Unicode Support in the Solaris Operating Environment

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

Part Number 806-5584
May 2000

Adobe PostScript

Please
Recycle

Contents

Preface

The *Unicode Support in the Solaris™ Operating Environment* white paper presents information and software features for internationalizing software with Unicode.

Who Should Use This Book

This white paper is intended for software developers who are interested in developing internationalized software with Unicode in the Solaris™ operating environment. This white paper is part of a 4–part series on internationalization for Solaris software developers. The four internationalization white papers are:

- *Asian-Language Support in the Solaris™ Operating Environment*
- *Complex Text Layout Language Support in the Solaris™ Operating Environment*
- *Unicode Support in the Solaris™ Operating Environment*
- *Euro Currency Support in the Solaris™ Operating Environment*

How This Book Is Organized

Chapter 1 describes Unicode, multilingual computing, and software internationalization.

Chapter 2 provides an overview of the Unicode standard.

Chapter 3 provides information about Unicode in the Solaris Operating Environment.

Chapter 4 addresses the technical concerns of Unicode in an internationalized application.

Appendix A lists the codeset conversions.

Related Books

The following books are related to software internationalization:

- *Creating Worldwide Software: Solaris International Developer's Guide* Bill Tuthill and David Smallberg.
- *Internationalization Guide, Version 2: Open Group Guide* The Open Group
- *International Language Environments Guide* Solaris Developer Collection.
- *Programming for the World: A Guide to Internationalization* Sandra Martin O'Donnell.
- *The Unicode Standard, Version 3.0* The Unicode Consortium.
- *X Windows on the World, Developing Internationalized Software with X, Motif, and CDE* Thomas C. McFarland.

Ordering Sun Documents

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

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

Accessing Sun Documentation Online

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

CHAPTER **1**

Unicode and Multilingual Computing

Today's global economy demands global computing solutions. Instant communications across continents—and computer platforms—characterize a business world at work 24 hours a day, 7 days a week. The widespread use of the Internet and e-commerce continue to create new international challenges.

More and more, users are demanding a computing environment to suit their own linguistic and cultural needs. They want applications and file formats they can share around the world, interfaces in their own language, and local time and date displays. Essentially, users want to write and speak at the keyboard the way they write and speak in the office.

The Solaris operating environment multilingual framework (including multiple character sets and multiple cultural attributes) uses the standard universal encoding codeset, Unicode (*The Unicode Standard, Version 3.0*). Unicode is well-suited to applications such as multilingual databases, e-commerce, and government research and reference.

1.1 Multilingual Computing

"Multilingual" computing can mean:

- Multilanguage—multiple launches of one locale, one script.
- Multiscript—single launch of one locale, multiple scripts.
- Multilingual—single launch of multiple locales, multiple scripts.

The movement from multilanguage to multiscript to multilingual implies an increased level of complexity in the underlying operating environment.

1.1.1 Multilanguage Environment

In a *multilanguage* environment, a locale supports one script and one set of cultural attributes. An application inherits all the language and cultural attributes of the current locale. Document text is written in one script and text manipulated according to the locale language rules. A separate application launch in another locale is required to use different language and cultural attributes.

For example, to write a document in Chinese, a user first sets the Chinese locale before launching the application. To write a Russian document, the Russian locale must be separately set and the application launched again. Chinese and Russian text cannot be mixed in the same document

1.1.2 Multiscript Environment

In a *multiscript* environment, a locale can support more than one script, but only one locale can be set as current. An application creates a document in different scripts by tagging each separate script run (text in the same script). However, the current locale environment settings apply—for example, text is sorted according to the sorting rules of the current locale.

In the Chinese/Russian example above, rather than create two separate documents, the user creates one multiscript document containing both Chinese and Russian text. The cultural attributes of the active locale still apply—in the Chinese locale, the Chinese sorting rules apply to the mixed-script text.

Note - In a Unicode locale, tagging script runs is not necessary because all language attributes are inherent in the Unicode codeset.

1.1.3 Multilingual Environment

In a *multilingual* environment, a locale can support multiple scripts and multiple cultural attributes, giving an application greater control over text manipulation. For example, a document containing text in multiple scripts can sort text according to the sort order of each script rather than the current locale.

In the Chinese/Russian example above, the Chinese locale sorting rules apply to the Chinese text and the Russian sorting rules apply to the Russian text.

The multilingual environment is closest to the ideal of multilingual computing. An application uses locale data from numerous locales, while at the same time allowing easy text manipulation in a variety of scripts. All users can easily work in their own language and be understood by others around the world.

1.2 Software Internationalization

Sun Microsystems defines the following levels at which an application can support a customer's international needs:

- Internationalization
- Localization

Software *internationalization* is the process of designing and implementing software to transparently manage different linguistic and cultural conventions without additional modification. The same binary copy of an application should run on any localized version of the Solaris operating environment, without requiring source code changes or recompilation.

Software *localization* is the process of adding language translation (including text messages, icons, buttons, and so on), cultural data, and components (such as input methods and spell checkers) to a product to meet regional market requirements.

The Solaris operating environment is an example of a product that supports both internationalization and localization. The Solaris operating environment is a single internationalized binary that is localized into various languages (for example, French, Japanese, and Chinese) to support the language and cultural conventions of each language.

Properly designed applications can easily accommodate a localized interface without extensive modification. One suggestion for creating easy-to-localize software is to first internationalize the software and then encapsulate the language- and cultural-specific elements in a locale-specific database. This greatly simplifies the localization process, should a developer choose to localize in the future.

At a minimum, Sun Microsystems strongly encourages developers to internationalize their software. Internationalized applications can run on any localized version of the Solaris operating environment and easily manage the language and cultural preferences.

1.3 Internationalization Framework

A properly internationalized application *separates language– and cultural-specific information from the application code*. The Solaris operating environment internationalization framework achieves this with:

- Locales.
- Localizable interface.
- Codeset independence.

A *locale* is the language and cultural data set by the user and dynamically loaded into memory at run time. The locale settings are applied to the operating system and to subsequent application launches.

The Solaris operating environment includes APIs for developers to directly access language and cultural data in the current locale. Applications can run in any locale without prior input of language or cultural data. For example, an application does not need to encode a particular currency symbol. By calling the appropriate system API, the current locale currency symbol is returned.

A *localizable interface* considers variations in an interface translated into another language. The Solaris operating environment provides messaging APIs and utilities to collect, generate, and process messages.

In an application, *codeset independence* does not assume a particular codeset. For example, text-handling routines should not define in advance the size of the character codeset.

For more information about designing applications with Unicode, see Chapter 3, *Technical Considerations*. For more information about the internationalization framework, see the whitepaper *Asian Language Support in the Solaris Operating Environment*.

