISO8859-1	Spanish, Chile
es_CO.ISO8859-1	ISO8859-1	Spanish, Colombia

Locales	Code Set	Description
es_CR.ISO8859-1	ISO8859-1	Spanish, Costa Rica
es_EC.ISO8859-1	ISO8859-1	Spanish, Ecuador
es_ES.ISO8859-1	ISO8859-1	Spanish, Spain
es_ES.ISO8859-15	ISO8859-15	Spanish, Spain
es_GT.ISO8859-1	ISO8859-1	Spanish, Guatemala
es_MX.ISO8859-1	ISO8859-1	Spanish, Mexico
es_NI.ISO8859-1	ISO8859-1	Spanish, Nicaragua
es_PA.ISO8859-1	ISO8859-1	Spanish, Panama
es_PE.ISO8859-1	ISO8859-1	Spanish, Peru
es_PY.ISO8859-1	ISO8859-1	Spanish, Paraguay
es_SV.ISO8859-1	ISO8859-1	Spanish, El Salvador
es_UY.ISO8859-1	ISO8859-1	Spanish, Uruguay
es_VE.ISO8859-1	ISO8859-1	Spanish, Venezuela
et_EE.ISO8859-15	ISO8859-15	Estonian, Estonia
fi_FI.ISO8859-1	ISO8859-1	Finnish, Finland
fi_FI.ISO8859-15	ISO8859-15	Finnish, Finland
fr_BE.ISO8859-1	ISO8859-1	French, Belgium
fr_BE.ISO8859-15	ISO8859-15	French, Belgium
fr_CA.ISO8859-1	ISO8859-1	French, Canada
fr_CH.ISO8859-1	ISO8859-1	French, Switzerland
fr_FR.ISO8859-1	ISO8859-1	French, France
fr_FR.ISO8859-15	ISO8859-15	French, France
he_IL.ISO8859-8	ISO8859-8	Hebrew, Israel
hr_HR.ISO8859-2	ISO8859-2	Croatian, Croatia
hu_HU.ISO8859-2	ISO8859-2	Hungarian, Hungary
is_IS.ISO8859-1	ISO8859-1	Icelandic, Iceland
it_IT.ISO8859-1	ISO8859-1	Italian, Italy
it_IT.ISO8859-15	ISO8859-15	Italian, Italy
ja_JP.PCK	PCK	Japanese, Japan (PC Kanji code, aka. Shift-JIS)
		JIS X 0201:1976, JIS X 0208:1990, Vendor-defined characters (VDC), User-defined characters (UDC)
ja_JP.eucJP	EUC-JP	Japanese EUC environment. Compliant to UI-OSF Japanese Environment Implementation Agreement Version 1.1
		JIS X 0201:1976, JIS X 0208:1990, JIS X 0212:1990 VDC, UDC
ja_JP.UTF-8@cldr	Unicode 6.0	Japanese, Japan (CLDR)

Locales	Code Set	Description
		JIS X 0201:1976, JIS X 0208:1990, JIS X 0212:1990, JIS X 0213:2004, VDC, UDC
ko_KR.EUC	KS X 1001	Korean, Korea
ko_KR.EUC@dict	KS X 1001	Korean, Korea (dict)
ko_KR.UTF-8@dict	Unicode 6.0	Korean, Korea (dict)
lt_LT.ISO8859-13	ISO8859-13	Lithuanian, Lithuania
lv_LV.ISO8859-13	ISO8859-13	Latvian, Latvia
mk_MK.ISO8859-5	ISO8859-5	Macedonian, Macedonia
nb_NO.ISO8859-1	ISO8859-1	Norwegian Bokmal, Norway
nl_BE.ISO8859-1	ISO8859-1	Dutch, Belgium
nl_BE.ISO8859-15	ISO8859-15	Dutch, Belgium
nl_NL.ISO8859-1	ISO8859-1	Dutch, Netherlands
nl_NL.ISO8859-15	ISO8859-15	Dutch, Netherlands
nn_NO.ISO8859-1	ISO8859-1	Norwegian Nynorsk, Norway
pl_PL.ISO8859-2	ISO8859-2	Polish, Poland
pt_BR.ISO8859-1	ISO8859-1	Portuguese, Brazil
pt_PT.ISO8859-1	ISO8859-1	Portuguese, Portugal
pt_PT.ISO8859-15	ISO8859-15	Portuguese, Portugal
ro_RO.ISO8859-2	ISO8859-2	Romanian, Romania
ru_RU.ANSI1251	ANSI1251	Russian, Russia
ru_RU.ISO8859-5	ISO8859-5	Russian, Russia
ru_RU.KOI8-R	KOI8-R	Russian, Russia
sk_SK.ISO8859-2	ISO8859-2	Slovak, Slovakia
sl_SI.ISO8859-2	ISO8859-2	Slovenian, Slovenia
sq_AL.ISO8859-2	ISO8859-2	Albanian, Albania
sr_ME.ISO8859-5	ISO8859-5	Serbian, Montenegro
sv_SE.ISO8859-1	ISO8859-1	Swedish, Sweden
sv_SE.ISO8859-15	ISO8859-15	Swedish, Sweden
sv_SE.ISO8859-15@euro	ISO8859-15	Swedish, Sweden (Euro)
sv_SE.UTF-8@euro	Unicode 6.0	Swedish, Sweden (Euro)
th_TH.TIS620	TIS-620	Thai, Thailand
tr_TR.ISO8859-9	ISO8859-9	Turkish, Turkey
zh_CN.EUC	GB2312	Simplified Chinese, China
zh_CN.EUC@pinyin	GB2312	Simplified Chinese, China (pinyin)
zh_CN.EUC@radical	GB2312	Simplified Chinese, China (radical)
zh_CN.EUC@stroke	GB2312	Simplified Chinese, China (stroke)
zh_CN.GB18030	GB18030	Simplified Chinese, China
zh_CN.GB18030@pinyin	GB18030	Simplified Chinese, China (pinyin)

