



Solaris Internationalization Guide for Developers

Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A. 415-960-1300

Copyright 1997 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, SunSoft, SunDocs, SunExpress, JumpStart, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1997 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, SunSoft, SunDocs, SunExpress, JumpStart, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPENDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Please
Recycle



Adobe PostScript

Contents

Preface	xv
Who Should Use This Book	xv
Organization and Summary of this Book	xvi
Related Books	xvi
Ordering Sun Documents	xvii
Typographic Conventions	xviii
Shell Prompts in Command Examples	xviii
1. Solaris Internationalization Overview	1
New Internationalization Features in Solaris 2.6	1
Internationalization and Localization	2
Basic Steps in Internationalization	2
What Is a Locale?	3
Full vs. Partial Locales	3
Locales in Solaris	4
Locale Categories	4
Using Locale Categories for Localization	5
Time Formats	5
Date Formats	6

	Numbers	7
	Currency	8
	Word and Letter Differences	9
	Codesets for x86	11
	Keyboard Differences	12
	Other Differences	12
	Punctuation	12
	Symbols	12
	Measurements	13
	Gender	13
	Titles and Addresses	13
	Paper Sizes	13
	<i>Creating Worldwide Software: The Book</i>	14
	Overview	14
2.	Contents of the Base Solaris Product	17
	Summary of the Base Product	17
	Core Set of Locales	18
	Extended Set of Locales	19
	New Unicode Locale: <code>en_US.UTF-8</code>	20
	New User Locales in Base Solaris	20
	Multiple Key Compose Sequences for New Locales	21
	Keyboard Mapping for Greek and Russian Scripts	22
	New Keyboard Support in Solaris 2.6	22
	Changing Between Keyboards on SPARC	22
	Changing Between Keyboards on x86	23
	New Locales in the Base Installation	24
	Using Jumpstart	24

3. Contents of the Localized Solaris 2.6 Products	25
The European Localized Solaris 2.6 Product	25
Font Formats	29
The Asian Localized Solaris 2.6 Products	30
Korean	31
Chinese: Simplified and Traditional	32
Japanese	34
4. Overview of UTF-8	41
The Universal Transformation Format	41
System Environment	42
Code Conversions	47
Script Selection and Input Modes	50
Printing	61
Programming Environment	62
FontSet Used with UTF-8	62
5. Installation	65
Adding Packages	65
Installing Software From a Mounted CD	67
Installing Software From a Remote Package Server	67
Installing the Localization Product	68
European Package	69
French Files	69
German Files	70
Italian Files	71
Spanish Files	72
Swedish Files	73
Eastern European Files	73

	Detailed Descriptions of European Files	74
	European Codesets	79
	European Font Packages	79
	Asian Packages	80
	Description of General Packages	84
	Asian Localization Packages Disk Space	100
6.	Internationalization Framework in Solaris 2.6	101
	Codeset Independence Support	101
	The CSI Approach	102
	CSI-enabled Commands	102
	Solaris 2.6 CSI-enabled Libraries	103
	Locale Database	104
	Process Code Format	104
	Dynamically Linked Applications	104
	libw and libintl	106
	ctype Macros	107
	Internationalization APIs in libc	107
	genmsg Utility	112
7.	Writing Internationalized Code	115
	Linking	115
	Text and Codesets	115
	Call <code>setlocale()</code>	115
	Make Software 8-bit Clean	116
	Watch for Sign Extension Problems	117
	Use <code>ctype</code> Library Routines	119
	Formats	119
	Time and Date Formats	120

Currency and Number Formats	121
Collation	122
Replace <code>strcmp()</code> with <code>strcoll()</code>	122
Messaging for Program Translation	124
Messaging Using <code>catgets()</code>	125
Locating Message Catalogs	125
Using <code>catgets()</code>	127
Create the Source Message Catalog	128
Translate the Source Message Catalog	131
Generate the Binary Message Catalogs	131
Messaging Using <code>gettext()</code>	132
Locating Message Catalogs	133
Surround Strings with <code>gettext()</code>	134
Create the Source Message Catalog	135
Create the Binary Message Catalog	136
Problem Areas	136
Other Programming Languages	141
Summary	142
8. X/DPS	143
Localization Resource Category	144
Information on Language Interpreters	144
9. Desktop Environments	145
Overview	145
Locales	147
Integrating Fonts	147
Input Methods	147
Internationalization and CDE	148

Matching Fonts to Character Sets	148
Storage of Localized Text	148
Xlib Dependencies	149
Message Guidelines	149
Internationalization and Distributed Networks	149
Mail Interchange	150
OpenWindows	150
10. Printing	151
Localization Printing Support Under Solaris 2.6	151
European Printing Support	151
Asian Printing Support	152
Index	155

Figures

- FIGURE 4-1 English Input Mode 50
- FIGURE 4-2 Cyrillic Input Mode 58
- FIGURE 4-3 Russian Keyboard Layout 59
- FIGURE 4-4 Greek Input Mode 59
- FIGURE 4-5 Greek Keyboard Layout (European Keyboard) 60
- FIGURE 4-6 Greek Keyboard Layout (UNIX Keyboard) 60

Tables

TABLE P-1	Typographic Conventions	xviii
TABLE P-2	Shell Prompts	xviii
TABLE 1-1	International Time Formats	5
TABLE 1-2	International Date Formats	6
TABLE 1-3	International Numeric Conventions	7
TABLE 1-4	International Monetary Conventions	8
TABLE 1-5	Common International Page Sizes	13
TABLE 2-1	Core Set of Locales in <code>SUNWploc</code> and <code>SUNWplow</code>	18
TABLE 2-2	Extended Set of Locales in <code>SUNWploc1</code> and <code>SUNWplow1</code>	19
TABLE 2-3	New User Locales Included in Solaris 2.6	20
TABLE 2-4	Layouts for Type 4 Keyboards	23
TABLE 2-5	New Locales Offered in Installation	24
TABLE 3-1	European 2.6 Locales	25
TABLE 3-2	New Eastern European Locales in Solaris 2.6	27
TABLE 3-3	<code>iconv</code> Support for Major Codesets	28
TABLE 3-4	Summary of Asian Locales	31
TABLE 3-5	Codeset Conversions Supported for Korean <code>ko</code> , <code>ko.UTF-8</code>	32
TABLE 3-6	Codeset Conversions for Simplified Chinese	33
TABLE 3-7	Codeset Conversions for Traditional Chinese	33

TABLE 3-8	Japanese Input Systems	34
TABLE 3-9	Japanese TrueType Fonts	35
TABLE 3-10	Japanese F3 Fonts	35
TABLE 3-11	Japanese Bitmap Fonts	35
TABLE 3-12	<code>iconv</code> Conversion Support	36
TABLE 3-13	Japanese-specific Commands	37
TABLE 4-1	STREAMS Modules Supported by <code>en_US.UTF-8</code>	43
TABLE 4-2	Available Code Conversions in <code>en_US.UTF-8</code>	48
TABLE 4-3	Common Latin-1 Compose Sequences	50
TABLE 4-4	Common Latin-2 Compose Sequences	54
TABLE 4-5	Common Latin-4 Compose Sequences	56
TABLE 4-6	Common Latin-5 Compose Sequences	58
TABLE 5-1	Pan-European Files for Localization and Windowing	69
TABLE 5-2	French Files for Localization and Windowing	69
TABLE 5-3	German Files for Localization and Windowing	70
TABLE 5-4	Italian Files for Localization and Windowing	71
TABLE 5-5	Spanish Files for Localization and Windowing	72
TABLE 5-6	Swedish Files for Localization and Windowing	73
TABLE 5-7	European Files for Localization and Windowing	73
TABLE 5-8	European Package Descriptions	74
TABLE 5-9	Font Packages in Solaris 2.6	79
TABLE 5-10	Asian Package for Localization and Windowing	80
TABLE 5-11	Korean Package for Localization and Windowing	81
TABLE 5-12	Chinese Package for Localization and Windowing	82
TABLE 5-13	Japanese Package for Localization and Windowing	83
TABLE 5-14	Packages	84
TABLE 5-15	Korean Package	85
TABLE 5-16	Chinese Package	86

TABLE 5-17	Japanese Package	87
TABLE 5-18	ko Locale	90
TABLE 5-19	ko.UTF-8 Locale	91
TABLE 5-20	zh Locale	92
TABLE 5-21	zh_TW Locale	92
TABLE 5-22	zh_TW.BIG5 Locale	93
TABLE 5-23	ja/ja_JP.PCK Common Packages	94
TABLE 5-24	ja Locale	95
TABLE 5-25	ja_JP.PCK Locale	96
TABLE 5-26	CDE Packages	97
TABLE 5-27	Approximate Disk Space in Megabytes (MB) Required for Software Groups (SPARC)	100
TABLE 5-28	Approximate Disk Space in MB Required for Software Groups (x86)	100
TABLE 6-1	CSI-enabled Commands in Solaris 2.6	103
TABLE 6-2	Stub Entry Points in libw and libintl	106
TABLE 6-3	internationalization APIs in libc	108
TABLE 7-1	Library Routines for Codeset Independence	119
TABLE 10-1	prolog.ps Fonts	152
TABLE 10-2	Japanese Printer Support	153

Preface

The *Solaris Internationalization Guide for Developers* describes new internationalization features in Solaris 2.6. It contains important information on how to use Solaris 2.6 to build global software products that support various languages and cultural conventions.

Specifically, this guide contains:

- Guidelines and tips for developers on how to use Solaris 2.6 to write applications for international markets.
- An overall view of internationalization topics that apply to various layers within the Solaris environment.
- Pointers to more detailed documentation.

Where appropriate, this guide points you to other books in the documentation set that contain additional or more detailed information on internationalization features in this release.

Who Should Use This Book

This book is intended for software developers who want to design global products and applications for the Solaris 2.6 environment. Readers will find the latest Sun-specific information pertaining to this release.

This book assumes knowledge of the C programming language, and a few chapters discuss X11[®] window system toolkits.

All operating system information pertains to SunOS[™] 5.6. The hardware platforms covered are SPARC[®] and Intel x86. For the most part, support for these architectures should be identical, but a note appears when this is not the case.

Organization and Summary of this Book

The chapters in this book are organized as follows:

- **Chapter 1, “Solaris Internationalization Overview,”** provides an overview of the localized products available on the base Solaris release, the European localized release, and the Asian localized releases.
- **Chapter 2, “Contents of the Base Solaris Product,”** describes the content of the Solaris 2.6 base product.
- **Chapter 3, “The European Localized Solaris 2.6 Product,”** describes Codeset Independence (CSI) support for Extended Unix Code (EUC) and non-EUC codesets.
- **Chapter 4, “Overview of UTF-8,”** covers the system environment, code conversions, script selection, printing, and the programming environment.
- **Chapter 5, “Installation,”** describes the procedures for installing the localization packages.
- **Chapter 6, “Internationalization Framework in Solaris 2.6,”** contains details about the internationalization features incorporated into this release.
- **Chapter 7, “Writing Internationalized Code,”** is a detailed look at the procedures in creating a localized version: codesets, formats, collation, and messaging.
- **Chapter 8, “X/DPS,”** covers the X Windows system’s extension with the X Display PostScript system.
- **Chapter 9, “Desktop Environments,”** covers the Solaris desktop environments: the Common Desktop Environment (CDE) and OpenWindows. The section on CDE has an overview of the application internationalization process, including locale management, localized resources, font management, localized text tasks, interclient communication, and internationalized functions.
- **Chapter 10, “Printing,”** covers printing support under Solaris 2.6, with specific information for European and the Asian printing.

Related Books

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

Common Desktop Environment: Internationalization Programmer’s Guide. Mountain View, California, SunSoft Press, 1996. The CDE documentation set can be ordered by title through SunExpress. The CDE programmer’s guide is also part of the CDE

Developer's AnswerBook set that is shipped on the Solaris documentation CD. Available through the SunDocs program (see page xvii). Contains information on locale management, font management, distributed networks, User Interface Language (UIL), Xt, and Xlib dependencies. See Chapter 9, "Desktop Environments," for a summary of contents.

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

OSF/Motif Programmer's Reference, Release 1.2. Englewood Cliffs, New Jersey, Prentice-Hall, 1992. The Open Software Foundation's (OSF) *Reference* is the collection of reference pages to OSF/Motif commands, functions, toolkit, window manager, user interface language commands, and functions.

PostScript Language Reference Manual, Second Edition. Adobe Systems Inc., Addison-Wesley, 1990. The standard reference work for PostScript™ covers the fundamentals of PostScript as a device-independent printing language.

PostScript Language Reference Manual Supplement. Adobe Systems Inc., 1994.

Programming the Display PostScript System with X. Reading, Mass., Adobe Systems Inc., Addison-Wesley, 1993. For application developers working with X Windows and Display PostScript to produce information for the screen display and the printer output.

OLIT Reference Manual. Sun Microsystems, 1994.

XView Developer's Notes. O'Reilly & Associates, 1992.

Ordering Sun Documents

The SunDocs program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals from SunDocs.

For a list of documents and how to order them, see the catalog section of the SunExpress™ Internet site at <http://www.sun.com/sunexpress>.

Typographic Conventions

TABLE P-1 describes the typographic conventions used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. machine_name% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<pre>machine_name% su Password:</pre>
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.

Shell Prompts in Command Examples

TABLE P-2 shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

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

Solaris Internationalization Overview

Solaris 2.6 includes full Unicode 2.0 support, as defined in ISO-10646, in selected locales. Solaris 2.6 is a major release for Sun's international markets. It includes a number of new features for Asian customers and significantly expands language support for Eastern Europe and the Baltic States.

New Internationalization Features in Solaris 2.6

- Unicode 2.0 support
 - Unicode 2.0 supported through UTF-8 in English and Korean locales
 - UTF-8 locales support multi-script input and output for all European locales and Korean
- Codeset Independence
- Expanded language coverage
 - Ten new locales added for Eastern Europe, Russia, Greece, Turkey, and the Baltic States
 - Additional input methods provided for the Japanese locale (Wnn6 and ATOK8)
 - Easy-to-use font administration tool for adding and managing fonts
- Improved PC data interoperability
 - Popular Asian PC file encoding (PC-Kanji and Big5)
 - TrueType font support in all versions and TrueType fonts included in Asian versions
 - Utilities provided for easy two-way conversion of PC files to UNIX encoding

Internationalization and Localization

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. International software can be developed using interfaces that modify program behavior at run time in accordance with specific cultural requirements. Localization involves establishing on-line information to support a language or region, called a *locale*.

Unlike software that must be completely rewritten before it can work with different native languages and customs, internationalized software does not require rewriting. It can be ported from one locale to another without change. The Solaris system is internationalized, providing the infrastructure and interfaces you need to create internationalized software. Chapter 3, “Contents of the Localized Solaris 2.6 Products” and Chapter 4, “Overview of UTF-8” describe what facilities are available and how to use them.

Internationalization and localization are different procedures.

- Internationalization is the process of making software that is independent of any locale. It can then be easily adapted to specific locales.

The following localized products are available in Solaris 2.6:

- English Solaris
- European Solaris (German, French, Spanish, Swedish, Italian)
- Simplified Chinese Solaris for the People’s Republic of China
- Traditional Chinese Solaris for Taiwan
- Japanese Solaris
- Korean Solaris

Basic Steps in Internationalization

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

- Use the interfaces described in this book to create software whose environment can be modified dynamically without the necessity of recompiling the software.
- Separate software into executable and messages. The messages include all printable and displayable messages that the user sees. Keep the message strings in a message database.

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

Locales are not the same as a language. A language may contain various regions: for example, French is spoken in France and Canada, but each country has different ways of displaying monetary and time information.

To use a localized version of a product, the user sets the environment variables (described at “Locale Categories” on page 4). The product then displays the user messages in their translated form. Date, time, currency, and other information is formatted and displayed according to locale-specific conventions.

What Is a Locale?

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

The locale consists of a number of categories for which there are language-dependent formatting or other specifications. A program’s locale defines its codesets, date and time formatting conventions, monetary conventions, decimal formatting conventions, and collation order.

A locale name is comprised of language, territory, and possibly codeset, although territory is dropped when not needed. Codeset is usually assumed. For example, German is `de`, an abbreviation for Deutsch, while Swiss German is `de_CH`, CH being an abbreviation for Confederation Helvetica.

Generally the locale name is specified by the `LANG` environment variable. Locale categories are subordinate to `LANG`, but may be set separately, in which case they override `LANG`. If `LC_ALL` is set, it overrides not only `LANG`, but all the separate locale categories as well.

Full vs. Partial Locales

A full Solaris locale has all of the listed functions and the localized system messages in that language. The German `de` locale is a full locale. A German user will see all system messages in German.

Partial locales have the listed functions but they don’t provide localized messages. For example, the Russian `ru` locale can process input, output, sorting, and so on, but it does not have localized messages in Russian. For this reason it is a partial locale.

Some partial locales do use English messages because there may be a full locale with the localized messages. For example, the `de_AT` is a partial locale for Austria. Austrians speak German, but use a different currency. The Austrian locale is a subset of the German `de` locale. It displays messages in German and currency in Austrian schillings instead of German marks.

Locales in Solaris

Different cultures use different conventions for writing the date, the time, numbers, currency, delimiting words and phrases, and quoting material.

A locale defines the behavior of a program at runtime according to a language or cultural region's conventions. Throughout the system, a locale will determine the behavior of the following:

- Encoding and processing of text data
- Identifying the language and encoding of resource files and their text values
- Rendering and layout of text strings
- Interchanging text that is used for interclient text communication
- Selecting the input method (that is, which codeset will be generated) and the processing of text data
- Encoding and decoding for interclient text communication
- Font and icon files that are culturally specific
- Actions and file types
- User Interface Definition (UID) files
- Date and time formats
- Numeric formats
- Monetary formats
- Collation order
- Format for informative and diagnostic messages and interactive responses

The CDE separates language and culture-dependent information from the application and saves it outside the application.

By separating the language and culture-dependent information from the application, the developer does not need to translate, rewrite, or recompile the application for each market. The only requirement to enter a new market is to localize the external information to the local language and customs.

Locale Categories

The locale categories are as follows:

- `LC_CTYPE`
A category which controls the behavior of character handling functions.
- `LC_TIME`
This category specifies date and time formats, including month names, days of the week, and common full and abbreviated representations.
- `LC_MONETARY`
This category specifies monetary formats. Few SunOS system commands or library routines actually use this category.
- `LC_NUMERIC`
This category specifies the decimal separator (or radix character) and the thousands separator.
- `LC_COLLATE`
This category specifies the sorting order for a locale, and string conversions required to attain this ordering.
- `LC_MESSAGES`
This category specifies the language in which the localized messages will be written.

Using Locale Categories for Localization

The localization of a product should be done in consultation with native users in that target language or region. Certain styles and information styles and formats may seem perfectly obvious and universal to the developer, but to the user, these will either look awkward, wrong, or possibly offensive. The following pages describe the elements which Solaris allows you to control and specify so that you can successfully internationalize your product.

Time Formats

TABLE 1-1 shows some of the ways to write 11:59 p.m.

TABLE 1-1 International Time Formats

Locale	Format
Canadian	23:59
Finnish	23.59

TABLE 1-1 International Time Formats (*Continued*)

Locale	Format
German	23.59 Uhr
Norwegian	Kl 23.59
U.K.	11.59 PM

Time is represented by both a 12-hour clock and a 24-hour clock—sometimes known as “railroad time.” The hour and minute separator can be either a colon (:) or a period (.).

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

Daylight Savings Time (DST) starts and ends on different dates that can vary from country to country.

Date Formats

TABLE 1-2 shows some of the date formats used around the world. Note that even within a country, there may be variations.