1.4 Supporting the Unicode Standard

Unicode (Universal Codeset) is a universal character encoding scheme developed and promoted by the Unicode Consortium, a non-profit organization which includes Sun Microsystems. The Unicode standard encompasses most alphabetic, ideographic, and symbolic characters.

Using one universal codeset enables applications to support text from multiple scripts in the same documents without elaborate tagging. However, applications must treat Unicode as any another codeset—applying codeset independence to Unicode as well.

Unicode locales are called the same way and function the same way as all other locales in the Solaris operating environment. These locales provide the extra benefits that the Unicode codeset brings to the work environment, including the ability to create text in multiple scripts without having to switch locales. Sun Microsystems provides the same level of Unicode locale support for both 32-bit and 64-bit Solaris environments.

1.5 Benefits of Unicode

Support for Unicode provides many benefits to application developers, including:

- Global source and binary.
- Support for mixed-script computing environments.
- Improved cross-platform data interoperability through a common codeset.
- Space-efficient encoding scheme for data storage.
- Reduced time-to-market for localized products.
- Expanded market access.

Developers can use Unicode to create global applications. Users can exchange data more freely using one flat codeset without elaborate code conversions to comprehend characters.

In the Solaris operating environment internationalization framework, Unicode is "just another codeset." By adopting and implementing codeset independence to design, applications can handle different codesets without extensive code rework to support specific languages.

CHAPTER **2**

Unicode

In most writing systems, keyboard input is converted into character codes, stored in memory, and converted to glyphs in a particular font for display and printing. The collection of characters and character codes form a codeset. To represent characters of different languages, a different codeset is used.

A character code in one codeset, however, does not necessarily represent the same character in another codeset. For example, the character code `0xB1` is the plus-minus sign (±) in Latin-1 (ISO 8859–1 codeset), capital BE in Cyrillic (ISO 8859–5 codeset), and does not represent anything in Arabic (ISO 8859–6 codeset) or Traditional Chinese (CJK unified ideographs).

In Unicode, every character, ideograph, and symbol has a unique character code, eliminating any confusion between character codes of different codesets. In Unicode, multiple codesets need not be defined. Unicode represents characters from most of the world's languages as well as publishing characters, mathematical and technical symbols, and punctuation characters. This universal representation for text data has been further enhanced and extended in the latest release of Unicode: *The Unicode Standard, Version 3.0.*

2.1 Unicode Coded Representations

In recent years, the Unicode Consortium and other related organizations have developed different formats to represent and store a Unicode codeset. To represent characters from all major languages in multibyte format, the ISO/IEC International Standard 10646-1 (commonly referred to as 10646) has defined the Universal Multiple-Octet Coded Character Set (UCS) format. Character forms contained in the 10464 specifications are:

- Universal Coded Character Set-2 (UCS-2) also known as Basic Multilingual Plane (BMP)—characters are encoded in two bytes on a single plane.
- Universal Coded Character Set-4 (UCS-4)—characters encoded in four bytes on multiple planes and multiple groups.
- UCS Transformation Format 16-bit form (UTF-16)—extended variant of UCS-2 with characters encoded in 2-4 bytes.
- UCS Transformation Format 8-bit form (UTF-8)—a transformation format using characters encoded in 1-6 bytes.

UCS-2 defines a 64K coding space, or BMP, to represent character codes in a two-octet row and cell format. The row and cell octets designate the cell location of a particular character code within a 256 by 256 (00-FF) plane.

UCS-4 defines a four-octet coding space divided into four units: group, plane, row, and cell. The row and cell octets designate the cell location of a particular character code within a plane. The plane octet designates the plane number (00-FF), and the group octet the group number (00–7F) to which the plane belongs. In total, there are 256 planes occurring 127 times.

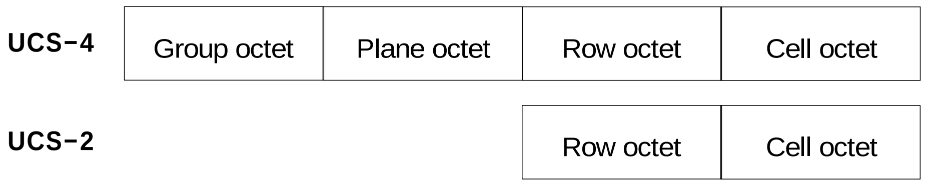


Figure 2–1 UCS-2 and UCS-4 coding schemes

In addition to the 10646 UCS forms, Unicode defines another form called UTF (UCS Transformation Format). One version of UTF is an extended UCS-2 encoding form designed to include characters from outside the BMP 64K coding space. This form was first called UCS-2E (extended UCS-2), but is now known as UTF-16 (UCS Transformation Format 16-bit form).

The UTF-16 form translates a range of UCS-4 codes into a two-octet encoded string. It does this by reserving an area of codes in the BMP coding space for mapping to and from 16 planes of group 00 of UCS-4. Each plane is assigned a certain set of code positions in the two-octet UCS-2 scheme. Specifically, Planes 01 to 0E (14 planes, or 14 x 65,536 = 917,504 characters) are reserved for standard encodings and Planes 0F and 10 (2 planes, or 2 x 65,536 = 131,072 characters) are reserved for private use.

Although UCS-4 and UTF-16 provide comprehensive ways to represent several character sets, they do not preserve the byte values for ASCII characters. Because all UNIX systems are based on an ASCII kernel, they reserve certain character codes for I/O operations, such as the null character as a string terminator, the slash (/)

character as a path name separator, and the DEL and SPACE control characters. To circumvent this problem, another version of UTF was devised, called FSS-UTF (File System Safe-UTF), now commonly known as UTF-8.

UTF-8 is an encoding scheme which maps the entire UCS-4 character set to a series of single-octet and multi-octet strings. In this scheme, the most significant bit is 0 for ASCII characters and 1 for all other characters. The ASCII character range is contained in a single-byte encoding, and all other characters in a range from 2 up to 6-byte encoding.

TABLE 2–1 UTF-8 encoding scheme

Bits	**Hex Min**	**Hex Max**	**UTF-8 Binary Encoding**
7	00000000	0000007F	0xxxxxxx
11	00000080	000007FF	110xxxxx 10xxxxxx
16	00000800	0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
21	00010000	001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
26	00200000	03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
31	04000000	7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Note - The UTF-8 scheme does not use any ASCII byte values in its 2- to 6-byte sequences, yet ASCII values remain 8-bit within the new byte structure. Thus, UTF-8 is compatible with all legacy file systems and other systems that parse for the ASCII byte, while UCS-2/UTF-16 and UCS-4 are not compatible with ASCII.

Furthermore, applications supporting Unicode can use existing data in ASCII format without applying a conversion utility. In addition, there is support within the Internet community for adopting UTF-8 as the Internet encoding standard.