Locales	Code Set	Description
zh_CN.GB18030@radical	GB18030	Simplified Chinese, China (radical)
zh_CN.GB18030@stroke	GB18030	Simplified Chinese, China (stroke)
zh_CN.GBK	GBK	Simplified Chinese, China
zh_CN.GBK@pinyin	GBK	Simplified Chinese, China (pinyin)
zh_CN.GBK@radical	GBK	Simplified Chinese, China (radical)
zh_CN.GBK@stroke	GBK	Simplified Chinese, China (stroke)
zh_CN.UTF-8@pinyin	Unicode 6.0	Simplified Chinese, China (pinyin)
zh_CN.UTF-8@radical	Unicode 6.0	Simplified Chinese, China (radical)
zh_CN.UTF-8@stroke	Unicode 6.0	Simplified Chinese, China (stroke)
zh_HK.BIG5HK	BIG5-HKSCS	Traditional Chinese, Hong Kong SAR China
zh_HK.BIG5HK@radical	BIG5-HKSCS	Traditional Chinese, Hong Kong SAR China (radical)
zh_HK.BIG5HK@stroke	BIG5-HKSCS	Traditional Chinese, Hong Kong SAR China (stroke)
zh_HK.UTF-8@radical	Unicode 6.0	Traditional Chinese, Hong Kong SAR China (radical)
zh_HK.UTF-8@stroke	Unicode 6.0	Traditional Chinese, Hong Kong SAR China (stroke)
zh_TW.BIG5	BIG5	Traditional Chinese, Taiwan
zh_TW.BIG5@pinyin	BIG5	Traditional Chinese, Taiwan (pinyin)
zh_TW.BIG5@radical	BIG5	Traditional Chinese, Taiwan (radical)
zh_TW.BIG5@stroke	BIG5	Traditional Chinese, Taiwan (stroke)
zh_TW.BIG5@zhuyin	BIG5	Traditional Chinese, Taiwan (zhuyin)
zh_TW.EUC	CNS11643-1992	Traditional Chinese, Taiwan
zh_TW.EUC@pinyin	CNS11643	Traditional Chinese, Taiwan (pinyin)
zh_TW.EUC@radical	CNS11643	Traditional Chinese, Taiwan (radical)
zh_TW.EUC@stroke	CNS11643	Traditional Chinese, Taiwan (stroke)
zh_TW.EUC@zhuyin	CNS11643	Traditional Chinese, Taiwan (zhuyin)
zh_TW.UTF-8@pinyin	UTF-8	Traditional Chinese, Taiwan (pinyin)
zh_TW.UTF-8@radical	UTF-8	Traditional Chinese, Taiwan (radical)
zh_TW.UTF-8@stroke	UTF-8	Traditional Chinese, Taiwan (stroke)
zh_TW.UTF-8@zhuyin	UTF-8	Traditional Chinese, Taiwan (zhuyin)

Note - In earlier Oracle Solaris releases specification of `ja` locale (compatible with older Oracle Solaris release) was different from `ja_JP.eucJP` locale, but `ja` locale is now locale alias to `ja_JP.eucJP` in Oracle Solaris 11.

Index