TABLE 1-2 International Date Formats

Locale	Convention	Example
Canadian (English)	yyyy-mm-dd	1989-08-13
Canadian (French)	yyyy-mm-dd	1989-08-13
Danish	dd/mm/yy	13/08/89
Finnish	dd.mm.yyyy	13.08.1989
French	dd/mm/yy	13/08/89
German	dd.mm.yy	13.08.89
Italian	dd.mm.yy	13.08.89
Norwegian	dd.mm.yy	13.08.89
Spanish	dd-mm-yy	13-08-89

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

Locale	Convention	Example
Swedish	yyyy-mm-dd	1989-08-13
UK-English	dd/mm/yy	13/08/89
US-English	mm-dd-yy	08-13-89

Numbers

Decimal and Thousands Separators

The United Kingdom and the United States are two of the few places in the world that use a period to indicate the decimal place. Many other countries use a comma instead. The decimal separator is also called the *radix* character. Likewise, while the U.K. and U.S. use a comma to separate thousands groups, many other countries use a period for this instead, and some countries separate thousands groups with a thin space. TABLE 1-3 shows some commonly used numeric formats.

TABLE 1-3 International Numeric Conventions

Locale	Large Number
Canadian (French)	4 294 967 295,00
Canadian (English)	4 294 967 295,00
Danish	4.294.967.295,00
Finnish	4.294.967.295,00
French	4.294.967.295,00
German	4 294 967 295,00
Italian	4.294.967.295,00
Norwegian	4.294.967.295,00
Spanish	4.294.967.295,00
Swedish	4.294.967.295,00
UK-English	4,294,967,295.00
US-English	4,294,967,295.00

Data files containing locale-specific formats will be misinterpreted when transferred to a system in a different locale. For example, a file containing numbers in a French format will not be useful to a U.K.-specific program.

List Separators

There are no particular locale conventions that specify how to separate numbers in a list. They are sometimes comma-delimited in the UK and the US, but often spaces and semicolons are used.

Currency

Currency units and presentation order vary greatly around the world. TABLE 1-4 shows monetary formats in some countries.

TABLE 1-4 International Monetary Conventions

Locale	Currency	Example
Canadian (English)	Dollar (\$)	\$1 234.56
Canadian (French)	Dollar (\$)	1 234.56\$
Danish	Kroner (kr)	kr.1.234,56
Finnish	Markka (mk)	1.234 mk
French	Franc (F)	F1.234,56
German	Deutsche Mark (DM)	1,234.56DM
Italian	Lira (L)	L1.234,56
Japanese	Yen (¥)	¥1,234
Norwegian	Krone (kr)	kr 1.234,56
Spanish	Peseta (Pts)	1.234,56Pts
Swedish	Krona (Kr)	1234.56KR
UK-English	Pound (£)	£1,234.56
US-English	Dollar (\$)	\$1,234.56

Note that local and international symbols for currency can differ. For example, the designation for the French franc is “F” in France but this is often written as “FRF” internationally to distinguish it from other francs, such as the Swiss franc or the Polynesian francs.

Be aware also that a *converted* currency amount may take up more or less space than the original amount. To illustrate: \$1,000 can become L1.307.000.

Word and Letter Differences

Word Delimiters

Usually, words are separated by a space character. In Japanese and Thai, however, there is often no delimiter between words.

Word Order

The order of words in phrases and sentences varies between languages. For instance, the order of the words “cat” and “black” in “a black cat” is reversed in the equivalent Spanish phrase, “uno gato negro.” And in French, the negatives “ne” and “pas” surround the word they negate, as in the phrase “I do not speak,” which in French is “Je ne parle pas.”

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.

Character Sets

Number of Characters

While the English alphabet contains only 26 characters, some languages contain many more characters. Japanese, for example, can contain over 40,000 characters; Chinese even more.

Western European Alphabets

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

Japanese Text

Japanese text is composed of three different scripts mixed together: Kanji ideographs derived from Chinese, and two phonetic scripts (or syllabaries), Hiragana and Katakana.

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

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

Although it is possible to avoid the use of Kanji completely, most Japanese readers find text containing Kanji easier to understand.

Korean Text

Korean is similar to Japanese in that Chinese-based ideograms, called Hanja, are mixed together with a phonetic alphabet, Hangul. Hanja is used mostly to avoid confusion when Hangul would be ambiguous.

Hangul characters are formed by combining 10 basic vowels and 14 consonants, 2 to 5 of which compose one syllable. Hangul characters are often arranged in a square like the four on a pair of dice, so that the group takes up the same space as a Hanja character.

Korean text requires over 6000 Hanja characters, plus about 96 Hangul characters.

Chinese Text

Chinese usually consists entirely of characters from the ideographic script called Hanzi. In the People's Republic of China (PRC) there are about 7000 commonly used Hanzi characters, although emerging standards number Hanzi in the tens of thousands. In the Republic of China (ROC or Taiwan) current standards require more than 13,000 characters; 6000 others have been recently standardized but are considered rare.

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

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

There are several phonetic systems for representing Chinese. In mainland 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), an extensive phonetic alphabet with unique letter forms, is often used instead.

Commercial applications, particularly those that deal with people's names, need to consider the impact of codeset expansion. Many people in the ROC have names containing characters that do not exist in any standard codeset. Space needs to be provided in unassigned codesets to deal with this issue.

Codesets for x86

The default codeset on the Solaris system for x86 is ISO-8859-1. IBM DOS 437 codeset is provided as an option in text mode; however, it is provided only at internationalization level 1. That is, if you choose to download IBM DOS 437 codeset by typing:

```
loadfont -c 437
pcmapkeys -f /usr/share/lib/keyboards/437/en_US
```

there will be no support for nonstandard U.S. date, time, currency, numbers, units, and collation. There will be no support for non-English message and text presentation, and no multibyte character support. Therefore, non-Microsoft-Windows users should use IBM DOS 437 codeset only in the default C locale.

- You must be in the text mode to download the IBM codeset, not the graphics mode.
- If you are not using the standard U.S. PC keyboard, replace `en_US` with the keyboard map related to your keyboard.
- To download the default codeset in text mode, type:
 - `loadfont -c 8859`
 - `pcmapkeys -f /usr/share/lib/keyboards/8859/en_US`
- See the `loadfont` (1) and `pcmapkeys` (1) manual pages.

Keyboard Differences

Not all characters on the US keyboard appear on other keyboards. Similarly, other keyboards often contain many characters not visible on the US keyboard. However, the Compose key can be used to produce any character in the ISO Latin-1 codeset on any keyboard that supports it.

Other Differences

Punctuation

Both the position and the type of punctuation symbols can vary between languages. In Spanish, “¿” and “¡” appear at the beginnings of sentences, while in Finnish colons (:) can occur inside words.

Symbols

Commonly used symbols in one culture often have no meaning in another culture. For example, because the common U.S. rural mailbox does not exist in other countries, it would not make a universal email icon.

Measurements

While most countries now use the metric system of measurement, the United States, parts of Canada, and the United Kingdom (albeit unofficially) still use the imperial system. The symbols for feet (') and inches (") are not understood in all countries.

Gender

The spelling of adjectives, articles, and nouns are gender-dependent in some languages. In French, for example, “un petit gamin” and “une petite gamine” both mean “a cute kid.” The first expression, however, refers to a boy, and the second expression, to a girl. Also, neuter objects in English (“a computer” for example) have gender in other languages (“un ordinateur” is a masculine noun in French).

Titles and Addresses

Mr., Miss, Mrs., and Ms. are common titles in the US but are not used in many other countries.

Address formats differ from country to country. In many countries, the postal code includes letters as well as numbers.

Paper Sizes

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

Internationalized applications should not make assumptions about the page sizes available to them. The Solaris system provides no support for tracking output page size; this is the responsibility of the application program itself.

TABLE 1-5 Common International Page Sizes

Paper Type	Dimensions	Countries
ISO A4	21.0 cm by 29.7 cm	Everywhere except US
ISO A5	14.8 cm by 21.0 cm	Everywhere except US
JIS B4	25.9 cm by 36.65 cm	Japan

TABLE 1-5 Common International Page Sizes (Continued)

Paper Type	Dimensions	Countries
JIS B5	18.36 cm by 25.9 cm	Japan
US Letter	8.5 inch by 11 inches	US and Canada
US Legal	8.5 inch by 14 inches	US and Canada

Standard paper trays distributed with LaserWriter and LaserWriter II printers support U.S. letter, U.S. legal, and A4 paper sizes. The SPARCprinter™ paper tray supports all these sizes, in addition to B5.

Creating Worldwide Software: The Book

The book *Creating Worldwide Software*, 2nd edition, by Bill Tuthill and David Smallberg (Sun Microsystems Press, 1997), is a guide to localizing for the Solaris platform. The book is recommended for developers who work with the Solaris system. See “Related Books” on page xvi for a full citation.

Overview

The book *Creating Worldwide Software* is for developers and managers who develop products for the worldwide UNIX platform, especially for the Sun Solaris system.

- **Chapter 1, “Winning in Global Markets,”** briefly shows the market potential of internationalizing your products and defines the steps of internationalization and localization.
- **Chapter 2, “Understanding Linguistic and Cultural Differences,”** shows through examples how an item will appear in various cultures.
- **Chapter 3, “Encoding Character Sets,”** describes how to encode character sets in any language.
- **Chapter 4, “Establishing Your Locale Environment,”** looks at how a user selects a locale. It leads you through the steps of creating a specific locale for your product, including formats for time, date, money, and so on.
- **Chapter 5, “Messaging for Program Translation,”** explains how to prepare your product to handle localized messages. It discusses how to create and install your translated message catalogs.
- **Chapter 6, “Displaying Localized Text,”** discusses font, user interface, and printing issues.
- **Chapter 7, “Handling Language Input,”** discusses the various input methods for various languages.

- **Chapter 8, “Working with CDE,”** explains the CDE environment and your localization.
- **Chapter 9, “Motif Programming,”** discusses how to write applications under Motif and CDE.
- **Chapter 10, “X11 Programming,”** discusses internationalization with X11.
- **Chapter 11, “Communicating Network Data,”** discusses issues in sharing and distributing data across networks.
- **Chapter 12, “Writing International Documentation,”** includes guidelines for writing manuals and documentation to be translated.
- **Chapter 13, “Product Localization,”** discusses business issues.
- **Chapter 14, “Standards Organizations,”** is a summary of the international standards organizations.
- **Chapter 15, “Internationalization Checklist,”** has a checklist for internationalization.
- **Appendix A, “Languages, Territories, and Locale Names,”** lists the standard names for languages, locales, and so on.
- **Appendix B, “Locale Summaries and Keyboard Layouts,”** lists many locale-specific information and keyboard layouts.
- **Appendix C, “OpenWindows and DevGuide,”** explains how internationalization works with OpenWindows.
- **Appendix D, “XView Programming,”** discusses internationalization with XView.
- **Appendix E, “OLIT Programming,”** discusses internationalization with OPEN LOOK Intrinsic Toolkit (OLIT).
- **Appendix F, “Example Program,”** offers a complete source code for an internationalized Motif application.
- **Appendix G, “Annotated Bibliography,”** is a summary of additional suggested books.
- **Appendix H, “Glossary,”** is a list of key terms.

Contents of the Base Solaris Product

Summary of the Base Product

The base English Solaris 2.6 product includes a number of partial European locales as well as the `en_US.UTF-8` locale.

Solaris 2.6 includes the `en_US.UTF-8` locale, which looks the same as English. For the European locales, it can handle different sets of languages in a single application.

The File System Safe Universal Transformation Format, or UTF-8, is an encoding defined by X/Open as a multi-byte representation of Unicode. The `en_US.UTF-8` locale is the first locale that uses UTF-8 as the codeset to support multiple scripts in the Solaris system. UTF-8 is a variant of UNICODE 2.0. UTF-8 provides input and output support for all Solaris single-byte locales.

The partial locales provide the basic mechanism for entering, displaying, and printing local languages. Messages appear in English.

The partial locales can be split into two groups: the core set and the extended set. The core set is packaged in `SUNWploc` (operating system locale) and `SUNWplow` (window system locale). Since these packages are part of the end user cluster, they are installed automatically. The extended set of locales is packaged in `SUNWploc1` (operating system locale) and `SUNWplow1` (Window system locale). `SUNWpldte` has CDE support for the Eastern European locales.

`SUNWploc1` and `SUNWplow1` are available on the entire cluster only. `SUNWploc1` and `SUNWplow1` need to be added to your system before you can use the locales in the second group.

Core Set of Locales

The core set of locales are installed automatically. The core sets are listed in TABLE 2-1.

TABLE 2-1 Core Set of Locales in `SUNWploc` and `SUNWplow`

Locale	Language	Country	Encoding
de	German	Germany	iso-8859-1
en_AU	English	Australia	iso-8859-1
en_CA	English	Canada	iso-8859-1
en_UK	English	United Kingdom	iso-8859-1
en_US	English	United States	iso-8859-1
en_US.UTF-8	English	United States	UTF-8
es	Spanish	Spain	iso-8859-1
es_AR	Spanish	Argentina	iso-8859-1
es_BO	Spanish	Bolivia	iso-8859-1
es_CL	Spanish	Chile	iso-8859-1
es_CO	Spanish	Columbia	iso-8859-1
es_CR	Spanish	Costa Rica	iso-8859-1
es_EC	Spanish	Ecuador	iso-8859-1
es_GT	Spanish	Guatemala	iso-8859-1
es_MX	Spanish	Mexico	iso-8859-1
es_NI	Spanish	Nicaragua	iso-8859-1
es_PA	Spanish	Panama	iso-8859-1
es_PE	Spanish	Peru	iso-8859-1
es_PY	Spanish	Paraguay	iso-8859-1
es_SV	Spanish	El Salvador	iso-8859-1
es_UY	Spanish	Uruguay	iso-8859-1
es_VE	Spanish	Venezuela	iso-8859-1
fr	French	France	iso-8859-1
it	Italian	Italy	iso-8859-1
sv	Swedish	Sweden	iso-8859-1

Extended Set of Locales

The extended set of locales is not installed automatically. If you want to use locales listed in TABLE 2-2, you need to install these manually.

TABLE 2-2 Extended Set of Locales in SUNWploc1 and SUNWplow1

Locale	Language	Country	Encoding
cz	Czech	Czechoslovakia	iso-8859-2
da	Danish	Denmark	iso-8859-1
de_AT	German	Austria	iso-8859-1
de_CH	German	Switzerland	iso-8859-1
el	Greek	Greece	iso-8859-7
en_IE	English	Ireland	iso-8859-1
en_NZ	English	New Zealand	iso-8859-1
et	Estonian	Estonia	iso-8859-1
fr_BE	French	Belgium	iso-8859-1
fr_CA	French	Canada	iso-8859-1
fr_CH	French	Switzerland	iso-8859-1
hu	Hungarian	Hungary	iso-8859-2
lt	Lithuanian	Lithuania	iso-8859-4
lv	Latvian	Latvia	iso-8859-4
nl	Dutch	Netherlands	iso-8859-1
nl_BE	Dutch	Belgium	iso-8859-1
no	Norwegian	Norway	iso-8859-1
pl	Polish	Poland	iso-8859-2
pt	Portuguese	Portugal	iso-8859-1
pt_BR	Portuguese	Brazil	iso-8859-1
ru	Russian	Russia	iso-8859-5
su	Finnish	Finland	iso-8859-1
tr	Turkish	Turkey	iso-8859-9

New Unicode Locale: `en_US.UTF-8`

The `en_US.UTF-8` locale enables programming that can input and output scripts in multiple single-byte languages. This is the first locale with this capability in the Solaris operating environment. For more detailed information, see Chapter 6, “Internationalization Framework in Solaris 2.6.”

This locale uses UTF-8 (Universal Character Set Transformation Format for 8 bits) encoding, which was developed by the X/Open-Uniform Joint Internationalization Working Group (XoJIG). This standard has been adopted by the Unicode Consortium, the International Standards Organization, and the International Electrotechnical Commission as a part of Unicode 2.0 and ISO/IEC 10646-1. The `en_US.UTF-8` locale supports the CDE environment only, including the Motif and CDE libraries. This locale is part of the developer cluster.

The locale supports computation for every code point value, which is defined in Unicode 2.0 and ISO/IEC 10646-1. In Solaris 2.6, language script support is limited to pan-European locales. Input method support has been enabled for the following language scripts only. Due to limited font resources, Solaris 2.6 software includes only character glyphs from the following codesets:

- ISO 8859-1 (most Western European languages, such as English, French, Spanish, and German)
- ISO 8859-2 (most Central European languages, such as Czech, Polish, and Hungarian)
- ISO 8859-4 (Scandinavian and Baltic languages)
- ISO 8859-5 (Russian)
- ISO 8859-7 (Greek)
- ISO 8859-9 (Turkish)

New User Locales in Base Solaris

The base English Solaris 2.6 includes the following new locale support:

TABLE 2-3 New User Locales Included in Solaris 2.6

Country	Locale-Name	ISO codeset
Austria	<code>de_AT</code> (German Partial Locale)	8859-1
Estonia	<code>et</code>	8859-1
Czech	<code>cz</code>	8859-2

TABLE 2-3 New User Locales Included in Solaris 2.6 (Continued)

Country	Locale-Name	ISO codeset
Hungary	hu	8859-2
Poland	pl	8859-2
Latvia	lv	8859-4
Lithuania	lt	8859-4
Russia	ru	8859-5
Greece	el	8859-7
Turkey	tr	8859-9

These locales are supported through the `SUNWploc1` (for operating system support), `SUNWplow1` (for OpenWindows support), and `SUNWpldte` (for locales support) packages, which are part of the entire cluster. The fonts for these packages have the format `SUNiXxf`.

- `iX` represents the ISO 8859 codeset.
- `xf` indicates whether the font is optional or required.

`SUNWilrf` contains the required font and `SUNWilof` contains the optional font for an ISO 8859-1 codeset locale. These packages are in different clusters; install the entire cluster or selectively add the appropriate packages. After the packages have been installed, users can login through `dtlogin` to either OpenWindows or CDE and use the characters associated with their locale.

Multiple Key Compose Sequences for New Locales

The Solaris 2.6 operating environment supports compose sequences to create the diacritical marks used in writing the scripts covered in the following codesets:

- ISO 8859-2 (Latin2) Czech, Polish, and Hungarian
- ISO 8859-4 (Latin4) Latvian and Lithuanian
- ISO 8859-9 (Latin5) Turkish

These are the new diacritic characters which can be created with the following keys and the Compose key.

- diaeresis = citation (") (for example, Compose + A + " = Ä)
- caron = v (for example, Compose + E + v = E caron)
- breve = u
- ogonek = a

- doubleacute = > greater
- degree symbol = O + 0 (oh plus zero)
- currency symbol = 0 + x (zero plus x)

Keyboard Mapping for Greek and Russian Scripts

The Solaris 2.6 operating environment supports new keyboard mapping for Greek and Russian, which allows Greek or Russian script input with the appropriate Sun keyboard.

- ISO 8859-5 Russian
- ISO 8859-7 Greek

New Keyboard Support in Solaris 2.6

The following locales have keyboard layouts for sparc (X-server) and X86 (Xserver PLUS console):

- Czech
- Hungary
- Poland
- Latvia
- Lithuania
- Russia
- Greece
- Turkey

[X-server is CDE and OW, console is command line]

Changing Between Keyboards on SPARC

Support for changing layouts in Solaris is achieved only by using the dip-switch settings under the keyboard. The keyboard layout determined by the dip switches. A list of keyboard layouts and corresponding defined dip-switch settings is at `/usr/openwin/share/etc/keytables/keytable.map`.

The following table is for a type 4 keyboard (1=switch up 0=switch down).

TABLE 2-4 Layouts for Type 4 Keyboards

Dip Switch in Hex	Keyboard	Setting in Binary
51	Hungary5.kt	110011
52	Poland5.kt	110100
53	Czech5.k	110101
54	Russia5.kt	110110
55	Latvia5.k	110111
56	Turkey5.kt	111000
57	Greece5.kt	111001
58	Lithuania5.kt	111011

Changing the layout from US/UK to Czech is done by changing the dip-switch settings to the setting defined in the file (the file defines them in hex - this needs to be converted into binary as it was done above) and then re-booting.

Russian and Greek keyboard support can be toggled on and off using the Sparc Compose key (Ctrl+Shift+F1 on x86).

Changing Between Keyboards on x86

On x86, a keyboard is selected during the `kdmconfig` part of install. To change this at any time after installation, use `kdmconfig`:

1. Exit CDE/OW to command line
2. Type `kdmconfig -u` (in other words, *kdmconfig unconfigure*)
3. Type `kdmconfig` to run the program
4. Follow instructions to get a new keyboard layout

There are no 'utilities' for either Sparc or x86 (apart from standard Unix tools such as `xmodmap`, `pcmapkeys`) bundled by ELC into Solaris 2.6 for switching keyboards.