In addition to its backward compatibility with 7-bit ASCII, UTF-8 is a space-efficient encoding scheme when the encoded data needs only one-byte or less (as for English and other Roman character-based writing systems). Because UTF-8 stores one-byte data as one byte, rather than, for example, the two bytes required by UTF-16, this can significantly decrease the storage space required to hold large blocks of international data.

Because of its flexibility and compatibility with ASCII and UNIX, Unicode support of the UTF-8 format is used in the Solaris operating environment. UTF-8 provides developers with a format compatible with existing internationalized environments and an easy path for Internet and legacy data interoperability. As a file system safe format, UTF-8 supports one-byte unit I/O operations and can represent the Unicode formats UCS-2 and UCS-4. Furthermore, UTF-8 fits well within the XPG internationalization framework.

CHAPTER **3**

Unicode in the Solaris 8 Operating Environment

The support of Unicode, Version 3.0 in the Solaris 8 Operating Environment's Unicode locales has provided an enhanced framework for developing multiscript applications. Properly internationalized applications require no changes to support the Unicode locales. All internationalized CUI and GUI utilities and commands in the Solaris operating environment are available in Unicode locales without modification.

All Unicode locales in the Solaris operating environment are based on the UTF-8 format. Each locale includes a base language in the UTF-8 codeset and regional data related to the base language and its cultural conventions (such as local formatting rules, text messages, help messages, and other related files). Each locale also supports several other scripts for input, display, code conversion, and printing.

3.1 Unicode UTF-8 `en_US.UTF-8` Locale

`en_US.UTF-8` is the flagship Unicode locale in the Solaris operating environment. The `en_US.UTF-8` locale is an American English-based locale with multiscript processing support for characters in many different languages. New and enhanced features of all Unicode locales include support of the Unicode 3.0 character set, complex text layout scripts in correct rendition, native Asian input methods, more MIME character sets in `dtmail`, various new `iconv` code conversions, and an enhanced PostScript print filter.

All Unicode locales in the Solaris operating environment support multiple scripts. Thirteen input modes area available: English / European, Cyrillic, Greek, Arabic, Hebrew, Thai, Unicode Hex, Unicode Octal, Table lookup, Japanese, Korean, Simplified Chinese, and Traditional Chinese.

Users can input characters from any combination of scripts and the entire Unicode coding space.

Note - To choose an input mode, press the `Compose` key and a two-letter code. For example, to input text in Thai, press `Compose+tt`. Alternatively, click the status area and select an input mode as shown in Figure 3–1. (To select the default English/European mode, press `Control+Space`.)

TABLE 3–1 UTF-8 Input Mode two-letter codes

Language	**Code**
Cyrillic	`cc`
Greek	`gg`
Thai	`tt`
Arabic	`ar`
Hebrew	`hh`
Unicode Hex	`uh`
Unicode Octal	`uo`
Lookup	`ll`
Japanes	`ja`
Korean	`ko`
Simplified Chinese	`sc`
Traditional Chinese	`tc`
English/European	`Control+Space`

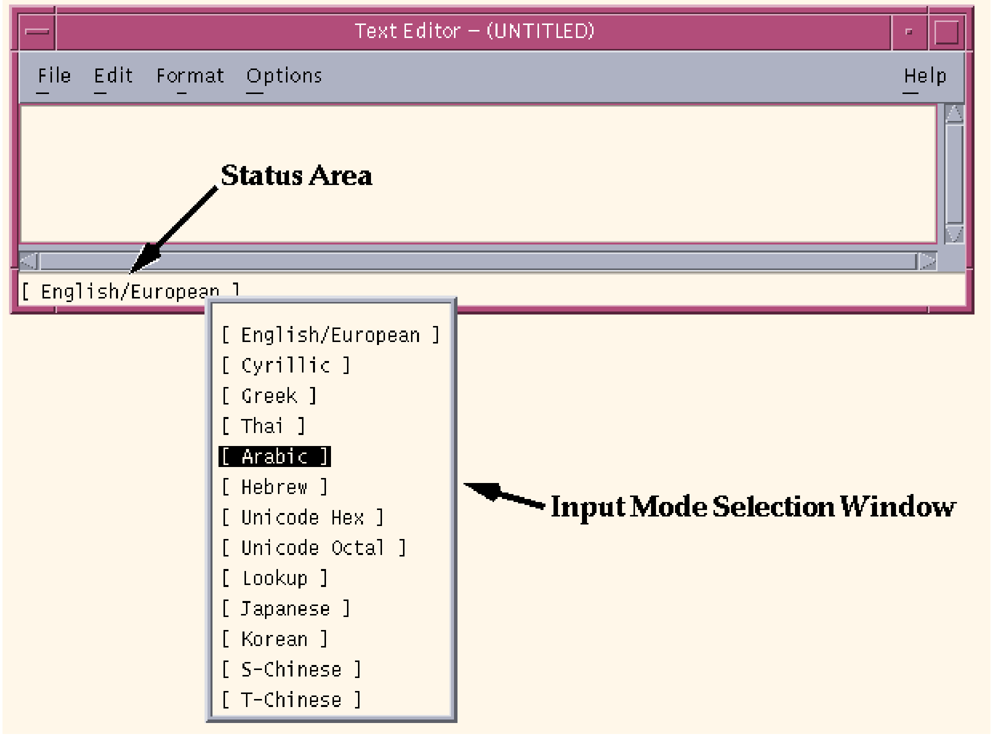


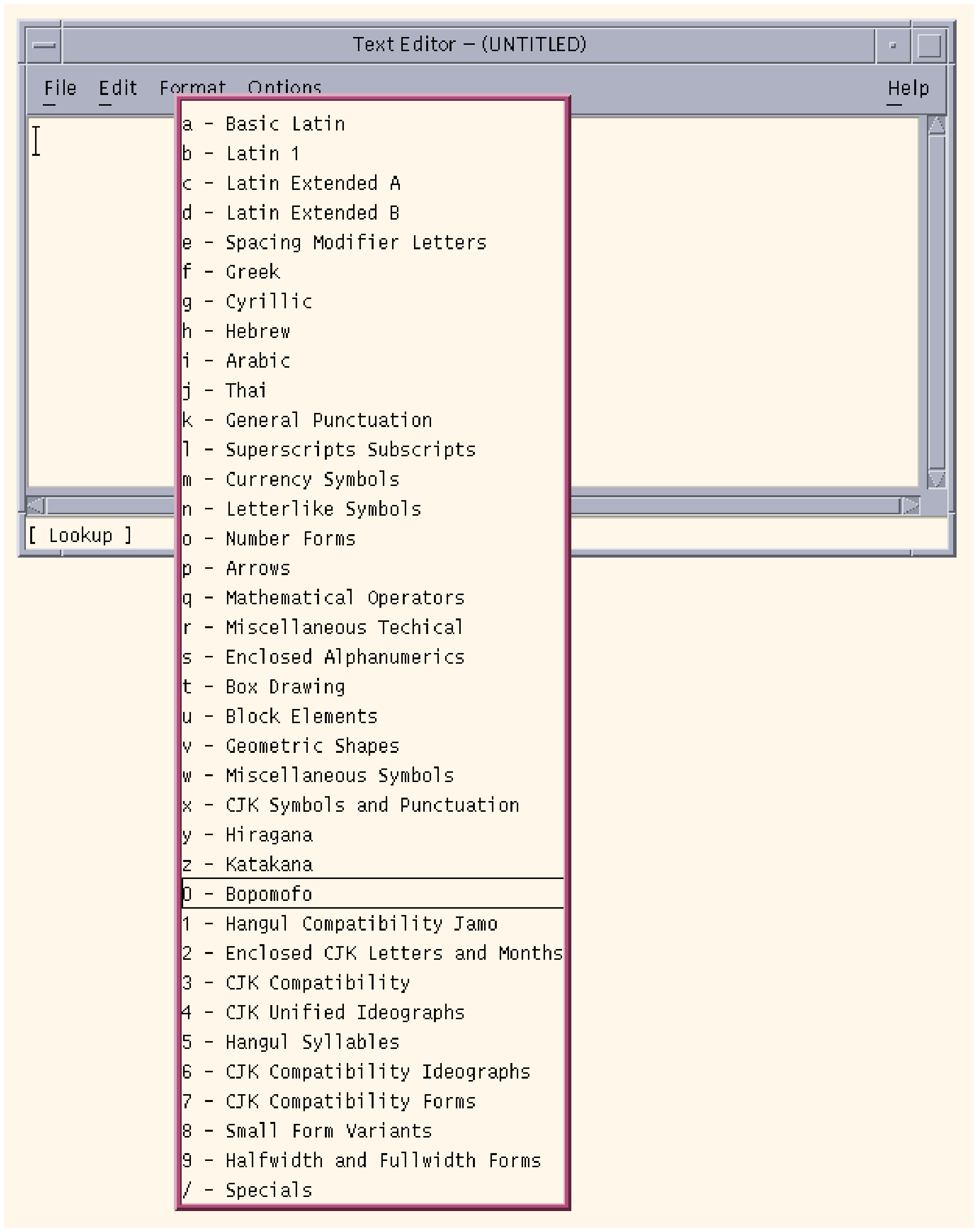
Figure 3–1 UTF-8 Input Mode selection

To input text from a Lookup table, select the Lookup input mode. A lookup table with all input modes and various symbol and technical codesets appears, as shown in Figure 3–2.

The Table lookup input mode is the easiest for non-native speakers to input characters in a foreign language—a lookup window displays characters from a selected script, as shown for the Asian input mode in Figure 3–3.

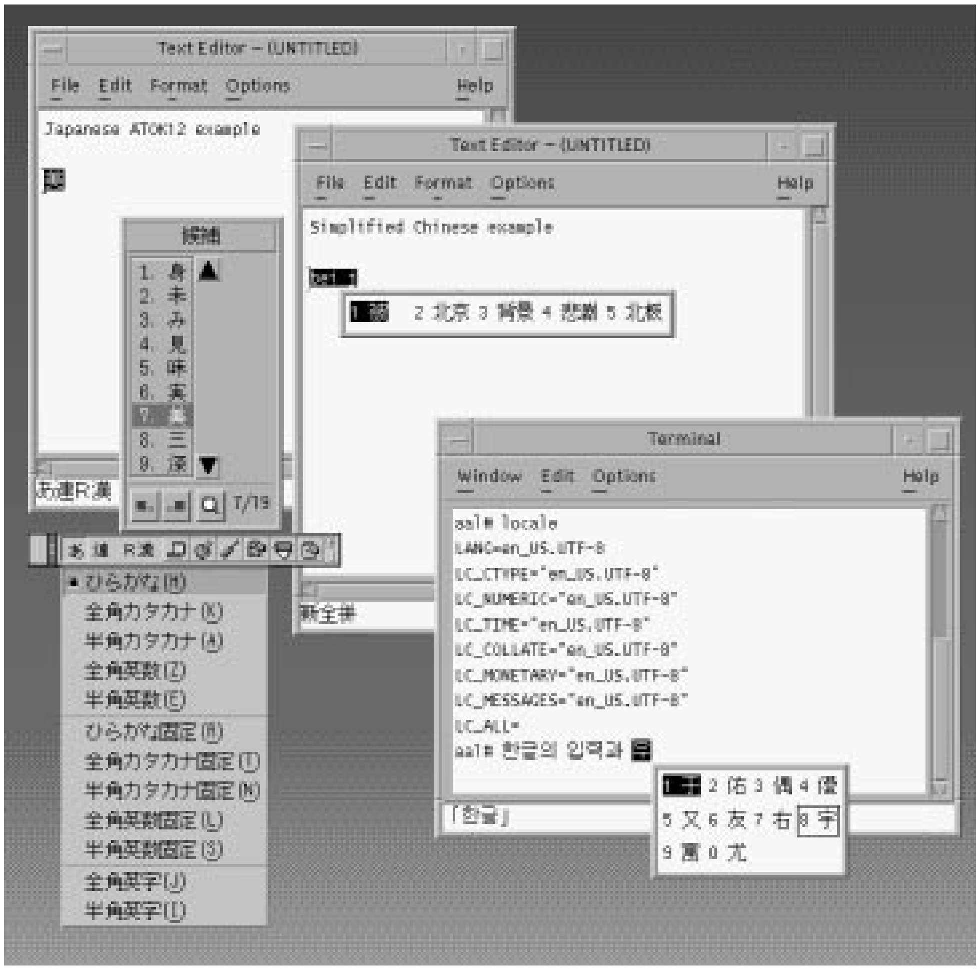
The Arabic, Hebrew, and Thai input modes provide full complex text layout features, including right-to-left display and context-sensitive character rendering. The Unicode octal and hexadecimal code input modes generate Unicode characters from their octal and hexadecimal equivalents, respectively.