New Locales in the Base Installation

The installation window in the base Solaris 2.6 offers several English language locales. To use 8-bit characters, install one of the `en_XX` options. The locale used in the installation becomes the default system locale.

TABLE 2-5 New Locales Offered in Installation

Locale Name	Language/Territory	Codeset
C	American English	7-bit
en_AU	Australian English	8-bit
en_CA	Canadian English	8-bit
en_UK	UK English	8-bit
en_US	American English	8-bit

Using Jumpstart

To enable Jumpstart™ for the new 8-bit locales, add the line `locale xx` (substituting the appropriate 8-bit locale for `xx`, for example, `en_US`) to the Jumpstart profile file. For complete instructions, see Chapter 4 of *Automating Solaris Installation*, available from SunSoft Press. Current Jumpstart users should set the default locale to bypass the language prompt during installation.

How to Use iconv Command

The `iconv` command converts the characters or sequences of characters in file from one codeset to another and writes the results to standard output. If there is no conversion for a particular character, it is converted into an underscore '_' in the target codeset. See the `iconv` man page for more information.

The following options are supported:

- `-f fromcode` Symbol of the input codeset.
- `-t tocode` Symbol of the output codeset.

To convert a mail file from one encoding into another, use the `iconv` command:

```
example% iconv -f from_codeset -t to_codeset mail.codeset > mail.codeset
```

Contents of the Localized Solaris 2.6 Products

The European Localized Solaris 2.6 Product

European Solaris is available in three localized versions: French, German, and European. All three versions of Solaris share the same software media, which includes a fully localized CDE environment, error messages, and online documentation in six languages—French, German, Spanish, Swedish, Italian, and English. The difference is in the printed documentation. The French and German Solaris include localized printed documentation, while the printed documentation for the European version is in English only.

TABLE 3-1 shows a list of locales in the European product. This includes both full and partial locales.

TABLE 3-1 European 2.6 Locales

Locale Name	Language/Territory
C	POSIX English (7 bit)
cz	Czech Republic
da	Denmark
de	Germany
de_AT	Austria
de_CH	Switzerland

TABLE 3-1 European 2.6 Locales (*Continued*)

Locale Name	Language/Territory
el	Greece
en_AU	Australia
en_CA	Canada
en_IE	Ireland
en_NZ	New Zealand
en_UK	United Kingdom
en_US	U.S.A.
es	Spanish
es_AR	Argentina
es_BO	Bolivia
es_CL	Chile
es_CO	Colombia
es_CR	Costa Rica
es_EC	Ecuador
es_GT	Guatemala
es_MX	Mexico
es_NI	Nicaragua
es_PA	Panama
es_PE	Peru
es_PY	Paraguay
es_SV	El Salvador
es_UY	Uruguay
es_VE	Venezuela
et	Estonia
fr	France
fr_BE	Belgium (French)
fr_CA	Canada (French)
fr_CH	Switzerland (French)
hu	Hungary
it	Italy

TABLE 3-1 European 2.6 Locales (*Continued*)

Locale Name	Language/Territory
lt	Lithuania
lv	Latvia
nl	Netherlands
nl_BE	Netherlands/Belgium
no	Norway
pl	Poland
pt	Portugal
pt_BR	Portuguese Brazil
ru	Russian
su	Finland
sv	Sweden

All of these locales are also present in the base Solaris 2.6 release. However, only the European product contains the localized messages.

As mentioned, the locales include partial locales. These are based on core locales for the main language. For example, the `fr_CA` (French Canadian) is based on the `fr` (French) locale. These partial locales utilize the messages that are delivered into its parent locale (French for `fr_CA`). If a locale hasn't been fully localized, then it may contain only English messages.

A number of Eastern European locales have also been added into Solaris 2.6. Previously Sun locales were based on ISO-8859-1. The Eastern European locales are based on other ISO standards, as shown in TABLE 3-2.

Locales that are not listed are still based on ISO-8859-1.

TABLE 3-2 New Eastern European Locales in Solaris 2.6

Locale Name	Language/Territory	ISO
de_AT	German (Austrian)	8859-1
et	Estonian	8859-1
cz	Czech	8859-2
hu	Hungarian	8859-2
pl	Polish	8859-2
lv	Latvian	8859-4

TABLE 3-2 New Eastern European Locales in Solaris 2.6 (Continued)

Locale Name	Language/Territory	ISO
lt	Lithuanian	8859-4
ru	Russian	8859-5
el	Greek	8859-7
tr	Turkish	8859-9

All of the locales support character input and output. There is also `iconv` support for many of the major codesets. (For more on `iconv`, see the man pages.) The `iconv` modules are available on the end-user cluster of the Euro product and on the entire cluster of other products, including the base product. See TABLE 3-3 for details.

TABLE 3-3 `iconv` Support for Major Codesets

Code	Symbol	Target Code	Symbol	Comment
ISO 8859-2	iso2	MS 1250	win2	Windows Latin 2
ISO 8859-2	iso2	MS 852	dos2	MS-DOS Latin 2
ISO 8859-2	iso2	Mazovia	maz	Mazovia
ISO 8859-2	iso2	DHN	dhn	Dom Handlowy Nauki
MS 1250	win2	ISO 8859-2	iso2	ISO Latin 2
MS 1250	win2	MS 852	dos2	MS-DOS Latin 2
MS 1250	win2	Mazovia	maz	Mazovia
MS 1250	win2	DHN	dhn	Dom Handlowy Naduki
MS 852	dos2	ISO 8859-2	iso2	ISO Latin 2
MS 852	dos2	MS 1250	win2	Windows Latin 2
MS 852	dos2	Mazovia	maz	Mazovia
MS 852	dos2	DHN	dhn	Dom Handlowy Nauki
Mazovia	maz	ISO 8859-2	iso2	ISO Latin 2
Mazovia	maz	MS 1250	win2	Windows Latin 2
Mazovia	maz	MS 852	dos2	MS-DOS Latin 2
Mazovia	maz	DHN	dhn	Dom Handlowy Nauki
DHN	dhn	ISO 8859-2	iso2	ISO Latin 2
DHN	dhn	MS 1250	win2	Windows Latin 2
DHN	dhn	MS 852	dos2	MS-DOS latin 2
DHN	dhn	Mazovia	maz	Mazovia

TABLE 3-3 `iconv` Support for Major Codesets (*Continued*)

Code	Symbol	Target Code	Symbol	Comment
ISO 8859-5	iso5	KOI8-R	koi8	KOI8-R
ISO 8859-5	iso5	PC Cyrillic	alt	Alternative PC Cyrillic
ISO 8859-5	iso5	MS 1251	win5	Window Cyrillic
ISO 8859-5	iso5	Mac Cyrillic	mac	Macintosh Cyrillic
KOI8-R	koi8	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
KOI8-R	koi8	PC Cyrillic	alt	Alternative PC Cyrillic
KOI8-R	koi8	MS 1251	win5	Windows Cyrillic
KOI8-R	koi8	Mac Cyrillic	mac	Macintosh Cyrillic
PC Cyrillic	alt	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
PC Cyrillic	alt	KOI8-R	koi8	KOI8-R
PC Cyrillic	alt	MS 1251	win5	Windows Cyrillic
PC Cyrillic	alt	Mac Cyrillic	mac	Macintosh Cyrillic
MS 1251	win5	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
MS 1251	win5	KOI8-R	koi8	KOI8-R
MS 1251	win5	PC Cyrillic	alt	Alternative PC Cyrillic
MS 1251	win5	Mac Cyrillic	mac	Macintosh Cyrillic
Mac Cyrillic	mac	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
Mac Cyrillic	mac	KOI8-R	koi8	KOI8-R
Mac Cyrillic	mac	PC Cyrillic	alt	Alternative PC Cyrillic
Mac Cyrillic	mac	MS 1251	win5	Windows Cyrillic

Font Formats

There are many different font formats. The extension lets you determine the font type.

- **PostScript Type 1 Fonts**

PostScript Type 1 fonts, which are also known as Adobe Type Manager (ATM) fonts, Type 1, and outline fonts, contains information in outline form that allows a PostScript printer or ATM to generate fonts of any size. Most of these fonts also contain hinting information which allows fonts to be rendered more readable at a low resolution or a small type size.

- **Bitmap Fonts**

Bitmap fonts contain a picture of the font at a specific size that has been optimized to look good at that specific size. If the font is scaled larger or smaller, the quality may degrade. On the other hand, bitmap fonts display quickly.

Location of Fonts on the System

Fonts are located at:

```
/usr/openwin/lib/locale/iso_8859_x/X11/fonts/X11/Type1/afm
```

or

```
/usr/openwin/lib/locale/iso_8859_x/X11/fonts/X11/75dpi
```

Adding and Removing Font Packages

To manually add font packages to the system:

1. Always add the required font packages before the optional font packages.
2. When you are removing font packages from the system, remove the optional font packages first.

You must follow this procedure in adding or removing fonts. The class action scripts in the font packages depend on this for proper function. The new optional font packages contain scripts that concatenate information onto the required font packages that are already resident on the system. If the required font packages are not there, problems may occur.

The Asian Localized Solaris 2.6 Products

In contrast to the European locales, which are all packaged on one CD, there are four separate Asian products on separate CDs: Japanese, Korean, Traditional Chinese, and Simplified Chinese.

The following table shows the Asian locales supported by these Asian products.

TABLE 3-4 Summary of Asian Locales

CD Set	Locale Name	Description	Supported Character Set
Korean	ko	Korean	KS C 5601-1992
	ko.UTF-8	Korean (UTF-8 locale)	KS C 5700-1995
Simplified Chinese	zh	Simplified Chinese (Mainland China)	GB 2312-1980
Traditional Chinese	zh_TW	Traditional Chinese (Taiwan)	CNS 11643
	zh_TW.BIG5	Traditional Chinese (BIG5 locale)	BIG5
Japanese	ja	Japanese	JIS x 0201-1976
	ja_JP.PCK		JIS x 0208-1990 JIS x 0212-1990

Korean

In December 1995, the Korean government announced a new standard Korean codeset, KSC-5700, which is based on ISO-10646-1/Unicode 2.0. The new standard codeset replaces KSC 5601, which was based on ISO-2022.

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

The ISO-10646 character set cannot be used directly on IBM-PC-based operating systems. For example, the kernel and many other modules of the Solaris operating environment interpret certain byte values as control instructions, such as a null character (0x00) in any string. The ISO-10646 character set can be encoded with any bit combinations in the first or subsequent bytes. The ISO-10646 characters cannot be freely transmitted through the Solaris system with the above limitations. In order to establish a migration path, the ISO-10646 character set defines the UCS Transformation Format (UTF), which recodes the ISO-10646 characters without using C0 controls (0x00..0x1F), C1 controls (0x80..0x9F), space (0x20), and DEL (0x7F).

The ko.UTF-8 is a new Solaris locale to support KSC-5700, the new Korean standard codeset. It supports all characters in the previous KSC 5601 and all 11,172 Korean characters. Korean UTF-8 supports only the Korean language-related ISO-10646 characters and fonts. Because ISO-10646 covers all characters in the world, it is necessary to supply all of the various input methods and fonts so that you may

input and output any character in any language. Before Universal UTF/UCS becomes available, Korean UTF-8 supports only the ISO-10646 code subset that is related to Korean characters and all other characters in the previous Korean standard codeset as well as Extended ASCII.

TABLE 3-5 lists the Korean codesets.

TABLE 3-5 Codeset Conversions Supported for Korean ko, ko.UTF-8

Code	Symbol	TargetCode	Symbol
UTF-8	ko_KR-UTF-8	Wansung	ko_KR-euc
UTF-8	ko_KR-UTF-8	Johap	ko_KR-johap92
UTF-8	ko_KR-UTF-8	Packed	ko_KR-johap
UTF-8	ko_KR-UTF-8	ISO-2022-KR	ko_KR-iso2022-7
Wansung	ko_KR-euc	UTF-8	ko_KR-UTF-8
Johap	ko_KR-johap92	UTF-8	ko_KR-UTF-8
Packed	ko_KR-johap	UTF-8	ko_KR-UTF-8
ISO-2022-KR	ko_KR-iso2022-7	UTF-8	ko_KR-UTF-8
Wansung	ko_KR-euc	Johap	ko_KR-johap92
Wansung	ko_KR-euc	Packed	ko_KR-johap
Wansung	ko_KR-euc	N-Byte	ko_KR-nbyte
Wansung	ko_KR-euc	ISO-2022-KR	ko_KR-iso2022-7
Johap	ko_KR-johap92	Wansung	ko_KR-euc
Packed	ko_KR-johap	Wansung	ko_KR-euc
N-Byte	ko_KR-nbyte	Wansung	ko_KR-euc
ISO-2022-KR	ko_KR-iso2022-7	Wansung	ko_KR-euc

Chinese: Simplified and Traditional

Chinese is written in two standards: Simplified and Traditional.

The People's Republic of China (P.R.C.) uses Simplified Chinese. In five-year steps, Chinese characters, which are often composed of an elaborate number of marks, are being simplified to make it quicker to write and easier to develop technology. The number of characters is also being reduced.

Simplified Chinese Solaris uses the EUC scheme to support the PRC Chinese national standard character set GB2312-80.

The Republic of China (R.O.C.) in Taiwan continues to use Traditional Chinese. Taiwan has a significant computer industry. They are the world leaders in production of laptops. The Taiwanese computer industry uses two mechanisms to produce Traditional Chinese: CNS-11643 and Big-5 encoding. The CNS-11643 codeset is used by the R.O.C. government. The Big-5 codeset is used by industry and private users, especially PC users.

Traditional Chinese Solaris currently uses the EUC scheme to support the government's CNS-11643 codeset. Since many PC applications handle only Big-5 code, there is a significant market demand to support Big-5 code in Traditional Chinese Solaris. Therefore, Solaris 2.6 supports the Big-5 locale. The Big-5 locale allows users to exchange Big-5 encoded files between PC and Solaris 2.6 without conversion procedures. The official name of the Big-5 locale is `zh_TW.BIG5`.

TABLE 3-6 shows the supported codeset conversions for Simplified Chinese.

TABLE 3-6 Codeset Conversions for Simplified Chinese

Code	Symbol	TargetCode	Symbol
GB2312-80	zh_CN.euc	ISO 2022-7	zh_CN.iso2022-7
ISO 2022-7	zh_CN.iso2022-7	GB2312-80	zh_CN.euc
GB2312-80	zh_CN.euc	ISO 2022-CN	zh_CN.iso2022-CN
ISO-2022-CN	zh_CN.iso2022-CN	GB2312-80	zh_CN.euc
UTF-8	UTF-8	GB2312-80	zh_CN.euc
GB2312-80	zh_CN.euc	UTF-8	UTF-8

TABLE 3-7 shows the supported codeset conversions for Traditional Chinese.

TABLE 3-7 Codeset Conversions for Traditional Chinese

Code	Symbol	TargetCode	Symbol
CNS 11643	zh_TW-euc	Big-5	zh_TW-big5
CNS 11643	zh_TW-euc	ISO 2022-7	zh_TW-iso2022-7
Big-5	zh_TW-big5	CNS 11643	zh_TW-euc
Big-5	zh_TW-big5	ISO 2022-7	zh_TW-iso2022-7
ISO 2022-7	zh_TW-iso2022-7	CNS 11643	zh_TW-euc
ISO 2022-7	zh_TW-iso2022-7	Big-5	zh_TW-big5
CNS 11643	zh_TW-eu	ISO 2022-CN-EXT	zh_TW-iso2022-CN-EXT
ISO 2022-CN-EXT	zh_TW-iso2022-CN-EXT	CNS 11643	zh_TW-euc

TABLE 3-7 Codeset Conversions for Traditional Chinese (*Continued*)

Code	Symbol	TargetCode	Symbol
Big-5	zh_TW-big5	ISO 2022-CN	zh_TW-iso2022-CN
ISO 2022-CN	zh_TW-iso2022-CN	Big	zh_TW-big5
UTF-8	UTF-8	CNS 11643	zh_TW-euc
CNS 11643	zh_TW-euc	UTF-8	UTF-8
UTF-8	UTF-8	Big-5	zh_TW-big5
Big-5	zh_TW-big5	UTF-8	UTF-8
UTF-8	UTF-8	ISO 2022-7	zh_TW-iso2022-7
ISO 2022-7	zh_TW-iso2022-7	UTF-8	UTF-8
ISO 2022-CN-EXT	zh_TW-iso2022-CN-EX	Big-5	zh_TW-big5
Big-5	zh_TW-big5	ISO 2022-CN-EXT	zh_TW-iso2022-CN-EXT

Japanese

Three Japanese input systems are available for Japanese Solaris 2.6. They can be used in the `ja` and `ja_JP.PCK` locale. However, some maintenance utilities do not support the PCK codeset.

TABLE 3-8 Japanese Input Systems

Type	Description
Wnn6	Wnn6 consists of the Kana-Kanji conversion server (<code>jservr</code>), interface module for <code>htt</code> (X Input Method Server) called <code>xjsi.so</code> , utilities, dictionaries, and configuration files. Wnn6 is co-packaged with Japanese Solaris 2.6. Wnn6 for Solaris 2.3/2.4 is distributed by OMRON SOFTWARE. Wnn6 for Solaris 2.5/2.5.1 is included in Update CD of Solaris and distributed by Nihon SMCC.
ATOK8	ATOK8 consists of <code>atok8</code> X Input Method Server, utilities, and dictionaries. SunSoft had been releasing ATOK7 from Japanese Solaris 2.1 until 2.5.1. ATOK8 replaces ATOK7. ATOK is a popular Japanese input facility for the Japanese PC market. It is distributed by JUSTSYSTEM. ATOK8 has been co-packaged since Japanese Solaris 2.5.
cs00	cs00 consists of Kana-Kanji conversion server (<code>cs00</code>), interface module for <code>htt</code> (X Input Method Server) called <code>xci.so</code> , utilities, and dictionaries. cs00 is the only set of these three systems that has been bundled. It has been included since Japanese Solaris 2.1.

TABLE 3-9 Japanese TrueType Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
hg gothic b	R	TrueType	RICOH	JISX0208.1983, JISX0201.1976
hg mincho l	R	TrueType	RICOH	JISX0208.1983, JISX0201.1976
heiseimin	R	TrueType	RICOH	JISX0212.1990

TABLE 3-10 Japanese F3 Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
ryumin light kl	R	F3	MORISAWA	JISX0208.1983, JISX0201.1976
gothic medium bbb	R	F3	MORISAWA	JISX0208.1989, JISX0201.1976

TABLE 3-11 Japanese Bitmap Fonts

Full Family Name	Subfamily	Format	Vendor	Encoding
gothic	R, B	PCF(12,14,16,20,24)		JISX0208.1983, JISX0201.1976
minchou	R	PCF(12,14,16,20,24)		JISX0208.1983, JISX0201.1976
hg gothic b	R	PCF(12,14,16,18,20,24)	RICOH	JISX0208.1983, JISX0201.1976
hg mincho l	R	PCF(12,14,16,18,20,2)	RICOH	JISX0208.1983, JISX0201.1976
ryumin light kl	R	PCF(10,12,14,16,18,20,22)	MORIS AWA	JISX0208.1983, JISX0201.1976
gothic medium bbb	R	PCF(10,12,14,16,18,20,22)	MORIS AWA	JISX0208.1983, JISX0201.1976
heiseimin	R	PCF(12,14,16,18,20,24)	RICOH	JISX0212.1990

Note – The F3 fonts and F3 bitmap fonts (Morisawa fonts) will be supported only up to and including the next Solaris release.

Japanese Locales

Japanese Solaris 2.6 supports two locales. The `ja` locale is based on Japanese EUC. The `ja_JP.PCK` locale is based on PC-Kanji code. See the `euCJP(5)` or `PCK(5)` man page for more details.

Japanese Messages and man Pages

Some messages and manual pages have been translated into Japanese in Japanese Solaris 2.6.