The Japanese, Korean, Simplified Chinese, and Traditional Chinese input modes provide full native Asian input.



```
a - Basic Latin
b - Latin 1
c - Latin Extended A
d - Latin Extended B
e - Spacing Modifier Letters
f - Greek
g - Cyrillic
h - Hebrew
i - Arabic
j - Thai
k - General Punctuation
l - Superscripts Subscripts
m - Currency Symbols
n - Letterlike Symbols
o - Number Forms
p - Arrows
q - Mathematical Operators
r - Miscellaneous Techical
s - Enclosed Alphanumerics
t - Box Drawing
u - Block Elements
v - Geometric Shapes
w - Miscellaneous Symbols
x - CJK Symbols and Punctuation
y - Hiragana
z - Katakana
0 - Bopomofo
1 - Hangul Compatibility Jamo
2 - Enclosed CJK Letters and Months
3 - CJK Compatibility
4 - CJK Unified Ideographs
5 - Hangul Syllables
6 - CJK Compatibility Ideographs
7 - CJK Compatibility Forms
8 - Small Form Variants
9 - Halfwidth and Fullwidth Forms
/ - Specials
```

Figure 3–2 UTF-8 Table Lookup



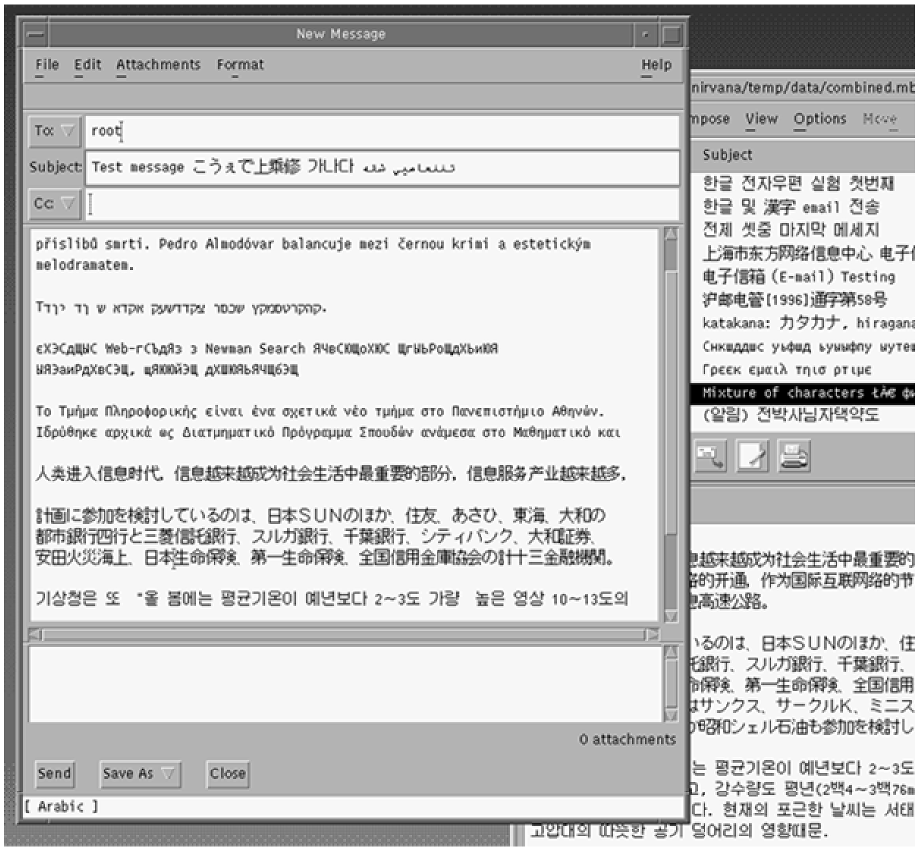
```
aal# locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_ALL=
aal# 한글의 입력과
```

Figure 3–3 Asian input mode

For more information on each input method, refer to the chapter *Overview of en_US.UTF-8 Locale Support* in the latest Solaris *International Language Environments Guide*, *ATOK12 User's Guide*, *Wnn6 User's Guide*, *cs00 User's Guide*, *Korean Solaris User's Guide*, *Simplified Chinese Solaris User's Guide*, and *Traditional Chinese Solaris User's Guide*.

The Unicode locales can use the enhanced `mp(1)` printing filter to print text files. `mp(1)` prints flat text files written in UTF-8 using various Solaris system and printer resident fonts (such as bitmap, Type1, TrueType) depending on the script. The output is standard PostScript. For more information, refer to the `mp(1) man` page.

The Unciode locale supports various MIME character sets in `dtmail`, including various Latin, Greek, Cyrillic, Thai, and Asian character sets. Some of the example character sets are: ISO-8859-1 ~ 10, 13, 14, 15, UTF-8, UTF-7, UTF-16, UTF-16BE, UTF-16LE, Shift_JIS, ISO-2022-JP, EUC-KR, ISO-2022-KR, TIS-620, Big5, GB2312, KOI8-R, KOI8-U, and ISO-2022-CN. With this support, users can send and receive email messages encoded in MIME character sets from almost any region in the world. `dtmail` automatically decodes e-mail by recognizing the MIME character set and content transfer encoding in the message. The sender specifies the MIME character set for the recipient mail user agent.



To: root

Subject: Test message こうぇで上乗修 가나다 ثثه تتتعامبي

Cc:

```
přislibů smrti. Pedro Almodóvar balancuje mezi černou krimi a estetickým
melodramatem.

Tדי דן ש אדקא קעודדשצ שכסר עקמסטרקקק.

єХЭСдЩЫС Web-гСъдЯз з Newman Search ЯЧвСЮЩоХЮС ЩгЫЬРоЩдХЬиЮЯ
ЫЯЭаиРдХвСЭЩ, щЯЮЮЙЭЩ дХШЮЯЬЯЧЩ6ЭЩ

Το Τμήμα Πληροφορικής είναι ένα σχετικά νέο τμήμα στο Πανεπιστήμιο Αθηνών.
Ιδρύθηκε αρχικά ως Διατμηματικό Πρόγραμμα Σπουδών ανάμεσα στο Μαθηματικό και

人类进入信息时代，信息越来越成为社会生活中最重要的部分，信息服务产业越来越多，

計画に参加を検討しているのは、日本ＳＵＮのほか、住友、あさひ、東海、大和の
都市銀行四行と三菱信託銀行、スルガ銀行、千葉銀行、シティバンク、大和証券、
安田火災海上、日本生命保険、第一生命保険、全国信用金庫協会の計十三金融機関。

기상청은 또 "올 봄에는 평균기온이 예년보다 2~3도 가량 높은 영상 10~13도의
```

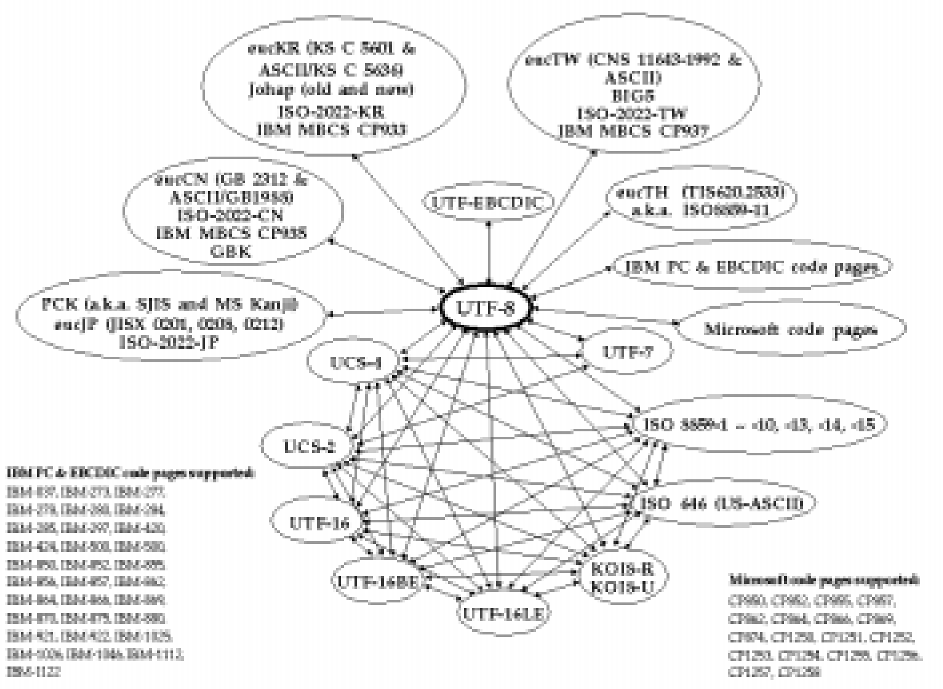
```
한글 전자우편 실험 첫번째
한글 및 漢字 email 전송
전체 셋중 마지막 메세지
上海市东方网络信息中心 电子
电子信箱 (E-mail) Testing
沪邮电管[1996]通字第58号
katakana: カタカナ, hiragana
Снкщддшс уьфщд ьуишфпу иутеш
Γρεεκ εμαιλ τηισ ρτιμε
Mixture of characters ŁÆ
(알림) 전박사님자택약도
```

```
息越来越成为社会生活中最重要的
络的开通，作为国际互联网络的节
息高速公路。

いるのは、日本ＳＵＮのほか、住
託銀行、スルガ銀行、千葉銀行、
命保険、第一生命保険、全国信用
はサンクス、サークルK、ミニス
の昭和シェル石油も参加を検討し

는 평균기온이 예년보다 2~3도
고, 강수량도 평년(2백4~3백76m
다. 현재의 포근한 날씨는 서태
고압대의 따뜻한 공기 덩어리의 영향때문.
```

Figure 3–4 Multiple character sets in `dtmail`

3.2 Codeset Conversion

The Solaris operating environment locale supports enhanced code conversion among the major codesets of several countries. Figure 3–5 shows the codeset conversions between UTF-8 and many other codesets.

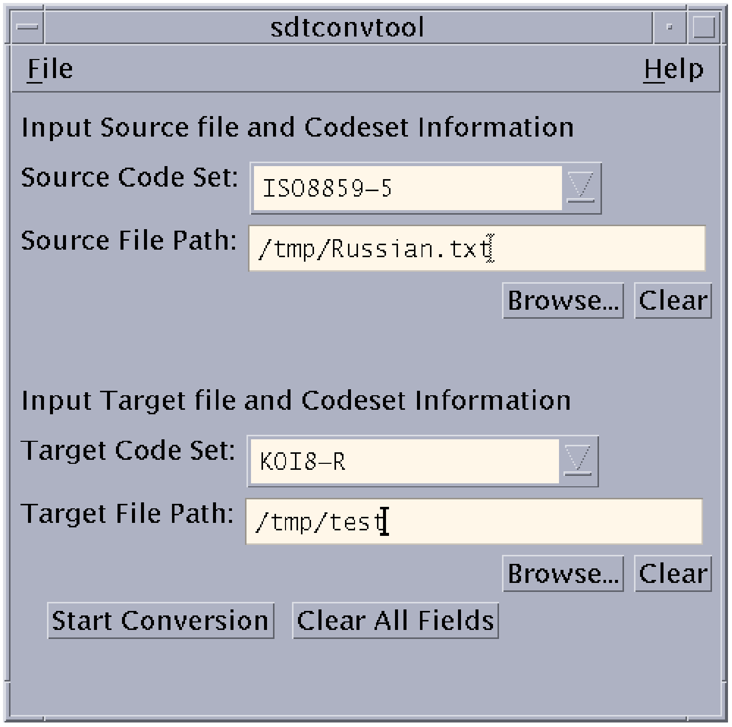

eucKR (KS C 5601 & ASCII/KS C 5636)
Johap (old and new)
ISO-2022-KR
IBM MBCS CP933
eucTW (CNS 11643-1992 & ASCII)
BIG5
ISO-2022-TW
IBM MBCS CP937
eucCN (GB 2312 & ASCII/GB1988)
ISO-2022-CN
IBM MBCS CP935
GBK
UTF-EBCDIC
eucTH (TIS620.2533) a.k.a. ISO8859-11
IBM PC & EBCDIC code pages
PCK (a.k.a. SJIS and MS Kanji)
eucJP (JISX 0201, 0208, 0212)
ISO-2022-JP
UTF-8
Microsoft code pages
UCS-4
UTF-7
ISO 8859-1 ~ -10, -13, -14, -15
UCS-2
ISO 646 (US-ASCII)
UTF-16
KOI8-R
KOI8-U
UTF-16BE
UTF-16LE

IBM PC & EBCDIC code pages supported:
IBM-037, IBM-273, IBM-277,
IBM-278, IBM-280, IBM-284,
IBM-285, IBM-297, IBM-420,
IBM-424, IBM-500, IBM-580,
IBM-850, IBM-852, IBM-855,
IBM-856, IBM-857, IBM-862,
IBM-864, IBM-866, IBM-869,
IBM-870, IBM-875, IBM-880,
IBM-921, IBM-922, IBM-1025,
IBM-1026, IBM-1046, IBM-1112,
IBM-1122

Microsoft code pages supported:
CP850, CP852, CP855, CP857,
CP862, CP864, CP866, CP869,
CP874, CP1250, CP1251, CP1252,
CP1253, CP1254, CP1255, CP1256,
CP1257, CP1258

Figure 3–5 Unicode codeset conversions

Codesets can be converted using the `sdtconvtool` utility or the `iconv(1)` command. `sdtconvtool` detects available `iconv` code conversions and presents them in an easy-to-use format.



Input Source file and Codeset Information

Source Code Set: ISO8859-5

Source File Path: /tmp/Russian.txt

Input Target file and Codeset Information

Target Code Set: KOI8-R

Target File Path: /tmp/test

Figure 3–6 `sdtconvtool` for converting between codesets

Users can also add their own code conversions and use them in `iconv(3)` functions, `iconv(1)` command line utilities, and `sdtconvtool(1)`. For more information on user-extensible, user-defined code conversions, refer to the `geniconvtbl(1)` and `geniconvtbl(4)` `man` pages.

Developers can use `iconv(3)` to access the same functionality. This includes conversions to and from UTF-8 and many ISO-standard codesets, including UCS-2, UCS-4, UTF-7, UTF-16, KO18-R, Japanese EUC, Korean EUC, Simplified Chinese EUC, Traditional Chinese EUC, GBK, PCK (Shift JIS), BIG5, Johap, ISO-2022-JP, ISO-2022-KR, and ISO-2022-CN.