Japanese Character Code Converter for `iconv`

The following table shows supported conversion with `iconv(1)` and `iconv(3)`. See the `iconv_ja(5)` man page for details.

TABLE 3-12 `iconv` Conversion Support

Source Code	Target Code
<code>euCJP</code>	<code>PCK</code>
<code>euCJP</code>	<code>JIS7</code>
<code>euCJP</code>	<code>SJIS</code>
<code>euCJP</code>	<code>UTF-8</code>
<code>euCJP</code>	<code>jis</code>
<code>euCJP</code>	<code>ibmj</code>
<code>SJIS</code>	<code>euCJP</code>
<code>SJIS</code>	<code>ISO-2022-JP</code>
<code>SJIS</code>	<code>UTF-8</code>
<code>SJIS</code>	<code>jis</code>
<code>SJIS</code>	<code>ibmj</code>
<code>PCK</code>	<code>euCJP</code>
<code>PCK</code>	<code>UTF-8</code>
<code>PCK</code>	<code>ISO-2022-JP</code>
<code>PCK</code>	<code>jis</code>
<code>PCK</code>	<code>ibmj</code>
<code>ISO-2022-JP</code>	<code>euCJP</code>

TABLE 3-12 `iconv` Conversion Support (*Continued*)

Source Code	Target Code
ISO-2022-JP	PCK
ISO-2022-JP	SJIS
UTF-8	euJP
UTF-8	SJIS
UTF-8	PCK
JIS7	euJP
jis	euJP
jis	PCK
jis	SJIS
ibmj	euJP
ibmj	PCK
ibmj	SJIS

Japanese-specific Commands

The following commands are for handling Japanese data in Japanese Solaris 2.6. See the `jistoec(1)`, `euctoibmj(1)`, `jtty(1)`, `jtops(1)`, or `kanji(1)` man pages for more details.

TABLE 3-13 Japanese-specific Commands

Command	Comments
<code>jistoec</code>	Converts JIS to Japanese EUC
<code>jistosj</code>	Converts JIS to PC Kanji
<code>euctojis</code>	Converts Japanese EUC to JIS
<code>euctosj</code>	Converts Japanese EUC to PC Kanji
<code>sjtojis</code>	Converts PC Kanji to JIS
<code>sjtoec</code>	Converts PC Kanji to Japanese EUC
<code>euctoibmj</code>	Converts Japanese EUC to IBM-Japanese
<code>ibmjtoec</code>	Converts IBM-Japanese to Japanese EUC

TABLE 3-13 Japanese-specific Commands (*Continued*)

Command	Comments
<code>jtty</code>	Sets Japanese terminal characteristics
<code>jtops</code>	PostScript filter for printing Japanese character codeset
<code>kanji</code>	Shows the list of Kanji codes with numbers of EUC, <code>ja_JP.PCK</code> , JIS and JIS kuten code

We recommend using `iconv` code conversion module instead of `jistoec(1)` or `euctoibmj(1)` converters. These last two converters are provided to allow backward compatibility.

Japanese Character Code Converter for TTY STREAMS

These are TTY STREAMS modules that are used to input and output Japanese characters on terminals. Usually `setterm(1)` organizes these modules/command properly for the user environment. `tty(1)` controls the behavior of those STREAMS modules.

Japanese-specific Printer Support

Japanese Solaris 2.6 supports the following Japanese-specific printers:

- Epson VP-5085 (based on ESC/P)
- NEC PC-PR201 (based on 201PL)
- Canon LASERSHOT (based on LIPS)
- Japanese PostScript Printer

JLE Binary Compatibility Package

Japanese Solaris 2.6 also provides Japanese Solaris1.1.x binary-compatibility packages the same as the base products.

User-Defined Character (UDC) Support

Several of the font tools available in the Solaris package are:

- The User-Defined Character (UDC) font editor handles both outline (Type1) and bitmap (PCF) fonts: `/usr/dt/bin/sdtudctool`
- OpenWindows font editor for bitmap font: `/usr/openwin/bin/fontedit`
- OpenWindows Type3 font editor: `/usr/openwin/bin/type3creator`

Note – `fontedit`, `type3creator`, and `fontmanager` will be supported only up to and including the next Solaris release.

Overview of UTF-8

The Universal Transformation Format

The File System Safe Universal Transformation Format, or UTF-8, is an encoding defined by X/Open-Uniform Joint Internationalization Working Group (XoJIG) of X/Open as a multi-byte representation of Unicode. The `en_US.UTF-8` locale is the first locale that uses UTF-8 as the codeset to support multi-scripts in the Solaris system.

The locale supports computation for every code point value defined at Unicode 2.0/ISO/IEC 10646-1. However, due to the limited set of font resources and the fact that few users intend to use all of the code point values, users of the `en_US.UTF-8` locale will see only character glyphs from the following scripts:

- 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)

Also, since this locale is primarily for developers, it belongs to the developer's cluster of Solaris 2.6. Therefore, when you install Solaris 2.6, you should choose the developer's cluster to install the locale on your system. For more information, see Chapter 5, "Installation."

Note – Motif and the CDE libraries have support for the `en_UTF-8` locale. OpenWindows, XView, and OPENLOOK do not support `en_UTF-8`.

System Environment

Locale Environment Variable

To use the `en_US.UTF-8` locale environment, make sure the locale is installed on your system, then choose the locale as follows.

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

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

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

To check current locale settings in various categories, use the `locale(1)` utility as shown below:

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

You can also start the `en_US.UTF-8` environment from the CDE desktop at the CDE login screen's `Options -> Language` menu and choosing `en_US.UTF-8`.

TTY Environment Setup

To ensure correct text edit operation by a terminal or by a terminal emulator such as `dtterm(1)`, users should push certain locale-specific STREAMS modules onto their Streams.

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

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

TABLE 4-1 STREAMS Modules Supported by `en_US.UTF-8`

STREAMS Module	Description
<code>/usr/kernel/strmod/eucu8</code>	UTF-8 STREAMS module for tail side
<code>/usr/kernel/strmod/u8euc</code>	UTF-8 STREAMS module for head side
<code>/usr/kernel/strmod/u8lat1</code>	Code conversion STREAMS module between UTF-8 and ISO 8859-1
Western European <code>/usr/kernel/strmod/u8lat2</code>	Code conversion STREAMS module between UTF-8 and ISO 8859-2
Eastern European <code>/usr/kernel/strmod/u8koi8</code>	Code conversion STREAMS module between UTF-8 and KOI8-R (Cyrillic)

Loading a STREAMS Module at Kernel

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

```
system% su
Password:
system#
```

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

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

If the STREAMS module, such as `eucu8`, is already installed, the output will look as follows:

```
system# modinfo | grep eucu8
89 ff798000 4b13 18 1 eucu8 (eucu8 module)
system#
```

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

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

The STREAMS module is installed at the kernel, and you can now push it onto a Stream.

To unload a module from the kernel, use `modunload(1M)`, as shown below. In this example, the `eucu8` module is being unloaded.

```
system# modinfo | grep eucu8
89 ff798000 4b13 18 1 eucu8 (eucu8 module)
system# modunload -i 89
```

dtterm and Terminals Capable of Input and Output UTF-8

The `dtterm(1)` and any terminal that supports input and output of UTF-8 codeset should have following STREAMS configuration:

```
head <-> u8euc <-> ttcompat <-> ldterm <-> eucu8 <-> pseudo-TTY
```

In this example, `u8euc` and `eucu8` are the modules supported by the `en_US.UTF-8` locale.

To set up the above STREAMS configuration, use `strchg(1)`, as shown below:

```
system% cat > /tmp/mystreams
u8euc
ttcompat
ldterm
eucu8
ptem
^D
system% strchg -f /tmp/mystreams
```

When using `strchg(1)`, be sure you are either superuser or the owner of the device. To see the current configuration of the STREAMS, use `strconf(1)` as shown below:

```
system% strconf
u8euc
ttcompat
ldterm
eucu8
ptem
pts
system%
```

To revert to the original configuration, set the STREAMS configuration again as shown below:

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

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

For terminals that support only Latin-1 (ISO 8859-1), Latin-2 (ISO 8859-2), or KOI8-R, you should have the following STREAMS configuration:

```
head <-> u8euc <-> ttcompat <-> ldterm <-> eucu8 <-> u8lat1 <-> TTY
```

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

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

```
system% cat > /tmp/mystreams
u8euc
ttcompat
ldterm
eucu8
u8lat1
ptem
^D
system% strchg -f /tmp/mystreams
```

Be sure that you are either superuser or the owner of the device when you use `strchg(1)`. To see the current configuration, use `strconf(1)`, as follows:

```
system% strconf
u8euc
ttcompat
ldterm
eucu8
u8lat1
ptem
pts
system%
```

To revert to the original configuration, set the STREAMS configuration as follows:

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

Setting Terminal Options

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

```
system% /bin/stty cs8 -istrip defucw
```

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

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

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

Saving the Settings in `~/ .cshrc`

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

```
setenv LANG en_US.UTF-8
if ($?USER != 0 && $?prompt != 0) then
    cat >! /tmp/mystreams$$ << _EOF
    u8euc
    ttcompat
    ldterm
    eucu8
    ptem
_EOF
    /bin/strchg -f /tmp/mystreams$$
    /bin/rm -f /tmp/mystreams$$
    /bin/stty cs8 -istrip defeucw
endif
```

With these lines in your `.cshrc` file, you do not have to type all of the commands each time. Note that the second `_EOF` should be in the first column of the file. You can also create a file called `mystreams` and save it so the `.cshrc` references to `mystreams` instead of creating it whenever you start a C shell.

Code Conversions

The `en_US.UTF-8` locale supports various code conversions among major codesets of several countries through `iconv(1)` and `iconv(3)`.

The available `fromcode` and `to code` names that can be applied to `iconv(1)` and `iconv_open(3)` are shown in TABLE 4-2:

TABLE 4-2 Available Code Conversions in `en_US.UTF-8`

From Code	To Code	Description
646	UTF-8	ISO 646 (US-ASCII) to UTF-8
UTF-8	8859-1	UTF-8 to ISO 8859-1
UTF-8	8859-2	UTF-8 to ISO 8859-2
UTF-8	8859-3	UTF-8 to ISO 8859-3
UTF-8	8859-4	UTF-8 to ISO 8859-4
UTF-8	8859-5	UTF-8 to ISO 8859-5 (Cyrillic)
UTF-8	8859-6	UTF-8 to ISO 8859-6 (Arabic)
UTF-8	8859-7	UTF-8 to ISO 8859-7 (Greek)
UTF-8	8859-8	UTF-8 to ISO 8859-8 (Hebrew)
UTF-8	8859-9	UTF-8 to ISO 8859-9
UTF-8	8859-10	UTF-8 to ISO 8859-10
8859-1	UTF-8	ISO 8859-1 to UTF-8
8859-2	UTF-8	ISO 8859-2 to UTF-8
8859-3	UTF-8	ISO 8859-3 to UTF-8
8859-4	UTF-8	ISO 8859-4 to UTF-8
8859-5	UTF-8	ISO 8859-5 (Cyrillic) to UTF-8
8859-6	UTF-8	ISO 8859-6 (Arabic) to UTF-8
8859-7	UTF-8	ISO 8859-7 (Greek) to UTF-8
8859-8	UTF-8	ISO 8859-8 (Hebrew) to UTF-8
8859-9	UTF-8	ISO 8859-9 to UTF-8
8859-10	UTF-8	ISO 8859-10 to UTF-8
UTF-8	KOI8-R	UTF-8 to KOI8-R (Cyrillic)
KOI8-R	UTF-8	KOI8-R (Cyrillic) to UTF-8
UTF-8	UCS-2	UTF-8 to UCS-2
UCS-2	UTF-8	UCS-2 to UTF-8
UTF-8	UCS-4	UTF-8 to UCS-4
UCS-4	UTF-8	UCS-4 to UTF-8
UTF-8	UTF-7	UTF-8 to UTF-7

TABLE 4-2 Available Code Conversions in `en_US.UTF-8` (Continued)

From Code	To Code	Description
UTF-7	UTF-8	UTF-7 to UTF-8
UTF-8	UTF-16	UTF-8 to UTF-16
UTF-16	UTF-8	UTF-16 to UTF-8
UTF-8	euJP	UTF-8 to Japanese EUC
UTF-8	PCK	UTF-8 to Japanese PC Kanji (a.k.a. SJIS)
euJP	UTF-8	Japanese EUC to UTF-8
PCK	UTF-8	Japanese PC Kanji (a.k.a. SJIS) to UTF-8
UTF-8	ko_KR-euc	UTF-8 to Korean EUC
UTF-8	ko_KR-johap	UTF-8 to Korean Johap (KS C 5601-1987
UTF-8	ko_KR-johap92	UTF-8 to Korean Johap (KS C 5601-1992)
UTF-8	ko_KR-iso2022-7	UTF-8 to ISO-2022-KR
ko_KR-euc	UTF-8	Korean EUC to UTF-8
ko_KR-johap	UTF-8	Korean Johap (KS C 5601-1987) to UTF-8
ko_KR-johap92	UTF-8	Korean Johap (KS C 5601-1992) to UTF-8
ko_KR-iso2022-7	UTF-8	ISO-2022-KR to UTF-8
UTF-8	gb2312	UTF-8 to Chinese/PRC EUC (GB 2312-1980
UTF-8	iso2022	UTF-8 to ISO-2022-CN
gb2312	UTF-8	Chinese/PRC EUC (GB 2312-1980) to UTF-8
iso2022	UTF-8	ISO-2022-CN to UTF-8
UTF-8	zh_TW-euc	UTF-8 to Chinese/Taiwan EUC (CNS 11643-1992)
UTF-8	zh_TW-big5	UTF-8 to Chinese/Taiwan Big5
UTF-8	zh_TW-iso2022-7	UTF-8 to ISO-2022-TW
zh_TW-euc	UTF-8	Chinese/Taiwan EUC (CNS 11643-1992) to UTF-8
zh_TW-big5	UTF-8	Chinese/Taiwan Big5 to UTF-8
zh_TW-iso2022-7	UTF-8	ISO-2022-TW to UTF-8

For more details on `iconv` code conversion, see the `iconv(1)`, `iconv_open(3)`, `iconv(3)`, and `iconv_close(3)` man pages. For more information on available code conversions, see `iconv_en_US.UTF-8(5)`.

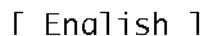
Script Selection and Input Modes

The `en_US.UTF-8` locale supports multiple scripts. This section contains details about each of the input modes: English, Cyrillic, and Greek.

English Input Mode

The English input mode encompasses not only the English alphabet but also characters with diacritical marks (for example, á, è, î, ò, and ü) and special characters (such as ¡, £, ¢, §, ¸).

The English input mode is the default mode for any application. The input mode is displayed at the bottom left corner of the GUI application, as shown in FIGURE 4-1:



English

FIGURE 4-1 English Input Mode

To insert characters with diacritical marks or special characters from Latin-1, Latin-2, Latin-4, and Latin-5, you must type a compose sequence, as shown in the following examples:

- For Ä, press and release Compose, then A, and then "
- For ¸, press and release Compose, then +, and then -

The following tables are the most commonly used compose sequences in Latin-1, Latin-2, Latin-4, and Latin-5 script input.

TABLE 4-3 Common Latin-1 Compose Sequences

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	[spacebar]	[spacebar]	Non-breaking space
Compose	s	1	Superscripted 1
Compose	s	2	Superscripted 2
Compose	s	3	Superscripted 3
Compose	!	!	Inverted exclamation mark
Compose	x	o	Currency symbol ‘¤’
Compose	p	!	Paragraph symbol ‘¶’

TABLE 4-3 Common Latin-1 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	/	u	mu 'u'
Compose	'		apostrophe '''
Compose	'		acute accent '''
Compose	,	,	cedilla ','
Compose	"	"	dieresis ''"
Compose	-	^	macron '-^'
Compose	o	o	degree '°'
Compose	x	x	multiplication sign 'x'
Compose	+	-	plus-minus '±'
Compose	-	-	soft hyphen '-'
Compose	-	:	division sign '÷'
Compose	-	a	ordinal (feminine) a 'ª'
Compose	a	-	ordinal (feminine) a 'ª'
Compose	-	o	ordinal (masculine) o 'º'
Compose	o	-	ordinal (masculine) o 'º'
Compose	-	,	not sign '¬'
Compose	.	.	middle dot '·'
Compose	1	2	vulgar fraction 1/2
Compose	1	4	vulgar fraction 1/4
Compose	3	4	vulgar fraction 3/4
Compose	<	<	left double angle quotation mark '«'
Compose	>	>	right double angle quotation mark '»'
Compose	?	?	inverted question mark '¿'
Compose	A	`	A grave 'À'
Compose	A	'	A acute 'Á'
Compose	A	*	A ring above 'Â'
Compose	A	"	A dieresis 'Ä'
Compose	A	^	A circumflex 'Â'
Compose	A	~	A tilde 'Ã'

TABLE 4-3 Common Latin-1 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	A	E	AE diphthong 'Æ'
Compose	C	,	C cedilla 'Ç'
Compose	C	o	copyright sign '©'
Compose	D	-	Capital eth 'Ð'
Compose	E	`	E grave 'È'
Compose	E	'	E acute 'É'
Compose	E	"	E dieresis 'Ë'
Compose	E	^	E circumflex 'Ê'
Compose	I	`	I grave 'Ì'
Compose	I	'	I acute 'Í'
Compose	I	"	I dieresis 'Ï'
Compose	I	^	I circumflex 'Î'
Compose	L	-	pound sign '£'
Compose	N	~	N tilde 'Ñ'
Compose	O	`	O grave 'Ò'
Compose	O	'	O acute 'Ó'
Compose	O	/	O slash 'Ø'
Compose	O	"	O dieresis 'Ö'
Compose	O	^	O circumflex 'Ô'
Compose	O	~	O tilde 'Õ'
Compose	R	O	registered mark '®'
Compose	T	H	Thorn 'Þ'
Compose	U	`	U grave 'Ù'
Compose	U	'	U acute 'Ú'
Compose	U	"	U dieresis 'Û'
Compose	U	^	U circumflex 'Û'
Compose	Y	'	Y acute 'Ý'
Compose	Y	-	yen sign '¥'
Compose	a	`	a grave 'à'

TABLE 4-3 Common Latin-1 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	a	'	a acute 'á'
Compose	a	*	a ring above 'å'
Compose	a	"	a dieresis 'ä'
Compose	a	^	a circumflex 'â'
Compose	a	~	a tilde 'ã'
Compose	a	^	a circumflex 'â'
Compose	a	e	ae diphthong 'æ'
Compose	c	,	c cedilla 'ç'
Compose	c	/	cent sign '¢'
Compose	c	o	copyright sign '©'
Compose	d	-	eth 'ð'
Compose	e	`	e grave 'è'
Compose	e	'	e acute 'é'
Compose	e	"	e dieresis 'ë'
Compose	e	^	e circumflex 'ê'
Compose	i	`	i grave 'ì'
Compose	i	'	i acute 'í'
Compose	i	"	i dieresis 'ï'
Compose	i	^	i circumflex 'î'
Compose	n	~	n tilde 'ñ'
Compose	o	`	o grave 'ò'
Compose	o	'	o acute 'ó'
Compose	o	/	o slash 'ø'
Compose	o	"	o dieresis 'ö'
Compose	o	^	o circumflex 'ô'
Compose	o	~	o tilde 'õ'
Compose	s	s	German double s 'ß'
Compose	t	h	thorn 'þ'
Compose	u	`	u grave 'ù'

TABLE 4-3 Common Latin-1 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	u	'	u acute 'ú'
Compose	u	"	u dieresis 'ü'
Compose	u	^	u circumflex 'û'
Compose	y	'	y acute 'ÿ'
Compose	y	"	y dieresis 'ÿ'
Compose			broken bar ' '

TABLE 4-4 contains the Latin-2 compose sequences.

Note – Composes sequences defined in TABLE 4-3 are not included in TABLE 4-4.