For a detailed listing of the supported code conversions, see Appendix A, *Codeset Conversions*.

3.3 European Unicode Locales

In the Solaris 8 operating environment, five European Unicode locales offer the same level of support as `en_US.UTF-8` with modifications for language and cultural data.

The five European Unicode locales are:

- `fr_FR.UTF-8` (French)
- `de_DE.UTF-8` (German)
- `it_IT.UTF-8` (Italian)
- `es_ES.UTF-8` (Spanish)
- `sv_SE.UTF-8` (Swedish)

Each locale contains the same feature set as `en_US.UTF-8` and regional definitions for numeric notation, date and time, currency, and translated text messages.

The following additional five European locales support the Euro currency symbol and monetary formatting conventions:

- `fr_FR.UTF-8@euro` (French with euro monetary convention)
- `de_DE.UTF-8@euro` (German with euro monetary convention)
- `it_IT.UTF-8@euro` (Italian with euro monetary convention)
- `es_ES.UTF-8@euro` (Spanish with euro monetary convention)
- `sv_SE.UTF-8@euro` (Swedish with euro monetary convention)

Note - All Unicode locales (including en_US.UTF-8 and Asian Unicode locales) support input and output of the new euro currency *symbol.*

Figure 3–7 Euro currency symbol

3.4 Asian Unicode Locales

The Solaris 8 operating environment also supports four Unicode locales with the same scope as `en_US.UTF-8` and the European Unicode locales, with the necessary language and cultural modifications:

- `ja_JP.UTF-8` (Japanese)
- `ko_KR.UTF-8` (Korean)
- `zh_CN.UTF-8` (Simplified Chinese)
- `zh_TW.UTF-8` (Traditional Chinese)

Each Asian Unicode locale is tailored to the Asian customer's needs. For example, the Japanese Unicode locale supports additional characters from JIS X0212-1990 at the presentation layer. All existing native Asian input methods and systems are also transparently supported.

3.5 Unicode Font Resources

The Unicode Standard, Version 3.0 contains 49,194 characters from the world's scripts, with over 25,000 ideographic characters for Chinese, Japanese, and Korean. The font resources representing these characters, however, are not always one to one—some Unicode code points associate different, multiple glyphs, enabling specific code points to be rendered correctly based upon their context. For example, in Asian languages, the Unified han glyphs are written and displayed differently in Simplified Chinese, Traditional Chinese, Japanese kanji, and Korean hanja ideographs.

To manage these difficulties, the Solaris operating environment contains an output method combining existing fonts to form a Unicode font set, instead of providing a single Unicode font. The Solaris 8 operating environment supports the following range of scripts:

- English / European
- Greek, Turkish, Cyrillic
- Arabic, Hebrew, Thai
- Simplified Chinese, Traditional Chinese, Japanese, Korean

For European scripts, there is a one-to-one mapping between Unicode characters and corresponding glyphs. For Complex Text Layout language text (Arabic, Hebrew, Thai), the Solaris Universal Multiscript Layout Engine pre-processes the text (right-to-left swapping, contextual analysis, and so on) before rendering the associated glyphs.

For Asian characters, the Solaris operating environment output methods provide dynamic remapping of the font and glyph index according to the locale definition. Each locale contains a font table with mapping mechanisms specifying which font and glyph to use for each character code. The mechanism remaps the Unicode code point values to existing Chinese, Japanese, and Korean fonts and glyph index pairs. A locale administrator can define the sort priority among fonts. For example, the mechanism may search the Simplified Chinese fonts for the appropriate glyph and then search the Traditional Chinese fonts, and so on.

CHAPTER 4

Technical Considerations

4.1 Internationalized Applications with Unicode

The Unicode codeset enables developers to write applications that support multiple scripts simultaneously. The base language script and one or more additional scripts, depending on the Unicode locale, can be input, displayed, and printed. Distributed applications within network environments can also provide individual users access to different language environments simultaneously.

By itself, an application using Unicode is not fully internationalized. For example, if an application customizes data handling for Unicode directly, it needs to provide codeset converters as wrappers to support a codeset other than Unicode. This approach is *direct Unicode localization*—not internationalization. With direct localization, developers may localize an application that duplicates or conflicts with the localization provided by the operating system. In addition, an application may assume that all characters are represented in two-octet cells, which conflicts with UTF-8.

To properly internationalize an application, use the following guidelines:

- Avoid direct access with Unicode. (This is a task of the platform's internationalization framework.)
- Use the POSIX model for multibyte and wide-character interfaces. See Section 4.2, *Unicode Application Interfaces*.
- Only call APIs that the internationalization framework provides for language and cultural-specific operations. All POSIX, X11, Motif, and CDE interfaces are available to Unicode locales.

- Remain codeset independent.

4.2 Unicode Application Interfaces

When internationalizing applications for Unicode, developers should use the POSIX or X Window model. These models define two sets of interfaces—multibyte and wide character—without specifying the encoding methods.

Standard multibyte codesets contain characters of varying widths; from one to several bytes. Characters are represented in minimal storage space, with the fewest number of bytes possible. Because multibyte codesets contain characters of varying widths, they are not conveniently processed by standard functions.

The Unicode codeset provides the necessary format for both multibyte and wide-character representation. In the Solaris operating environment Unicode locales, multibyte interfaces use UTF-8 character set representation and wide-character interfaces use UCS-4 representation.

4.3 Font Resources

Properly internationalized applications require only a few changes to run properly in the Solaris operating environment Unicode locales. One required change is to set the proper resource definitions for font sets (`FontSet`) or font list (`XmFontList`) in the application's resource file.

The `en_US.UTF-8` locale supports the following set of font character sets as the `FontSet`:

- ISO 8859-1 (Latin-1)
- ISO 8859-2 (Latin-2)
- ISO 8859-4 (Latin-4)
- ISO 8859-5 (Latin / Cyrillic)
- ISO 8859-7 (Latin / Greek)
- ISO 8859-9 (Latin-5)
- ISO 8859-15 (Latin-9)
- ISO 8859-6 based one (Arabic)
- ISO 8859-8 (Hebrew)
- TIS 620-2533 based one (Thai)

- BIG5 (Traditional Chinese)
- GB 2312-1980 (Simplified Chinese)
- JIS X0201-1976, JIS X0208-1983 (Japanese)
- KS C 5601-1992 Annex 3 (Korean)

4.4 Setting Resource Definitions

To create a font set for an application, the resource definition should contain the complete set of fonts supported by the Unicode locale. For example:

```
fs = XCreateFontSet(display,
"-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-1,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-2,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-4,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-5,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-6,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-7,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-8,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-9,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-15,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-big5-1,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-gb2312.1980-0,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-jisx0201.1976-0,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-jisx0208.1983-0,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-ksc5601.1992-3,
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-tis620.2533-0",
 -dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-unicode-fontspecific",
  &missing_ptr, &missing_count, &def_string);
```