TABLE 4-4 Common Latin-2 Compose Sequences

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	a	'	ogonek á
Compose	u	''	breve ü
Compose	v	''	caron
Compose	"	''	double acute "
Compose	A	a	A ogonek a
Compose	A	u	A breve
Compose	C	'	C acute
Compose	C	v	C caron
Compose	D	v	D caron
Compose	-	D	D stroke
Compose	E	v	E caron
Compose	E	a	E ogonek
Compose	L	'	L acute
Compose	L	-	L stroke
Compose	L	>	L caron
Compose	N	'	N acute

TABLE 4-4 Common Latin-2 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	N	v	N caron
Compose	O	>	O double acute
Compose	S	'	S acute
Compose	S	v	S caron
Compose	S	,	S cedilla
Compose	R	'	R acute
Compose	R	v	R caron
Compose	T	v	T caron
Compose	T	,	T cedilla
Compose	U	*	U ring above
Compose	U	>	U double acute
Compose	Z	'	Z acute
Compose	Z	v	Z caron
Compose	Z	.	Z dot above
Compose	a	a	a ogonek
Compose	a	u	a breve
Compose	c	'	c acute
Compose	c	v	c caron
Compose	d	v	d caron
Compose	-	d	d stroke
Compose	e	v	e caron
Compose	e	a	e ogonek
Compose	l	'	l acute
Compose	l	-	l stroke
Compose	l	>	l caron
Compose	n	'	n acute
Compose	n	v	n caron
Compose	o	>	o double acute
Compose	s	'	s acute

TABLE 4-4 Common Latin-2 Compose Sequences (*Continued*)

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	s	v	s caron
Compose	s	,	s cedilla
Compose	r	'	r acute
Compose	r	v	r caron
Compose	t	v	t caron
Compose	t	,	t cedilla
Compose	u	*	u ring above
Compose	u	>	u double acute
Compose	z	'	z acute
Compose	z	v	z caron
Compose	z	.	z dot above

TABLE 4-5 contains the Latin-4 compose sequences.

Note – Compose sequences defined in TABLE 4-3 or TABLE 4-4 are not included in this table.

TABLE 4-5 Common Latin-4 Compose Sequences

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	k	k	kra
Compose	A	_	A macron
Compose	E	_	E macron
Compose	E	.	E dot above
Compose	G	,	G cedilla
Compose	I	_	I macron
Compose	I	~	I tilde
Compose	I	a	I ogonek
Compose	K	,	K cedilla
Compose	L	,	L cedilla

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

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	N	,	N cedilla
Compose	O	–	O macron
Compose	R	,	R cedilla
Compose	T		T stroke
Compose	U	~	U tilde
Compose	U	a	U ogonek
Compose	U	–	U macron
Compose	N	N	Eng
Compose	a	–	a macron
Compose	e	–	e macron
Compose	e	.	e dot above
Compose	g	,	g cedilla
Compose	i	–	i macron
Compose	i	~	i tilde
Compose	i	a	i ogonek
Compose	k	,	k cedilla
Compose	l	,	l cedilla
Compose	n	,	n cedilla
Compose	o	–	o macron
Compose	r	,	r cedilla
Compose	t		t stroke
Compose	u	~	u tilde
Compose	u	a	u ogonek
Compose	u	–	u macron
Compose	n	n	eng

Note – Compose sequences defined in TABLE 4-3, TABLE 4-4, or TABLE 4-6 are not

included in this table.

TABLE 4-6 Common Latin-5 Compose Sequences

Press and Release	Then Press and Release	Then Press and Release	Result
Compose	G	u	G breve
Compose	I	.	I dot above
Compose	g	u	g breve
Compose	i	.	i dotless

Cyrillic Input Mode

To switch to Cyrillic input mode from English input mode, press Compose c c. If you are currently in Greek input mode, first return to English input mode, then switch to Cyrillic mode.

The input mode is displayed at the bottom left corner of your GUI application, as shown FIGURE 4-2:

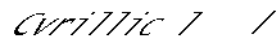


FIGURE 4-2 Cyrillic Input Mode

After you switch to Cyrillic input mode, you cannot enter English text. To switch back to English input mode, type Control-Space. The Russian keyboard layout appears in FIGURE 4-3:

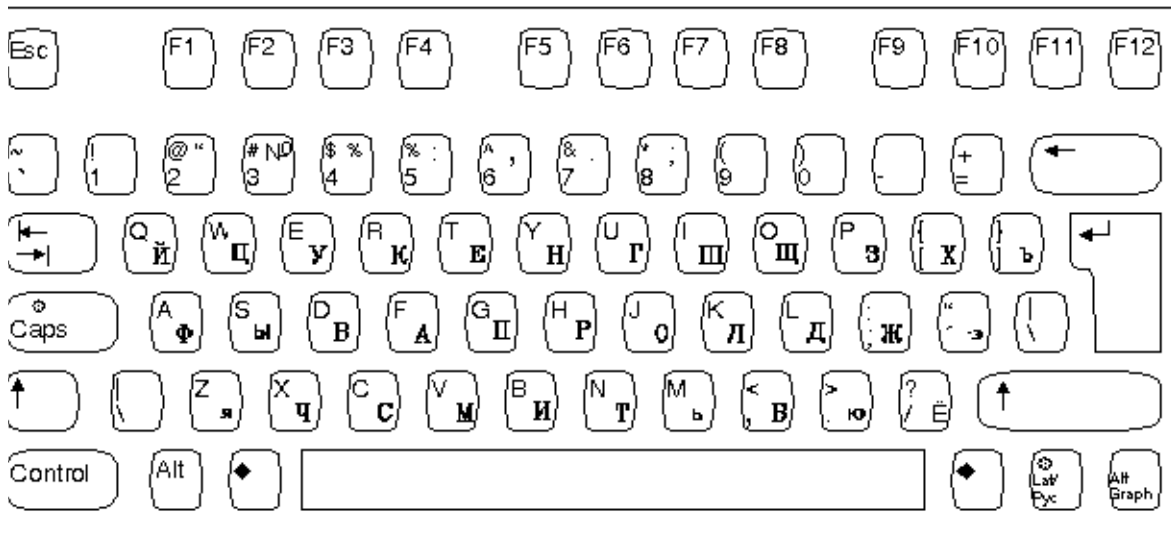


FIGURE 4-3 Russian Keyboard Layout

Greek Input Mode

To switch to Greek input mode from English input mode, press Compose g g. If you are currently in Cyrillic input mode, first return to English input mode and then switch to Greek mode.

The input mode is displayed at the left bottom corner of your GUI application is shown in FIGURE 4-4:

[Greek]

FIGURE 4-4 Greek Input Mode

After you switch to Greek input mode, you cannot enter English text. To switch back to English input mode, type Control-Space. The Greek keyboard layouts appear in FIGURE 4-5 and FIGURE 4-6:

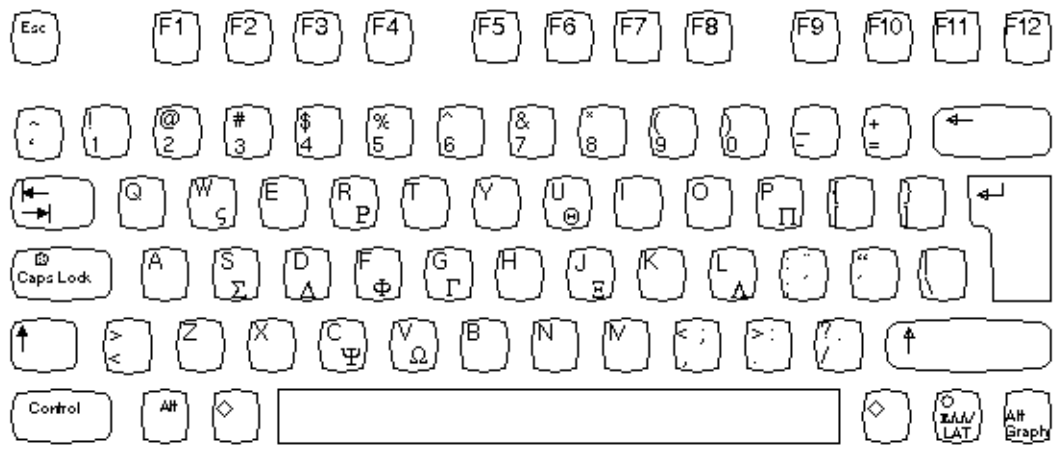


FIGURE 4-5 Greek Keyboard Layout (European Keyboard)

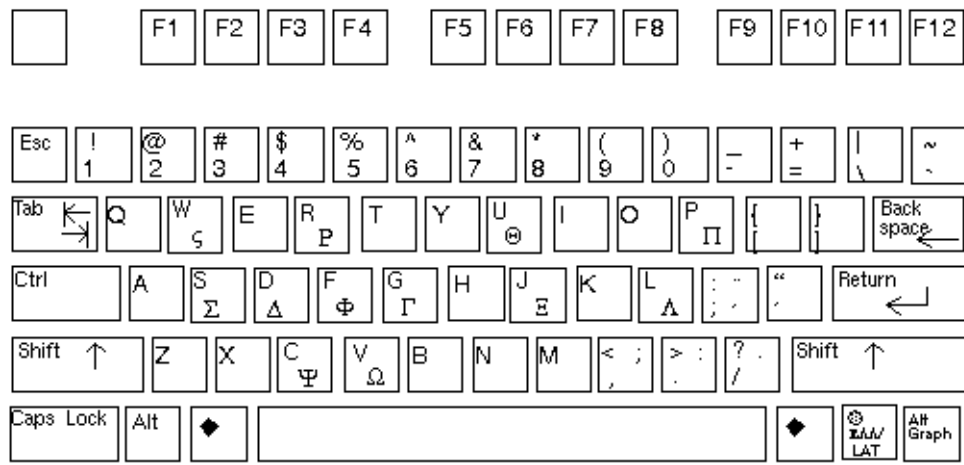


FIGURE 4-6 Greek Keyboard Layout (UNIX Keyboard)

Printing

The `en_US.UTF-8` locale provides a printing utility, `xutops(1)`. This utility can print flat text files written in UTF-8 using X11 bitmap fonts available on the system. Because the output from the utility is standard PostScript, the output can be sent to any PostScript printer.

To use the utility, type the following:

```
system% xutops filename | lp
```

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

```
system% lpr filename | xutops | lp
```

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

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

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

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

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

```
Input types: simple
Output types: XUTOPS
Command: /usr/openwin/bin/xutops
```

The filter converts default type file input to PostScript output using `/usr/openwin/bin/xutops`.

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

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

For more details on `xutops(1)`, refer to `xutops(1)` and `xutops(5)` man pages.

Programming Environment

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

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

FontSet Used with UTF-8

The `en_US.UTF-8` locale in Solaris 2.6 supports fonts for the following charsets:

- ISO 8859-1
- ISO 8859-2
- ISO 8859-4
- ISO 8859-5
- ISO 8859-7
- ISO 8859-9

Because Solaris 2.6 supports the CDE desktop environment, each charset has guaranteed sets of fonts.

The following list shows the Latin-1 fonts that are supported in Solaris 2.6:

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

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

A fontset for an application should have a collection of fonts that contains each of the above charsets, as in the following example:

```
fs = XCreateFontSet(display,
    "-dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-1,
    -dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-2,
    -dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-4,
    -dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-5,
    -dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-7,
    -dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*iso8859-9",
    &missing_ptr, &missing_count, &def_string);
```


Installation

Solaris 2.6 allows you to install more than one locale on a machine. This allows the developer to test different locales or the user to work in different locales for different projects. This chapter describes how to add additional locales on the machine.

Adding Packages

This section describes how to install packages with the `pkgadd` command.

▼ How to Add Packages to a Standalone System

1. **Log in as superuser.**
2. **Remove any packages with the same name as the ones you are adding**

This ensures that the system keeps a proper record of software that has been added and removed. There may be times when you want to maintain multiple versions of the same application on the system. For strategies on how to do this, see “Guidelines for Removing Packages,” and for task information, see “How to Remove a Package.” Both of these can be found in the *System Administration Guide*.

3. Add one or more software packages to the system.

```
# pkgadd -a admin-file -d device-name pkgid . . .
```

In this command,

-a <i>admin-file</i>	(Optional) Specifies an administration file that <code>pkgadd</code> should consult during the installation. (For details about using an administration file, see the <i>System Administration Guide</i>).
-d <i>device-name</i>	Specifies the absolute path to the software packages. <i>Device-name</i> can be a path to a device, a directory, or a spool directory. If you do not specify the path where the package resides, the <code>pkgadd</code> command checks the default spool directory (<code>/var/spool/pkg</code>). If the package is not there, the package installation fails.
<i>pkgid</i>	(Optional) Is the name of one or more packages (separated by spaces) to be installed. If omitted, the <code>pkgadd</code> command installs all available packages.

If `pkgadd` encounters a problem during installation of the package, it displays a message related to the problem, followed by this prompt:

```
Do you want to continue with this installation?
```

Respond with `yes`, `no`, or `quit`. If more than one package has been specified, type `no` to stop the installation of the package being installed. `pkgadd` continues to install the other packages. Type `quit` to stop the installation.

4. Verify that the package has been installed successfully, using the `pkgchk` command.

```
# pkgchk -v pkgid
```

If `pkgchk` determines there are no errors, it returns a list of installed files. Otherwise, it reports the error.

Installing Software From a Mounted CD

The following example shows a command to install the `SUNWaudio` package from a mounted Solaris 2.x CD. The example also shows use of the `pkgchk` command to verify that the packages files were installed properly.

```
# pkgadd -d /cdrom/cdrom0/s0/Solaris_2.6/Product SUNWaudio
.
.
.
Installation of SUNWaudio> complete.
# pkgchk -v SUNWaudio
/usr
/usr/bin
/usr/bin/audioconvert
/usr/bin/audioplay
/usr/bin/audiorecord
```

Installing Software From a Remote Package Server

If the packages you want to install are available from a remote system, you can mount the directory containing the packages (in package format) manually and install packages on the local system. The following example shows the commands to do this. In this example, assume the remote system named `package-server` has software packages in the `/latest-packages` directory. The mount command mounts the packages locally on `/mnt`, and the `pkgadd` command installs the `SUNWaudio` package.

```
# mount -F nfs -o ro package-server:/latest-packages /mnt
# pkgadd -d /mnt SUNWaudio
.
.
.
Installation of SUNWaudio> was successful.
```

If the automounter is running at your site, you do not need to mount the remote package server manually. Instead, use the automounter path (in this case, `/net/package-server/latest-packages`) as the argument to the `-d` option.

```
# pkgadd -d /net/package-server/latest-packages SUNWaudio
.
.
.
Installation of SUNWaudio> was successful.
```

The following example is similar to the previous one, except it uses the `-a` option and specifies an administration file named `noask-pkgadd`. In this example, assume the `noask-pkgadd` administration file is in the default location, `/var/sadm/install/admin`.

```
# pkgadd -a noask-pkgadd -d /net/package-server/latest-packages SUNWaudio
.
.
.
Installation of SUNWaudio> was successful.
```

Installing the Localization Product

The following describes the list of common packages for the operating system localization and the window system localization.

European Package

TABLE 5-1 Pan-European Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
All Euro	SUNWploc SUNWploc1 SUNWenise SUNWeuise	SUNWplow SUNWplow1 SUNWpldte		

French Files

TABLE 5-2 French Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
fr			SUNWfros	SUNWfoaud SUNWfobk SUNWfodcv SUNWfodem SUNWfodst SUNWfodte SUNWfoimt SUNWforte SUNWfrbas SUNWfrdst SUNWfrdte SUNWfrhe SUNWfrhed SUNWfrim SUNWfris SUNWfrwm SUNWftltk SUNWfwacx SUNWfxplt

German Files

TABLE 5-3 German Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
de			SUNWdeos	SUNWdoaud SUNWdobk SUNWdodcv SUNWdodem SUNWdodst SUNWdodte SUNWdoimt SUNWdorte SUNWdebas SUNWdedst SUNWdedte SUNWdehe SUNWdehed SUNWdeim SUNWdeis SUNWdewm SUNWdtltk SUNWdwacx SUNWdxplt

Italian Files

TABLE 5-4 Italian Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
it			SUNWitos	SUNWioaud SUNWiobk SUNWiodcv SUNWiodem SUNWiodst SUNWiodte SUNWioimt SUNWiorte SUNWitbas SUNWitdst SUNWitdte SUNWithe SUNWithed SUNWitim SUNWitis SUNWitwm SUNWitltk SUNWiwacx SUNWixplt

Spanish Files

TABLE 5-5 Spanish Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
es			SUNWesos	SUNWeoaud SUNWeobk SUNWeodcv SUNWeodem SUNWeodst SUNWeodte SUNWeoimt SUNWeorte SUNWesbas SUNWesdst SUNWesdte SUNWeshe SUNWeshed SUNWesim SUNWesis SUNWeswm SUNWetltk SUNWewacx SUNWexplt

Swedish Files

TABLE 5-6 Swedish Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
sv			SUNWsvos	SUNWsoaud SUNWsobk SUNWsodcv SUNWsodem SUNWsodst SUNWsodte SUNWsoimt SUNWsorte SUNWsvbas SUNWsvdst SUNWsvdte SUNWsvhe SUNWsvhed SUNWsvim SUNWsvis SUNWsvwm SUNWstltk SUNWswacx SUNWsxplt

Eastern European Files

TABLE 5-7 European Files for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
Font packages for the Eastern European locales		SUNWi2of SUNWi2rf SUNWi4of SUNWi4rf SUNWi5of SUNWi5rf SUNWi7of SUNWi7rf SUNWi9of SUNWi9rf		

Detailed Descriptions of European Files

TABLE 5-8 European Package Descriptions

Package Name	Package Description
SUNWi1of	ISO-8859-1 (Latin-1) Optional Fonts
SUNWi1of	ISO-8859-1 (Latin-1) Optional Fonts
SUNWi2of	X11 fonts for ISO-8859-2 character set (optional fonts)
SUNWi2rf	X11 fonts for ISO-8859-2 character set (required fonts)
SUNWi4of	X11 fonts for ISO-8859-4 character set (optional fonts)
SUNWi4rf	X11 fonts for ISO-8859-4 character set (required fonts)
SUNWi5of	X11 fonts for ISO-8859-5 character set (optional fonts)
SUNWi5rf	X11 fonts for ISO-8859-5 character set (required fonts)
SUNWi7of	X11 fonts for ISO-8859-7 character set (optional fonts)
SUNWi7rf	X11 fonts for ISO-8859-7 character set (required fonts)
SUNWi9of	X11 fonts for ISO-8859-9 character set (optional fonts)
SUNWi9rf	X11 fonts for ISO-8859-9 character set (required fonts)
SUNWi0aud	Italian OPEN LOOK (R) Audio applications
SUNWi0bk	Italian OpenWindows online handbooks
SUNWi0dcv	Italian OPEN LOOK (R) document and help viewer applications
SUNWi0dem	Italian OPEN LOOK (R) demo programs
SUNWi0dst	Italian OPEN LOOK (R) deskset tools
SUNWi0dte	Italian OPEN LOOK (R) desktop environment
SUNWi0imt	Italian OPEN LOOK (R) imagetool
SUNWi0rte	Italian OPEN LOOK (R) toolkits runtime environment
SUNWislcc	XSH4 conversion for Eastern European locales
SUNWisolc	XSH4 conversion for ISO Latin character sets
SUNWitbas	Base L10N it CDE functionality to run a CDE application
SUNWitdst	Italian CDE Desktop Applications messages
SUNWitdte	Italian CDE Desktop Environment
SUNWithe	Italian CDE Help Runtime Environment
SUNWithed	Italian CDE Help Developer Environment

TABLE 5-8 European Package Descriptions (*Continued*)

Package Name	Package Description
SUNWithev	Italian CDE Online Help
SUNWitim	Italian CDE Imageviewer
SUNWitis	Italian install software localization
SUNWitltk	Italian ToolTalk binaries and shared libraries
SUNWitos	Italian OS localization
SUNWitpmw	Italian (EUC) Localizations for Power Management OW Utilities
SUNWitreg	Italian Solaris User Registration prompts at desktop login for user registration
SUNWitwm	Italian CDE Desktop Window Manages Messages
SUNWiwacx	Italian OPEN LOOK (R) AccessX
SUNWiwbcpx	Italian OpenWindows Binary Compatibility Package
SUNWixplt	Italian X Windows platform software
SUNWeoaud	Spanish OPEN LOOK (R) Audio applications
SUNWeobk	Spanish OpenWindows online handbooks
SUNWeodcv	Spanish OPEN LOOK (R) document and help viewer applications
SUNWeodem	Spanish OPEN LOOK (R) demo programs
SUNWeodst	Spanish OPEN LOOK (R) deskset tools
SUNWeodte	Spanish OPEN LOOK (R) desktop environment
SUNWeoimt	Spanish OPEN LOOK (R) imagetool
SUNWeorte	Spanish OPEN LOOK (R) toolkits runtime environment
SUNWesbas	Base L10N fr CDE functionality to run a CDE application
SUNWesdst	Spanish CDE Desktop Applications
SUNWesdte	Spanish CDE Desktop Environment
SUNWeshe	Spanish CDE Help Runtime Environment
SUNWeshed	Spanish CDE Help Developer Environment
SUNWeshev	Spanish CDE Online Help
SUNWesim	Spanish CDE Desktop apps
SUNWesis	Spanish install software localization
SUNWesos	Spanish OS localization
SUNWespmw	Spanish (EUC) Localizations for Power Management OW Utilities

TABLE 5-8 European Package Descriptions (*Continued*)

Package Name	Package Description
SUNWesreg	Solaris User Registration prompts at desktop login for user registration
SUNWeswm	Spanish CDE Desktop Window Manages Messages
SUNWetltk	Spanish ToolTalk binaries and shared libraries
SUNWenise	English partial locales enabling during install
SUNWeuise	European partial locales enabling during install
SUNWewacx	Spanish OPEN LOOK (R) AccessX
SUNWexplt	Spanish X Windows platform software
SUNWfbcp	French OS Binary Compatibility Package
SUNWfoaud	French French OPEN LOOK (R) Audio applications
SUNWfobk	French OpenWindows online handbooks
SUNWfodcv	French OPEN LOOK (R) document and help viewer applications
SUNWfodem	French OPEN LOOK (R) demo programs
SUNWfodst	French OPEN LOOK (R) deskset tools
SUNWfodte	French OPEN LOOK (R) desktop environment
SUNWfoimt	French OPEN LOOK (R) imagetool
SUNWforte	French OPEN LOOK (R) toolkits runtime environment
SUNWfrbas	Base L10N fr CDE functionality to run a CDE application
SUNWfrdst	French CDE Desktop Applications
SUNWfrdte	french CDE Desktop Environment
SUNWfrhe	French CDE Help Runtime Environment
SUNWfrhed	French CDE Help Developer Environment
SUNWfrhev	French CDE Online Help
SUNWfrim	French CDE ImageViewer
SUNWfris	French install software localization
SUNWfros	French OS localization
SUNWfrpmw	French (EUC) Localizations for Power Management OW Utilities
SUNWfrwm	French CDE Desktop Window Manages Messages
SUNWftltk	French ToolTalk binaries and shared libraries
SUNWfwacx	French OPEN LOOK (R) AccessX

TABLE 5-8 European Package Descriptions (*Continued*)

Package Name	Package Description
SUNWfwbcp	French OpenWindows Binary Compatibility Package
SUNWfxplt	French X Windows platform software
SUNWsoaud	Swedish OPEN LOOK (R) Audio applications
SUNWsobk	Swedish OpenWindows online handbooks
SUNWsodcv	Swedish OPEN LOOK (R) document and help viewer applications
SUNWsodem	Swedish OPEN LOOK (R) demo programs
SUNWsodst	Swedish OPEN LOOK (R) deskset tools
SUNWsodte	Swedish OPEN LOOK (R) desktop environment
SUNWsoimt	Swedish OPEN LOOK (R) imagetool
SUNWsorte	Swedish OPEN LOOK (R) toolkits runtime environment
SUNWstltk	Swedish ToolTalk binaries and shared libraries
SUNWsvbas	Base Swedish CDE functionality messages
SUNWsvdst	Swedish CDE Desktop Applications messages
SUNWsvdte	Swedish CDE Desktop Environment messages
SUNWsvhe	Swedish CDE Help Runtime Environment
SUNWsvhed	Swedish CDE Help Developer Environment messages
SUNWsvhev	Swedish CDE Online Help
SUNWsvim	Swedish CDE Image editor messages
SUNWsvis	Swedish install software localization
SUNWsvos	Swedish OS localization
SUNWsvpmw	Swedish (EUC) Localizations for Power Management OW Utilities
SUNWsvreg	Swedish Solaris User Registration prompts at desktop login for user registration
SUNWsvwm	Swedish CDE Desktop Window Manages Messages
SUNWswacx	Swedish OPEN LOOK (R) AccessX
SUNWsxplt	Swedish X Windows platform software
SUNWdbcp	German OS Binary Compatibility Package
SUNWdebas	Base L10N German CDE functionality to run a CDE application
SUNWdedst	German CDE Desktop Applications
SUNWdedte	German CDE Desktop Login Environment

TABLE 5-8 European Package Descriptions (*Continued*)

Package Name	Package Description
SUNWdehe	German CDE Help Runtime Environment
SUNWdehed	German CDE Help Developer Environment
SUNWdehev	German CDE Online Help
SUNWdeim	German CDE Imageviewer
SUNWdeis	German install software localization
SUNWdeos	German message files for the OS-Networking consolidation
SUNWdepmw	German (EUC) Localizations for Power Management OW Utilities
SUNWdereg	German Solaris User Registration prompts at desktop login for user registration
SUNWdewm	German CDE Desktop Window Manages Messages
SUNWdoaud	German OPEN LOOK (R) Audio applications
SUNWdobk	German OpenWindows online handbooks
SUNWdodcv	German OPEN LOOK (R) document and help viewer applications
SUNWdodem	German OPEN LOOK (R) demo programs
SUNWdodst	German OPEN LOOK (R) deskset tools
SUNWdodte	German OPEN LOOK (R) desktop environment
SUNWdoimt	German OPEN LOOK (R) imagetool
SUNWdorte	German OPEN LOOK (R) toolkits runtime environment
SUNWdwacx	German OPEN LOOK (R) AccessX
SUNWdwbcx	German OpenWindows Binary Compatibility Package
SUNWpldte	CDE Eastern European locale support
SUNWploc	European Partial Locales
SUNWploc1	Supplementary Partial Locales
SUNWplocw	OpenWindows enabling for Partial Locales
SUNWplocw1	OpenWindows enabling for Supplementary Partial Locales

TABLE 5-8 European Package Descriptions (*Continued*)

Package Name	Package Description
SUNWfrrreg SUNWitreg SUNWsvreg SUNWesreg SUNWdereg	Localised e-reg software messages in the End-User cluster and above
SUNWfrpmw SUNWitpmw SUNWsvpmw SUNWespmw SUNWdepmw	Localised Power Management software in the End-User cluster and above
SUNWfwbcp SUNWiwbcp SUNWswbcp SUNWewbcp SUNWdwbcp	Localised Binary Compatibility Packages

European Codesets

In Solaris 2.6 several fonts will display characters which are encoded in the following codesets:

- Latin-1
- Latin-2
- Latin-4
- Cyrillic
- Greek
- Latin-5

European Font Packages

There are a number of font packages in Solaris 2.6.

TABLE 5-9 Font Packages in Solaris 2.6

Font Package	Description
SUNWi2of	Latin-2 Optional fonts
SUNWi2rf	Latin-2 Required fonts
SUNWi4of	Latin-4 Optional fonts

TABLE 5-9 Font Packages in Solaris 2.6

Font Package	Description
SUNWi4rf	Latin-4 Required fonts
SUNWi5of	Cyrillic Optional fonts
SUNWi5rf	Cyrillic Required fonts
SUNWi7of	Greek Optional fonts
SUNWi7rf	Greek Required fonts
SUNWi9of	Latin-5 Optional fonts
SUNWi9rf	Latin-5 Required fonts

- All required font packages are in the developer cluster.
- All fonts (both required and optional) are in the entire cluster.

Asian Packages

The remainder of this chapter covers the Asian packages.

TABLE 5-10 Asian Package for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
All	SUNWale	SUNWxi18n		
(ja/ko/zh/ zh_TW)	SUNWaled	SUNWxim		

TABLE 5-11 Korean Package for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
All (ja/ko/zh/ zh_TW)	SUNWale SUNWaled	SUNWxi18n SUNWxim		
ko (Korean)			SUNWkler SUNWkleu SUNWkbc ¹	SUNWkoaud SUNWkodcv SUNWkodem SUNWkodst SUNWkodte SUNWkoimt SUNWkoman SUNWkorte SUNWklttk SUNWkxman SUNWkxoft SUNWkxplt SUNWkxfnt SUNWkwbc ¹ SUNWkepmw SUNWkervl SUNWkexir SUNWkkcsr
ko.UTF-8 (Korean)			SUNWkiu8 SUNWkuleu	SUNWkcoft SUNWkuodf SUNWkupmw SUNWkuxft SUNWkuxpl

1. Denotes that the package is not delivered for x86.

TABLE 5-12 Chinese Package for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
zh (PRC)			SUNWcleu SUNWcbcp ¹	SUNWcoaud SUNWcodcv SUNWcodem SUNWcodst SUNWcodte SUNWcoimt SUNWcoman SUNWcorte SUNWctltk SUNWcxman SUNWcxoft SUNWcxplt SUNWcxfmt SUNWcwbc ¹ SUNWcepmw SUNWcervl SUNWcexir SUNWckcsr
zh_TW (Taiwan)			SUNWhler SUNWhleu SUNWhbc ¹ SUNWhkcc ¹	SUNWhoaud SUNWhodcv SUNWhodem SUNWhodst SUNWhodte SUNWhoimt SUNWhoman SUNWhorte SUNWhlttk SUNWhxman SUNWhxoft SUNWhxplt SUNWhxfnt SUNWhwbc ¹ SUNWhepmw SUNWhervl SUNWhexir SUNWhkcsr
zh_TW.BIG5 (Taiwan)			SUNW5leu	SUNW5odte SUNW5pmw SUNW5xfnt SUNW5xoft SUNW5xplt

1. Denotes that the package is not delivered for SPARC.

TABLE 5-13 Japanese Package for Localization and Windowing

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
ja/ja_JP.PCK common			SUNWjfp	JSat8xw
			SUNWjfp	SUNWjc0w
			SUNWjc0d	SUNWjwncx
			SUNWjc0r	SUNWjwndt
			SUNWjc0u	SUNWjreg
			SUNWjwncr	SUNWjxcft
			SUNWjwncu	SUNWjxfnt
			SUNWjwnsr	SUNWjxoft
			SUNWjwnsu	SUNWjfxmn
			SUNWjiu8	SUNWjbd
			SUNWjman	SUNWjcs3f
			SUNWjxf3	
			SUNWjxfa	
			SUNWxglj	
ja packages (Japanese)			SUNWjbc	SUNWjadis
			SUNWjrdm	SUNWjadma
			SUNWjeman	SUNWjepmw
			SUNWjeudc	
			SUNWjexir	
			SUNWjmfrn	
			SUNWjoaud	
			SUNWjodev	
			SUNWjodst	
			SUNWjodte	
			SUNWjoint	
			SUNWjorte	
			SUNWjxgld	
			SUNWjxgle	
			SUNWjtltk	
			SUNWjwbcp	
			SUNWjwbk	
			SUNWjxplt	
			SUNWjkcsr	
			SUNWjourn	
			SUNWjxumn	
			SUNWjxpmn	
			SUNWjervl	
			SUNWjffb	
			SUNWjleo	
			SUNWjodem	
			SUNWjsadl	
			SUNWjsxgl	
		SUNWjwacx		
		SUNWjexfa		

TABLE 5-13 Japanese Package for Localization and Windowing (*Continued*)

Locale	OS Common Packages	Win Common Packages	OS Packages	Desktop Packages
ja_JP.PCK pkgs (Japanese)			SUNWjpwnu	SUNWjpxgd
			SUNWjprdm	SUNWjpadm
			SUNWjpman	SUNWjpadi
			SUNWjpadi	
			SUNWjppmw	
			SUNWjpudc	
			SUNWjpxir	
			SUNWjpmfr	
			SUNWjptlt	
			SUNWjpxge	
			SUNWjpxpl	
			SUNWudct	
			SUNWjpkcs	
			SUNWjptlm	
			SUNWjpxpm	
			SUNWjpxum	
			SUNWjprvl	
			SUNWjpf fb	
			SUNWjpleo	
			SUNWjpsal	
		SUNWjpsxg		
		SUNWjpacx		
		SUNWjpxfa		

Description of General Packages

TABLE 5-14 Packages

Package Name	Package Description
SUNWale	Asian Language Environment Common Files
SUNWaled	Asian Language Environment Common Man Pages
SUNWxi18n	X Windows Internationalization Common Package
SUNWxim	X Windows X Input Method Server Package

Description of Korean Package

TABLE 5-15 Korean Package

Package Name	Package Description
SUNWkbcp	Korean Language Environment Binary Compatibility Package
SUNWkler	Korean Language Environment root files
SUNWkleu	Korean Language Environment user files
SUNWkoaud	Korean OpenLook Audio Applications Package
SUNWkodcv	Korean OpenLook Document and Help Viewer Applications Package
SUNWkodem	Korean OpenLook Demo Programs Package
SUNWkodst	Korean OpenLook Deskset Tools Package
SUNWkodte	Korean Core OpenLook Desktop Package
SUNWkoimt	Korean OpenLook Imagetool Package
SUNWkoman	Korean OpenLook Toolkit/Desktop Users Man Pages Package
SUNWkorte	Korean OpenLook Toolkits Runtime Environment Package
SUNWktltk	Korean ToolTalk Runtime Package
SUNWkxman	Korean X Windows Online User Man Pages Package
SUNWkxoft	Korean X Windows Optional Fonts Package
SUNWkxplt	Korean X Windows Platform Software Package
SUNWkxfnt	Korean X Windows Platform required Font Package
SUNWkwbcp	Korean OpenWindows Binary Compatibility Package
SUNWkepmw	Korean (EUC) Power Management OW Utilities
SUNWkcsr	Korean Localizations for Kodak Color Management System Runtime
SUNWkervl	Korean Localizations for SunVideo™ Runtime Support Software
SUNWkexir	Korean Localizations for XIL Runtime Environment
SUNWkiu8	Korean UTF-8 iconv modules for UTF-8
SUNWkuleu	Korean UTF-8 Language Environment user files
SUNWkcoft	Korean/Korean UTF-8 common optional font package
SUNWkuodf	Korean UTF-8 Core OPENLOOK Desktop Package
SUNWkupmw	Korean UTF-8 Power Management OW Utilities
SUNWkuxft	Korean UTF-8 X Windows Platform Required Fonts
SUNWkuxpl	Korean UTF-8 X Windows Platform Software Package

Description of Chinese Package

TABLE 5-16 Chinese Package

Package Name	Package Description
SUNWcbcp	Chinese/PRC Language Environment Binary Compatibility Package
SUNWcleu	Chinese/PRC Language Environment user files
SUNWcoaud	Chinese/PRC OpenLook Audio Applications Package
SUNWcodcv	Chinese/PRC OpenLook Doc and Help Viewer Applications Package
SUNWcodem	Chinese/PRC OpenLook Demo Programs Package
SUNWcodst	Chinese/PRC OpenLook Deskset Tools Package
SUNWcodte	Chinese/PRC Core OpenLook Desktop Package
SUNWcoimt	Chinese/PRC OpenLook Imagetool Package
SUNWcoman	Chinese/PRC OpenLook Toolkit/Desktop Users Man Pages Package
SUNWcorte	Chinese/PRC OpenLook Toolkits Runtime Environment Package
SUNWctltk	Chinese/PRC ToolTalk Runtime Package
SUNWcwbc	Chinese/PRC OpenWindows Binary Compatibility Package
SUNWcxman	Chinese/PRC X Windows Online User Man Pages Package
SUNWcxoft	Chinese/PRC X Windows Optional Fonts Package
SUNWcxplt	Chinese/PRC X Windows Platform Software Package
SUNWcxfmt	Chinese/PRC X Windows Platform required Font Package
SUNWcepmw	Chinese/PRC Power Management OW Utilities
SUNWckcsr	Chinese/PRC for Kodak Color Management System Runtime
SUNWcervl	Chinese/PRC Localizations for SunVideo Runtime Support Software
SUNWcexir	Chinese/PRC Localizations for XIL Runtime Environment

TABLE 5-16 Chinese Package (Continued)

Package Name	Package Description
SUNWhbcp	Chinese/Taiwan Language Environment Binary Compatibility Package
SUNWhler	Chinese/Taiwan Language Environment root files
SUNWhleu	Chinese/Taiwan Language Environment user files
SUNWhkccd	Chinese/Taiwan Kernel based Chinese Console Display package
SUNWhuccd	Chinese/Taiwan User based Chinese Console Display package
SUNWhoaud	Chinese/Taiwan OpenLook Audio Applications Package
SUNWhodcv	Chinese/Taiwan OpenLook Doc and Help Viewer Applications Package
SUNWhodem	Chinese/Taiwan OpenLook Demo Programs Package
SUNWhodst	Chinese/Taiwan OpenLook Deskset Tools Package
SUNWhodte	Chinese/Taiwan Core OpenLook Desktop Package
SUNWhoimt	Chinese/Taiwan OpenLook Imagetool Package
SUNWhoman	Chinese/Taiwan OpenLook Toolkit/Desktop Users Man Pages Package
SUNWhorte	Chinese/Taiwan OpenLook Toolkits Runtime Environment Package
SUNWhtltk	Chinese/Taiwan ToolTalk Runtime Package
SUNWhwbcp	Chinese/Taiwan OpenWindows Binary Compatibility Package
SUNWhxman	Chinese/Taiwan X Windows Online User Man Pages Package
SUNWhxoft	Chinese/Taiwan X Windows Optional Fonts Package
SUNWhxplt	Chinese/Taiwan X Windows Platform Software Package
SUNWhxfnt	Chinese/Taiwan X Windows Platform required Font Package
SUNWhepmw	Chinese/Taiwan Power Management OW Utilities
SUNWhkcsr	Chinese/Taiwan Localize for Kodak Color Management System Runtime
SUNWhervl	Chinese/Taiwan Localizations for SunVideo Runtime Support Software
SUNWhexir	Chinese/Taiwan Localizations for XIL Runtime Environment
SUNW5leu	Chinese/Taiwan BIG5 Language Environment user files
SUNW5odte	Chinese/Taiwan BIG5 Core OPENLOOK Desktop Package
SUNW5pmw	Chinese/Taiwan BIG5 Power Management OW Utilities
SUNW5xfnt	Chinese/Taiwan BIG5 X Windows Platform required Fonts Package
SUNW5xoft	Chinese/Taiwan BIG5 X Windows Optional Fonts Package
SUNW5xplt	Chinese/Taiwan BIG5 X Windows Platform Software Package

Description of Japanese Package

TABLE 5-17 Japanese Package

Package Name	Package Description
SUNWjfpr	Japanese Feature Package root files
SUNWjfpu	Japanese Feature Package usr files
SUNWjeuc	Japanese (EUC) Feature Package usr files
SUNWjpck	Japanese (PCK) Feature Package usr files

TABLE 5-17 Japanese Package (Continued)