Or, more simply:

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

The `XmFontList` resource definition of an application should also include all fonts for every character set supported by the locale. For example:

```
!
! This is an example XmNFontList definition for en_US.UTF-8 locale:
*fontList:\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-1;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-2;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-4;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-5;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-6;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-7;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-8;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-9;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-iso8859-15;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-big5-1;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-gb2312.1980-0;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-jisx0201.1976-0;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-jisx0208.1983-0;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-ksc5601.1992-3;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-tis620.2533-0;\
-dt-interface system-medium-r-normal-s*utf*-*-*-*-*-*-*-*-unicode-fontspecifc:
```

Or, more simply:

```
!
! This is an example XmNFontList definition for en_US.UTF-8 locale:
*fontList: -dt-interface system-medium-r-normal-s*utf*:
```

APPENDIX **A**

Codeset Conversions

A.1 Codeset Conversions

The following table provides a detailed listing of the supported code conversions.

Note - Unicode* includes all of the following codesets: UTF-8, UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4BE, UCS-4LE, UTF-16, UTF-16BE, UTF-16LE.

ISO 8859 codesets can also be referenced without the ISO prefix; for example, ISO 8859–1 = 8859–1.

TABLE A–1 Supported code conversions

Code	Code	Description
Unicode*	ISO 646	Unicode* <—> ISO 646 (ASCII)
Unicode*	ISO 8859-1	Unicode* <—> ISO 8859-1 (Latin-1)
Unicode*	ISO 8859-2	Unicode* <—> ISO 8859-2 (Latin-2)
Unicode*	ISO 8859-3	Unicode* <—> ISO 8859-3 (Latin-3)
Unicode*	ISO 8859-4	Unicode* <—> ISO 8859-4 (Latin-4)
Unicode*	ISO 8859-5	Unicode* <—> ISO 8859-5 (Cyrillic)
Unicode*	ISO 8859-6	Unicode* <—> ISO 8859-6 (Arabic)

TABLE A–1 Supported code conversions *(continued)*

Code	Code	Description
Unicode*	ISO 8859-7	Unicode* <—> ISO 8859-7 (Greek)
Unicode*	ISO 8859-8	Unicode* <—> ISO 8859-8 (Hebrew)
Unicode*	ISO 8859-9	Unicode* <—> ISO 8859-9 (Latin-5)
Unicode*	ISO 8859-10	Unicode* <—> ISO 8859-10 (Latin-6)
Unicode*	ISO 8859-13	Unicode* <—> ISO 8859-13
Unicode*	ISO 8859-14	Unicode* <—> ISO 8859-14
Unicode*	ISO 8859-15	Unicode* <—> ISO 8859-15
Unicode*	KOI8-R, KO18–U, koi8–r, koi8–u	Unicode* <—> KOI8-R, KO18–U, koi8–r, koi8–u (Cyrillic)
UTF-7	UCS-2, UCS-4, UTF-8	UTF-7 <—> UCS-2, UCS-4, UTF-8
UTF-8	UCS-2, UCS-4, UTF-16	UTF-8 <—> UCS-2, UCS-4, UTF-16
UTF-8	UCS-2BE, UCS-2LE, UCS-4BE, UCS-4LE, UTF-16BE, UTF-16LE	UTF-8 <—> UCS-2BE, UCS-2LE, UCS-4BE, UCS-4LE, UTF-16BE, UTF-16LE
UCS-4, UCS-4BE, UCS-4LE	UCS-2, UCS-2BE, UCS-2LE, UTF-16, UTF-16BE, UTF-16LE	UCS-4, UCS-4BE, UCS-4LE <—> UCS-2, UCS-2BE, UCS-2LE, UTF-16, UTF-16BE, UTF-16LE
UTF-8	UTF-EBCDIC	UTF-8 <—> UTF-EBCDIC
UTF-8	IBM-037, -273, -277, -278, -280 -284, -285, -297, -420 -424, -500, -850, -852 -855, -856, -857, -862 -864, -866, -869, -870 -875, -880, -921, -922 -1025, -1026, -1046, -1112, -1122	UTF-8 <—> various IBM code pages (PC and EBCDIC)

TABLE A–1 Supported code conversions *(continued)*

Code	Code	Description
UTF-8	CP850, CP852, CP855, CP857, CP862, CP864, CP866, CP869, CP874, CP1250, CP1251, CP1252, CP1252, CP1253, CP1254, CP1255, CP1256, CP1257, CP1258	UTF-8 <—> various Microsoft code pages
UTF-8	eucJP	UTF-8 <—> Japanese EUC (JIS X0201-1976, JIS X0208-1983 and JIS X0212-1990)
UTF-8	PCK	UTF-8 <—> Japanese PC Kanji (a.k.a. SJIS)
UTF-8	ISO-2022-JP	UTF-8 <—> Japanese MIME charset
UTF-8–Java	eucJP	UTF-8–Java to Japanese EUC (JIS X0201-1976, JIS X0208-1983 and JIS X0212-1990)
UTF-8–Java	PCK	UTF-8–Java to Japanese PC Kanji (a.k.a. SJIS)
UTF-8–Java	ISO-2022-JP.RFC1468	UTF-8–Java to Japanese MIME charset (one-way conversion)
UTF-8	ko_KR-euc	UTF-8 <—> Korean EUC (KS C 5636 and KS C 5601-1987)
UTF-8	ko_KR-johap	UTF-8 <—> Korean Johap (of KS C 5601-1987)
UTF-8	ko_KR-johap92	UTF-8 <—> Korean Johap (of KS C 5601-1992)
UTF-8	ko_KR-iso2022-7	UTF-8 <—> Korean MIME charset (ISO-2022-KR)
UTF-8	ko_KR-cp933	UTF-8 <—> IBM MBCS CP933 ko_KR-euc
UTF-8	gb2312	UTF-8 <—> Simplified Chinese EUC (GB 1988-1980 and GB 2312-1980)
UTF-8	iso2022	UTF-8 <—> Simplified Chinese MIME charset (ISO-2022-CN)
UTF-8	GBK	UTF-8 <—> Simplified Chinese GBK
UTF-8	zh_TW-euc	UTF-8 <—> Traditional Chinese EUC (CNS 11643-1992)

TABLE A–1 Supported code conversions *(continued)*

Code	Code	Description
UTF-8	zh_TW-big5	UTF-8 <—> Traditional Chinese Big5
UTF-8	zh_TW-iso2022-7	UTF-8 <—> Traditional Chinese MIME charset (ISO-2022-TW)
UTF-8	zh_TW-cp937	UTF-8 <—> IBM MBCS CP937