Package Name	Package Description
JSat8xw	Japanese Input System - ATOK8
SUNWjc0d	Japanese cs00 user dictionary maintenance tool for CDE Motif
SUNWjc0r	Japanese Kana-Kanji Conversion Server cs00 Root Files
SUNWjc0u	Japanese Kana-Kanji Conversion Server cs00 User Files
SUNWjc0w	Japanese cs00 user dictionary maintenance tool for OPEN LOOK
SUNWjdbas	Japanese CDE base
SUNWjdhev	Japanese CDE HELP VOLUMES
SUNWjdhj	Japanese HotJava Browser for Solaris
SUNWjwnrcr	Wnn6 Client Root Files
SUNWjwnctu	Wnn6 Client Usr Files (common)
SUNWjwnctx	Wnn6 Client X Window System Files
SUNWjwndt	Wnn6 Client User Files for CDE
SUNWjwnsr	Wnn6 Server Root Files
SUNWjwnsu	Wnn6 Server Usr Files
SUNWjreg	Japanese Solaris User Registration
SUNWjxcft	Japanese X Window System common (not required) fonts
SUNWjxfnt	Japanese X Window System required fonts
SUNWjadis	Japanese (EUC) admintool and install software
SUNWjadma	Japanese (EUC) System administration applications
SUNWjbcp	Japanese SunOS 4.x Binary Compatibility
SUNWjebas	Japanese (EUC) CDE base
SUNWjddst	Japanese (EUC) CDE DESKTOP APPS
SUNWjddte	Japanese (EUC) CDE DESKTOP LOGIN ENVIRONMENT
SUNWjdhe	Japanese (EUC) CDE HELP RUNTIME
SUNWjehev	Japanese (EUC) CDE HELP VOLUMES
SUNWjdim	Japanese (EUC) Solaris CDE Image Viewer
SUNWjdrme	Japanese (EUC) CDE README FILES
SUNWjdwm	Japanese (EUC) CDE DESKTOP WINDOW MANAGER
SUNWjepmw	Japanese (EUC) Power Management OW Utilities
SUNWjeudc	Japanese (EUC) User Defined Character tool for Solaris CDE
SUNWjexir	Japanese (EUC) XIL Runtime Environment
SUNWjmfrrn	Japanese (EUC) Motif RunTime Kit
SUNWjoaud	Japanese (EUC) OPEN LOOK Audio applications
SUNWjodcv	Japanese(EUC) OPEN LOOK document and help viewer applications
SUNWjodst	Japanese (EUC) OPEN LOOK deskset tools
SUNWjodte	Japanese (EUC) OPEN LOOK Desktop Environment
SUNWjoimt	Japanese (EUC) OPEN LOOK imagetool
SUNWjorte	Japanese (EUC) OPEN LOOK toolkits runtime environment
SUNWjrdrn	Japanese (EUC) On-Line Open Issues ReadMe
SUNWjxgld	Japanese (EUC) XGL Generic Loadable Libraries
SUNWjpxgd	Japanese (PCK) XGL Generic Loadable Libraries
SUNWjxgle	Japanese (EUC) XGL Runtime Environment
SUNWjtltk	Japanese (EUC) ToolTalk runtime
SUNWjwbcp	Japanese (EUC) OpenWindows binary compatibility
SUNWjwbk	Japanese (EUC) OpenWindows online handbooks
SUNWjxplt	Japanese (EUC) X Window System platform software

TABLE 5-17 Japanese Package (Continued)

Package Name	Package Description
SUNWjpadm	Japanese (PCK) System administration applications
SUNWjpadi	Japanese (PCK) admintool and install software
SUNWjpbas	Japanese (PCK) CDE base
SUNWjpdst	Japanese (PCK) CDE DESKTOP APPS
SUNWjpdte	Japanese (PCK) CDE DESKTOP LOGIN ENVIRONMEN
SUNWjphe	Japanese (PCK) CDE HELP RUNTIME
SUNWjphev	Japanese (PCK) CDE HELP VOLUMES
SUNWjpim	Japanese (PCK) Solaris CDE Image Viewer
SUNWjprme	Japanese (PCK) CDE README FILES
SUNWjpwmm	Japanese (PCK) CDE DESKTOP WINDOW MANAGER
SUNWjppmw	Japanese (PCK) Power Management OW Utilities
SUNWjpudc	Japanese (PCK) User Defined Character tool for Solaris CDE
SUNWjpwnu	Wnn6 Client Usr Files (PCK)
SUNWjpxir	Japanese (PCK) XIL Runtime Environment
SUNWjpmfr	Japanese (PCK) Motif RunTime Kit
SUNWjprdm	Japanese (PCK) On-Line Open Issues ReadMe
SUNWjptlt	Japanese (PCK) ToolTalk runtime
SUNWjpxge	Japanese (PCK) XGL Runtime Environment
SUNWjpxpl	Japanese (PCK) X Window System platform software
SUNWudct	User Defined Character tool for Solaris CDE environment
SUNWjdab	Japanese CDE DTBUILDER
SUNWjfxmn	Japanese Feature English Man Pages for X Window System
SUNWjiu8	Japanese iconv modules for UTF-8
SUNWjman	Japanese Feature Package Man Pages (English)
SUNWjxoft	Japanese X Window System optional fonts
SUNWjeab	Japanese (EUC) CDE DTBUILDER
SUNWjdhed	Japanese (EUC) CDE HELP DEVELOPER ENVIRONMENT
SUNWjedev	Japanese (EUC) Development Environment Package
SUNWjeman	Japanese (EUC) Feature Package Man Pages
SUNWjkcsr	Japanese (EUC) KCMS Runtime Environment
SUNWjoumn	Japanese (EUC) OPEN LOOK toolkit/desktop users man pages
SUNWjxumn	Japanese (EUC) X Window System online user man pages
SUNWjxpmn	Japanese (EUC) X Window System online programmers man pages
SUNWjpab	Japanese (PCK) CDE DTBUILDER
SUNWjphed	Japanese (PCK) CDE HELP DEVELOPER ENVIRONMENT
SUNWjpman	Japanese (PCK) Feature Package Man Pages
SUNWjpkcs	Japanese (PCK) KCMS Runtime Environment
SUNWjptlm	Japanese (PCK) ToolTalk manual pages
SUNWjpxpm	Japanese (PCK) X Window System online programmers man pages
SUNWjpxum	Japanese (PCK) X Window System online user man pages

TABLE 5-17 Japanese Package (Continued)

Package Name	Package Description
SUNWjbdff	Japanese BDF font source
SUNWjcs3f	Japanese JIS X0212 Type1 fonts for printing
SUNWjxf3	Japanese X Window System hinted F3 fonts
SUNWjxfa	Japanese X Window System Font Administrator
SUNWxgljff	Japanese XGL Stroke Font
SUNWjervl	Japanese (EUC) SunVideo Runtime Support Software
SUNWjfffb	Japanese (EUC) Creator Graphics (FFB) XGL Support
SUNWjleo	Japanese (EUC) ZX XGL support
SUNWjodem	Japanese (EUC) OPEN LOOK demo programs
SUNWjsadl	Japanese (EUC) Solstice Admintool launch
SUNWjsxgl	Japanese (EUC) SX XGL Support
SUNWjwacx	Japanese (EUC) AccessX client program
SUNWjexfa	Japanese (EUC) X Window System Font Administrator
SUNWjprvl	Japanese (PCK) SunVideo Runtime Support Software
SUNWjpfffb	Japanese (PCK) Creator Graphics (FFB) XGL Support
SUNWjpleo	Japanese (PCK) ZX XGL support
SUNWjpsal	Japanese (PCK) Solstice Admintool launcher
SUNWjpsxg	Japanese (PCK) SX XGL Support
SUNWjpacx	Japanese (PCK) AccessX client program
SUNWjpxfa	Japanese (PCK) X Window System Font Administrator

TABLE 5-18 and TABLE 5-19 show which Korean files will be installed for each type of installation: core, end user, developer, or the entire installation.

TABLE 5-18 ko Locale

Package Name	Core	End User	Developer	Entire
SUNWale	X	X	X	X
SUNWaled			X	X
SUNWxi18n		X	X	X
SUNWxim		X	X	X
SUNWkler	X	X	X	X
SUNWkleu	X	X	X	X
SUNWkbcpl		X	X	X

TABLE 5-18 ko Locale (Continued)

Package Name	Core	End User	Developer	Entire
SUNWkoaud		X	X	X
SUNWkodcv		X	X	X
SUNWkodem				X
SUNWkodst		X	X	X
SUNWkodte		X	X	X
SUNWkoimt		X	X	X
SUNWkoman				X
SUNWkorte		X	X	X
SUNWktltk		X	X	X
SUNWkwbcp		X	X	X
SUNWkxman			X	X
SUNWkxoft			X	X
SUNWkxplt		X	X	X
SUNWkxfnt		X	X	X
SUNWkepwm		X	X	X
SUNWkkcsr			X	X
SUNWkervl				X
SUNWkexir		X	X	X

TABLE 5-19 ko.UTF-8 Locale

Package Name	Core	End User	Developer	Entire
SUNWale	X	X	X	X
SUNWaled			X	X
SUNWxi18n		X	X	X
SUNWxim		X	X	X
SUNWkiu8	X	X	X	X
SUNWkuleu	X	X	X	X
SUNWkcoft		X	X	X
SUNWkuodf		X	X	X
SUNWkuxft			X	X
SUNWkupmw		X	X	X
SUNWkuxpl		X	X	X

TABLE 5-20, TABLE 5-21, and TABLE 5-22 shows which Chinese files will be installed for each type of installation: core, end user, developer, or the entire installation.

TABLE 5-20 zh Locale

Package Name	Core	End User	Developer	Entire
SUNWale	X	X	X	X
SUNWaled			X	X
SUNWxi18n		X	X	X
SUNWxim		X	X	X
SUNWcleu	X	X	X	X
SUNWcbcp		X	X	X
SUNWcoaud		X	X	X
SUNWcodcv		X	X	X
SUNWcodem				X
SUNWcodst		X	X	X
SUNWcodte		X	X	X
SUNWcoimt		X	X	X
SUNWcoman			X	X
SUNWcorte		X	X	X
SUNWctltk		X	X	X
SUNWcwbcp		X	X	X
SUNWcxman			X	X
SUNWcxoft			X	X
SUNWcxplt		X	X	X
SUNWcxfmt		X	X	X
SUNWcepmw		X	X	X
SUNWckcsr			X	X
SUNWcervl			X	X
SUNWcexir		X	X	X

TABLE 5-21 zh_TW Locale

Package Name	Core	End User	Developer	Entire
SUNWale	X	X	X	X
SUNWaled			X	X

TABLE 5-21 zh_TW Locale (Continued)

Package Name	Core	End User	Developer	Entire
SUNWxi18n		X	X	X
SUNWxim		X	X	X
SUNWhler	X	X	X	X
SUNWhleu	X	X	X	X
SUNWhbcp		X	X	X
SUNWhuccd		X	X	X
SUNWhkccd	X	X	X	X
SUNWhoaud		X	X	X
SUNWhodcv		X	X	X
SUNWhodem				X
SUNWhodst		X	X	X
SUNWhodte		X	X	X
SUNWhoimt		X	X	X
SUNWhoman			X	X
SUNWhorte		X	X	X
SUNWhtltk		X	X	X
SUNWhwbcp		X	X	X
SUNWhxman			X	X
SUNWhxoft			X	X
SUNWhxplt		X	X	X
SUNWhxfnt		X	X	X
SUNWhepmw		X	X	X
SUNWhkcsr			X	X
SUNWhervl				X
SUNWhexir		X	X	X

TABLE 5-22 zh_TW.BIG5 Locale

Package Name	Core	End User	Developer	Entire
SUNWale	X	X	X	X
SUNWaled			X	X
SUNWxi18n		X	X	X
SUNWxim		X	X	X
SUNWhleu	X	X	X	X
SUNW5leu	X	X	X	X
SUNW5odte		X	X	X
SUNW5pmw		X	X	X
SUNW5xoft			X	X
SUNW5xfnt		X	X	X
SUNW5xplt		X	X	X

TABLE 5-23, TABLE 5-24, and TABLE 5-25 shows which Japanese files will be installed for

each type of installation: core, end user, developer, or the entire installation.

TABLE 5-23 ja/ja_JP.PCK Common Packages

Package Name	Core	End User	Developer	Entire
SUNWjfprr	X	X	X	X
SUNWjfpur	X	X	X	X
JSat8xw		X	X	X
SUNWjpadi		X	X	X
SUNWjpadm		X	X	X
SUNWjc0d		X	X	X
SUNWjc0r		X	X	X
SUNWjc0u		X	X	X
SUNWjc0w		X	X	X
SUNWjwncr		X	X	X
SUNWjwncu		X	X	X
SUNWjwncx		X	X	X
SUNWjwndt		X	X	X
SUNWjwnsr		X	X	X
SUNWjwnsu		X	X	X
SUNWjreg		X	X	X
SUNWjxcft		X	X	X
SUNWjxfnt		X	X	X
SUNWudct		X	X	X
SUNWjfxmn			X	X
SUNWjiu8			X	X
SUNWjman			X	X
SUNWjxoft			X	X
SUNWjddf				X
SUNWjcs3f				X
SUNWjxf3				X
SUNWjxfa				X
SUNWxgljff				X

TABLE 5-24 ja Locale

Package Name	Core	End User	Developer	Entire
SUNWjeuc	X	X	X	X
SUNWjepmw		X	X	X
SUNWjeudc		X	X	X
SUNWjewnu		X	X	X
SUNWjexir		X	X	X
SUNWjadis		X	X	X
SUNWjadma		X	X	X
SUNWjbcpc		X	X	X
SUNWjmfrn		X	X	X
SUNWjoaud		X	X	X
SUNWjodcv		X	X	X
SUNWjodst		X	X	X
SUNWjodte		X	X	X
SUNWjoint		X	X	X
SUNWjorte		X	X	X
SUNWjrdrn		X	X	X
SUNWjtltk		X	X	X
SUNWjwbcpc		X	X	X
SUNWjwbk		X	X	X
SUNWjxgld		X	X	X
SUNWjxgle		X	X	X
SUNWjxplt		X	X	X
SUNWjedev			X	X
SUNWjeman			X	X
SUNWjkcsr			X	X
SUNWjourn			X	X
SUNWjtlmn			X	X
SUNWjxpmn			X	X

TABLE 5-24 ja Locale (Continued)

Package Name	Core	End User	Developer	Entire
SUNWjxumn			X	X
SUNWjervl				X
SUNWjexfa				X
SUNWjffb				X
SUNWjleo				X
SUNWjodem				X
SUNWjsadl				X
SUNWjsxgl				X
SUNWjwacx				X

TABLE 5-25 ja_JP.PCK Locale

Package Name	Core	End User	Developer	Entire
SUNWjpck	X	X	X	X
SUNWjppmw		X	X	X
SUNWjpudc		X	X	X
SUNWjpwnu		X	X	X
SUNWjpxir		X	X	X
SUNWjpmfr		X	X	X
SUNWjprdm		X	X	X
SUNWjptlt		X	X	X
SUNWjpxgd		X	X	X
SUNWjpxge		X	X	X
SUNWjpxpl		X	X	X
SUNWjpman			X	X
SUNWjpkcs			X	X
SUNWjptlm			X	X
SUNWjpxpm			X	X
SUNWjprvl				X
SUNWjpxum			X	X

TABLE 5-25 ja_JP.PCK Locale (Continued)

Package Name	Core	End User	Developer	Entire
SUNWjpf fb				X
SUNWjpleo				X
SUNWjpsal				X
SUNWjpsxg				X
SUNWjpxfa				X
SUNWjpacx				X

TABLE 5-26 lists the CDE localization packages.

TABLE 5-26 CDE Packages

Locale	CDE Package	CDE mininum	CDE End User	CDE-Developers
zh_TW (Taiwan)	SUNWhdab	SUNWhdbas	SUNWhdwm	SUNWhdab
	SUNWhdbas	SUNWhddte	SUNWhdhe	
	SUNWhddst	SUNWhdicn	SUNWhddst	
	SUNWhddte		SUNWhdhev	
	SUNWhdhe		SUNWhdim	
	SUNWhdhev		SUNWhreg	
	SUNWhdicn			
	SUNWhdim			
	SUNWhdwm			
	SUNWhreg			
zh_TW.BIG5 (Taiwan)	SUNW5dab	SUNW5dbas	SUNW5dwm	SUNW5dab
	SUNW5dbas	SUNW5ddte	SUNW5dhe	
	SUNW5ddst	SUNW5dicn	SUNW5ddst	
	SUNW5ddte		SUNW5dim	
	SUNW5dhe		SUNWhreg	
	SUNW5dicn			
	SUNW5dim			
	SUNW5dwm			
SUNWhreg				
zh (Chinese)	SUNWcdab	SUNWcdbas	SUNWcdwm	SUNWcdab
	SUNWcdbas	SUNWcddte	SUNWcdhe	
	SUNWcddst	SUNWcdicn	SUNWcddst	
	SUNWcddte	SUNWcdft	SUNWcdhev	
	SUNWcdhe		SUNWcdim	
	SUNWcdhev		SUNWcreg	
	SUNWcdicn			
	SUNWcdim			
	SUNWcdwm			
SUNWcreg				

TABLE 5-26 CDE Packages (Continued)

Locale	CDE Package	CDE minimum	CDE End User	CDE-Developers		
ko (Korean)	SUNWkdab	SUNWkdbas	SUNWkdwm	SUNWkdab		
	SUNWkdbas	SUNWkddte	SUNWkdhe			
	SUNWkddst	SUNWkdien	SUNWkddst			
	SUNWkddte	SUNWkdft	SUNWkdhev			
	SUNWkdhe		SUNWkdim			
	SUNWkdhev		SUNWkreg			
	SUNWkdien					
	SUNWkdim					
	SUNWkdwm					
	SUNWkdft					
	SUNWkreg					
	ko.UTF-8 (Korean)	SUNWkudab	SUNWkudbs		SUNWkudwm	SUNWkudab
		SUNWkudbs	SUNWkuddt		SUNWkudhr	
SUNWkudda		SUNWkudic	SUNWkudda			
SUNWkuddt		SUNWkudft	SUNWkudhv			
SUNWkudhr			SUNWkudim			
SUNWkudhv			SUNWkreg			
SUNWkudic						
SUNWkudim						
SUNWkudwm						
SUNWkudft						
SUNWkreg						

TABLE 5-26 CDE Packages (Continued)

Locale	CDE Package	CDE mininum	CDE End User	CDE-Developers
ja/ja_JP.PCK common	SUNWjdbas SUNWjdhev SUNWjdab	SUNWjdbas	SUNWjdhev	SUNWjdab
ja package	SUNWjebas SUNWjddte SUNWjddst SUNWjdw SUNWjdhe SUNWjehev SUNWjdim SUNWjdrme SUNWjdhed SUNWjeab	SUNWjebas SUNWjddte	SUNWjddst SUNWjdw SUNWjdhe SUNWjehev SUNWjdim SUNWjdrme	SUNWjdhed SUNWjeab
ja_JP.PCK	SUNWjpbas SUNWjpdte SUNWjpdst SUNWjpw SUNWjphe SUNWjphev SUNWjpim SUNWjprme SUNWjpab SUNWjphed	SUNWjpbas SUNWjpdte	SUNWjpdst SUNWjpw SUNWjphe SUNWjphev SUNWjpim SUNWjprme	SUNWjpab SUNWjphed

Asian Localization Packages Disk Space

The following tables display how much hard disk space will be taken by the various packages.

TABLE 5-27 Approximate Disk Space in Megabytes (MB) Required for Software Groups (SPARC)

Software Group	ko	zh	zh_TW	ja	ja_JP.PCK	ja and ja_JP.PCK
Core System Support	107	105	109	56	57	57
End User System Support	224	196	190	346	339	354
Developer System Support	405	307	524	617	608	632
Entire Distribution	481	385	726	798	790	813

TABLE 5-28 Approximate Disk Space in MB Required for Software Groups (x86)

Software Group	ko	zh	zh_TW	ja	ja_JP.PCK	a and ja_JP.PCK
Core System Support	104	105	109	64	64	64
End User System Support	183	183	217	339	339	347
Developer System Support	356	289	597	598	606	622
Entire Distribution	415	349	765	763	763	778

Internationalization Framework in Solaris 2.6

Solaris 2.6 contains several new internationalization features discussed in this chapter, such as:

- Codeset Independence support
- Locale database
- Process code format (wide character expression)
- `libw` and `libintl`
- `ctype` macros
- `genmsg` utility

This chapter also contains information useful for developing internationalized applications, such as:

- Dynamically linked applications
- Solaris 2.6 internationalized APIs

Codeset Independence Support

Before the release of the Solaris 2.6 operating system, the Sun OS and the Solaris internationalization framework supported only Extended UNIX Code (EUC) representation. This prevented support of new encodings that didn't fit the EUC model, such as PC-Kanji in Japan and Big-5 in Taiwan.

Because a large part of the computer market demands non-EUC codeset support, Solaris 2.6 provides a solid framework to enable both EUC and non-EUC codeset support. This support is called *Codeset Independence*, or CSI.

The goal of CSI is to remove EUC dependencies on specific codesets or encoding methods from Solaris OS libraries and commands. The CSI architecture allows the Solaris operating environment to support any UNIX file system safe encoding. CSI supports a number of new codesets, such as UTF-8, PC-Kanji¹, and Big-5.

The CSI Approach

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

Many existing internationalized applications (for example, Motif) automatically inherit CSI support from the underlying system. These applications work in the new locales without modification. OPEN LOOK applications, however, that are XView/OLIT based, don't work in the new locales because XView is codeset-dependent.

CSI is inherently independent from any codesets. However, the following assumptions on file code encodings (codesets) still apply to Solaris 2.6:

- File code is a superset of ASCII.
 - Unicode (16-bits fixed width) cannot be supported as file code.
- NULL (0x00) is not part of multibyte characters for support of null-terminated multibyte character strings.
- Slash / (0x2f) is not part of multibyte characters for support of the UNIX path names.
- Only stateless file code encodings are supported.

CSI-enabled Commands

TABLE 6-1 contains CSI-enabled commands in Solaris 2.6. These commands are marked with CSI capabilities on their `man` page.

1. Japanese Solaris 2.5.1 supports PC Kanji (also known as Shift-JIS).

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

TABLE 6-1 CSI-enabled Commands in Solaris 2.6

<code>/usr/lib/diffh</code>	<code>acctcom</code>	<code>gencat</code>	<code>script</code>
<code>/usr/sbin/accept</code>	<code>apropos</code>	<code>getopt</code>	<code>sdiff</code>
<code>/usr/sbin/reject</code>	<code>batch</code>	<code>getoptcv</code>	<code>settime</code>
<code>/usr/ucb/lpr</code>	<code>bdiff</code>	<code>head</code>	<code>sh</code>
<code>/usr/xpg4/bin/awk</code>	<code>cancel</code>	<code>join</code>	<code>split</code>
<code>/usr/xpg4/bin/cp</code>	<code>cat</code>	<code>jsh</code>	<code>strconf</code>
<code>/usr/xpg4/bin/date</code>	<code>catman</code>	<code>kill</code>	<code>strings</code>
<code>/usr/xpg4/bin/du</code>	<code>chgrp</code>	<code>ksh</code>	<code>sum</code>
<code>/usr/xpg4/bin/ed</code>	<code>chmod</code>	<code>lp</code>	<code>tabs</code>
<code>/usr/xpg4/bin/edit</code>	<code>chown</code>	<code>man</code>	<code>tar</code>
<code>/usr/xpg4/bin/egrep</code>	<code>cmp</code>	<code>mkdir</code>	<code>tee</code>
<code>/usr/xpg4/bin/env</code>	<code>col</code>	<code>msgfmt</code>	<code>touch</code>
<code>/usr/xpg4/bin/ex</code>	<code>comm</code>	<code>news</code>	<code>tty</code>
<code>/usr/xpg4/bin/expr</code>	<code>compress</code>	<code>nroff</code>	<code>uncompress</code>
<code>/usr/xpg4/bin/fgrep</code>	<code>cpio</code>	<code>pack</code>	<code>unexpand</code>
<code>/usr/xpg4/bin/grep</code>	<code>csh</code>	<code>paste</code>	<code>uniq</code>
<code>/usr/xpg4/bin/lm</code>	<code>csplit</code>	<code>pcat</code>	<code>unpack</code>
<code>/usr/xpg4/bin/ls</code>	<code>cut</code>	<code>pg</code>	<code>wc</code>
<code>/usr/xpg4/bin/more</code>	<code>diff</code>	<code>printf</code>	<code>whatis</code>
<code>/usr/xpg4/bin/mv</code>	<code>diff3</code>	<code>priocntl</code>	<code>write</code>
<code>/usr/xpg4/bin/nice</code>	<code>disable</code>	<code>ps</code>	<code>xargs</code>
<code>/usr/xpg4/bin/nohup</code>	<code>echo</code>	<code>pwd</code>	<code>zcat</code>
<code>/usr/xpg4/bin/od</code>	<code>expand</code>	<code>rcp</code>	
<code>/usr/xpg4/bin/pr</code>	<code>file</code>	<code>red</code>	
<code>/usr/xpg4/bin/rm</code>	<code>find</code>	<code>remsh</code>	
<code>/usr/xpg4/bin/sed</code>	<code>fold</code>	<code>rksh</code>	
<code>/usr/xpg4/bin/sort</code>	<code>ftp</code>	<code>rmdir</code>	
<code>/usr/xpg4/bin/tail</code>		<code>rsh</code>	
<code>/usr/xpg4/bin/tr</code>			
<code>/usr/xpg4/bin/vedit</code>			
<code>/usr/xpg4/bin/vi</code>			
<code>/usr/xpg4/bin/view</code>			

Solaris 2.6 CSI-enabled Libraries

Nearly all functions in Solaris 2.6 `libc` (`/usr/lib/libc.so`) are CSI-enabled. However, the following functions in `libc` are not CSI-enabled because they are EUC dependent functions:

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

Also the following macros are not CSI-enabled because they are EUC dependent:

- `csetno()` `wcsetno()`

Solaris 2.6 `libgen` (`/usr/ccs/lib/libgen.a`) are internationalized, but not CSI enabled.

Solaris 2.6 `libcurses` (`/usr/ccs/lib/libcurses.a`) are internationalized, but not CSI enabled.

Locale Database

The locale database format and structure in Solaris 2.6 have changed from previous Solaris releases. The locale database is private and subject to change in a future release. Therefore, when developing an internationalized application, do not directly access the locale database. Instead you should use the Solaris internationalization APIs.

Note – When using Solaris 2.6, use the locale databases that are included with Solaris 2.6. Do not use locales from previous Solaris versions.

Process Code Format

The process code format in Solaris 2.6 is private and subject to change in a future release. Therefore, when developing an international application, do not assume the process code format will be the same. Instead you should use the Solaris internationalization APIs which are described in TABLE 6-3 on page 108.

Dynamically Linked Applications

Solaris 2.6 users can choose how to link applications with the system libraries, such as `libc`, by using dynamic linking or static linking. However, any application that requires internationalization features in the system libraries must be dynamically

linked. If the application has been statically linked, the operation to set the locale to other than C and POSIX using the `setlocale` function will fail. Statically linked applications can be operated only in C and POSIX locales.

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

For example, if you type:

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

the command displays the following message:

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

The message indicates the `/sbin/sh` command is not a dynamically linked program. Also, if you type:

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

the command displays the following message:

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

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

To summarize, if the message from the `ldd` command to the application does not contain a `libc.so.1` entry, it indicates that the application has been statically linked with `libc`. In that case, you need to change the command line options to the linker so that dynamic linking is used instead, then re-link the application.

libw and libintl

In the Solaris 2.6 release, the implementation of `libw` and `libintl` has been moved to `libc`. The shared objects `libw.so.1` and `libintl.so.1` are provided as filters on `libc.so.1`, and the archives `libw.a` and `libintl.a` are provided as links to an empty archive.

The shared objects insure runtime compatibility for existing applications, and, together with the archives, provide compilation environment compatibility for building applications. However, it is no longer necessary to build applications against `libw` or `libintl`.

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

TABLE 6-2 shows the stub entry points in `libw` and `libintl`:

TABLE 6-2 Stub Entry Points in `libw` and `libintl`

<code>libw</code>	<code>fgetwc</code>	<code>fgetws</code>	<code>fputwc</code>	<code>fputws</code>	<code>getwc</code>
	<code>getwchar</code>	<code>getws</code>	<code>isenglish</code>	<code>isideogram</code>	<code>isnumber</code>
	<code>isphonogram</code>	<code>isspecial</code>	<code>iswalnum</code>	<code>iswalpha</code>	<code>iswcntrl</code>
	<code>iswctype</code>	<code>iswdigit</code>	<code>iswgraph</code>	<code>iswlower</code>	<code>iswprint</code>
	<code>iswpunct</code>	<code>iswspace</code>	<code>iswupper</code>	<code>iswxdigit</code>	<code>putwc</code>
	<code>putwchar</code>	<code>putws</code>	<code>strtows</code>	<code>towlower</code>	<code>towupper</code>
	<code>ungetwc</code>	<code>watoll</code>	<code>wscat</code>	<code>wchr</code>	<code>wscmp</code>
	<code>wscoll</code>	<code>wscopy</code>	<code>wscspn</code>	<code>wcsftime</code>	<code>wcslen</code>
	<code>wscncat</code>	<code>wscncmp</code>	<code>wscncpy</code>	<code>wcspbrk</code>	<code>wcsrchr</code>
	<code>wcsspn</code>	<code>wctod</code>	<code>wctok</code>	<code>wctol</code>	<code>wctoul</code>
	<code>wcswcs</code>	<code>wcswidth</code>	<code>wcsxfrm</code>	<code>wctype</code>	<code>wcwidth</code>
	<code>wscasecmp</code>	<code>wscat</code>	<code>wchr</code>	<code>wscmp</code>	<code>wscol</code>
	<code>wscoll</code>	<code>wscopy</code>	<code>wscspn</code>	<code>wsdup</code>	<code>wslen</code>
	<code>wsncasecmp</code>	<code>wsncat</code>	<code>wsncmp</code>	<code>wsncpy</code>	<code>wsprbrk</code>
	<code>wsprintf</code>	<code>wsrchr</code>	<code>wsscanf</code>	<code>wssp</code>	<code>wstod</code>
	<code>wstok</code>	<code>wstol</code>	<code>wstoll</code>	<code>wstostr</code>	<code>wsxfrm</code>
<code>libintl</code>	<code>bindtextdomain</code>	<code>dcgettext</code>	<code>dgettext</code>	<code>gettext</code>	<code>textdomain</code>

cctype Macros

Character classification and character transformation macros are defined in `/usr/include/cctype.h`. Solaris 2.6 provides a new set of `cctype` macros. The new macros support character classification and transformation semantics defined by XPG4. To access the new set of macros, one of the following conditions must be met:

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

This means that all XPG4 and XPG4.2 applications will automatically have the new macros. Since `_XOPEN_SOURCE`, `_XOPEN_VERSION`, and `_XOPEN_SOURCE_EXTENDED` will bring in extra XPG4 related features in addition to new `cctype` macros, non-XPG4 or XPG4.2 applications should use `__XPG4_CHAR_CLASS__`.

There are corresponding `cctype` functions. The Solaris 2.6 functions also support XPG4 semantics.

Refer to the `cctype` man page for details.

Internationalization APIs in `libc`

Solaris 2.6 offers two sets of APIs:

- multibyte (file codes)
- wide characters (process code)

Applications do their processing in wide character codes.

When a program takes input from a file, convert your file's multibyte data into wide character process code with the `mbtowc` and `mbtowcs` APIs. To convert the file output data from wide character format into multibyte format, use the `wcstombs` and `wctomb` APIs.

TABLE 6-3 shows a list of internationalization APIs included in Solaris 2.6.

TABLE 6-3 internationalization APIs in libc

API Type	Library Routine	Description
Messaging Functions	catclose()	Close a message catalog.
	catgets()	Read a program message.
	catopen()	Open a message catalog.
	dgettext()	Get a message from a message catalog with domain specified.
	dcgettext()	Get a message from a message catalog with domain and Category specified.
	textdomain()	Set and query the current domain.
	bindtextdomain()	Bind the path for a message domain.
Code conversion	iconv()	Convert codes.
	iconv_close()	Deallocate the conversion descriptor.
	iconv_open()	Allocate the conversion descriptor.
Regular expression	regcomp()	Compile the regular expression.
	regexec()	Execute the regular expression matching.
	regerror()	Provide a mapping from error codes to error message.
	regfree()	Free memory allocated by regcomp().
Wide character class	wctype()	Define character class.
Locale related	setlocale()	Modify and query a program's locale.
	nl_langinfo()	Get language and cultural information of current locale.
	localeconv()	Get monetary and numeric formatting information of current locale.
Character classification	isalpha()	Is character an alphabetic character?
	isupper()	Is character uppercase?

TABLE 6-3 internationalization APIs in libc (Continued)

API Type	Library Routine	Description
	islower()	Is character lowercase?
	isdigit()	Is character a digit?
	isxdigit()	Is character a hex digit?
	isalnum()	Is character an alphabetic character or digit?
	isspace()	Is character a space?
	ispunct()	Is character a punctuation mark?
	isprint()	Is character printable?
	iscntrl()	Is character a control character?
	isascii()	Is character an ASCII character?
	isgraph()	Is character a visible character?
	isphonogram()	Is wide character a phonogram?
	isideogram()	Is wide character an ideogram?
	isenglish()	Is wide char in English alphabet from a supplementary codeset?
	isnumber()	Is wide character a digit from a supplementary codeset?
	isspecial()	Is special wide character from a supplementary codeset?
	iswalpha()	Is wide character an alphabetic character?
	iswupper()	Is wide character uppercase?
	iswlower()	Is wide character lowercase?
	iswdigit()	Is wide character a digit?
	iswxdigit()	Is wide character a hex digit?
	iswalnum()	Is wide character an alphabetic character or digit?
	iswspace()	Is wide character white space?
	iswpunct()	Is wide character a punctuation mark?
	iswprint()	Is wide character a printable character?
	iswgraph()	Is wide character a visible character?
	iswcntrl()	Is wide character a control character?
	iswascii()	Is wide character an ASCII character?

TABLE 6-3 internationalization APIs in libc (*Continued*)

API Type	Library Routine	Description
Character transformation	toupper()	Convert a lowercase character to uppercase.
	tolower()	Convert an uppercase character to lowercase.
	towupper()	Convert a lowercase wide character to uppercase.
	towlower()	Convert an uppercase wide character to lowercase.
Character collation	strcoll()	Collate character strings.
	strxfrm()	Transform character strings for comparison.
	wscoll()	Collate wide char strings.
	wcsxfrm()	Transform wide char strings for comparison.
Monetary handling	strfmon()	Convert monetary value to string representation.
Date and Time handling	getdate()	Convert user format date and time.
	strftime()	Convert date and time to string representation.
	strptime()	Date and time conversion.
Multibyte handling	mblen()	Get length of multibyte character.
	mbtowc()	Convert multibyte to wide character.
	mbstowcs()	Convert multibyte string to wide character string.
Wide Characters	wscat()	Concatenate wide char strings.
	wcsncat()	Concatenate wide char strings to length <i>n</i> .
	wsdup()	Duplicate wide char string.
	wscmp()	Compare wide char strings.
	wcsncmp()	Compare wide char strings to length <i>n</i> .
	wscopy()	Copy wide char strings.
	wcsncpy()	Copy wide char strings to length <i>n</i> .
	wcschr()	Find character in wide char string.

TABLE 6-3 internationalization APIs in `libc` (Continued)

API Type	Library Routine	Description
	<code>wcsrchr()</code>	Find character in wide char string from right.
	<code>wcslen()</code>	Get length of wide char string.
	<code>wscol()</code>	Return display width of wide char string.
	<code>wcsspn()</code>	Return span of one wide char string in another.
	<code>wscspn()</code>	Return span of one wide char string not in another.
	<code>wcspbrk()</code>	Return pointer to one wide char string in another.
	<code>wcstok()</code>	Move token through wide char string.
	<code>wcswcs()</code>	Find string in wide character string.
	<code>wcstombs()</code>	Convert wide character string to multibyte string.
	<code>wctomb()</code>	Convert wide character to multibyte character.
	<code>wcwidth()</code>	Determine number of column positions of a wide character.
	<code>wcswidth()</code>	Determine number of column positions of a wide char string.
Wide Formatting	<code>wsprintf()</code>	Generate wide char string according to format.
	<code>wsscanf()</code>	Interpret wide char string according to format.
Wide Numbers	<code>wcstol()</code>	Convert wide char string to long integer.
	<code>wcstoul()</code>	Convert wide char string to unsigned long integer.
	<code>wcstod()</code>	Convert wide char string to double precision.
Wide Strings	<code>wscasecmp()</code>	Compare wide char strings, ignores case differences.
	<code>wncasecmp()</code>	Compare wide char strings to length <i>n</i> (ignores case).
Wide Standard I/O	<code>fgetwc()</code>	Get multibyte char from stream, convert to wide char.

TABLE 6-3 internationalization APIs in `libc` (*Continued*)

API Type	Library Routine	Description
	<code>getwchar()</code>	Get multibyte char from <code>stdin</code> , convert to wide char.
	<code>fgetws()</code>	Get multibyte string from stream, convert to wide char.
	<code>getws()</code>	Get multibyte string from <code>stdin</code> , convert to wide char.
	<code>fputwc()</code>	Convert wide char to multibyte char, puts to stream.
	<code>putwchar()</code>	Convert wide char to multibyte char, puts to <code>stdin</code> .
	<code>fputws()</code>	Convert wide char to multibyte string, puts to stream.
	<code>putws()</code>	Convert wide char to multibyte string, puts to <code>stdin</code> .
	<code>ungetwc()</code>	Push a wide char back into input stream.

genmsg Utility

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

from the information it finds. For example:

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

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

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

For more information, see the `genmsg` man page.

Note – The material in this section is used with permission from *Creating Worldwide Software: Solaris International Developer's Guide*, 2nd edition by Bill Tuthill and David A. Smallberg, published by Sun Microsystems Press/Prentice Hall. (c)1997 Sun Microsystems, Inc.

Writing Internationalized Code

This chapter describes some specific steps that you should take to internationalize applications. The material is divided into four main topics: text and codesets, formatting and collation, user messages, and nonglobal locales.

Linking

Some internationalization components depend on dynamic linking to function correctly. The default when compiling and linking in the Solaris environment is dynamic linking. Take care not to specify static linking.

Text and Codesets

Call `setlocale()`

The SunOS system supports the POSIX/ANSI C function `setlocale()`, which initializes language and cultural conventions. Most applications should set the locale category `LC_CTYPE` except those not concerned with character interpretation, such as block I/O to disk or network. To control the dynamic handling of different codesets in an application, add these lines to your code:

```
#include <locale.h>
main() {
    (void) setlocale(LC_CTYPE, "");
}
```

Among other things, this ensures that European accented characters such as ö are correctly identified with an `isalpha()` library call. Note that the empty string argument indicates that the application should set its codeset according to the environment variable `LC_ALL`, `LC_CTYPE`, or `LANG`—in that order of precedence. If none of these environment variables is set, the default locale is `C`, which results in old-style UNIX behavior.

`LC_CTYPE` affects the behavior of various `ctype(3)` library routines. The `LC_CTYPE` locale category may also affect other functions, including wide-character handling.

In most cases library packages should rely on the programmer to call `setlocale()` inside the application. Applications that fail to call `setlocale()` would simply fail to get international features.

To set all the above locale categories at the same time, use the `LC_ALL` argument to `setlocale()` instead of just `LC_CTYPE`. In practice, most applications should set the `LC_ALL` category once and for all.

Make Software 8-bit Clean

Programs shouldn't alter the most significant bit of a `char`. The computer industry used this bit for parity many years ago, but it didn't work out well—data got corrupted because software ignored the parity bit. Now standards committees have decided to define 8-bit codesets, which means you have to clean up your code now. Here are some problems to look for.

Code that explicitly uses the most significant bit for its own purposes is said to be “dirty”. There may be valid reasons for altering the most significant bit, but dirty code often involves setting and clearing private flags:

```
#define INVERSE 0x80          /* bad practice */
char c;
c |= INVERSE;
```

Find another way to encode this information. A trick used several times in the operating system was to extend this data type to be unsigned short or unsigned int, and later set the top bit of the new data type.

Code that assumes characters are only seven bits long is dirty. Here's an example of masking off the most significant bit on the assumption it's just the parity bit:

```
c = *(string+i) & 0x7F; /* bad practice */
```

A useful exercise is to search your code for constants like “0x80”, “0x7f”, “0200”, “0177”, “127”, and “128”. These constants often highlight problematic code immediately, if such bit patterns are used in conjunction with character handling.

Code that assumes a particular character range, such as:

```
if (c >= 'a' && c <= 'z')/* bad practice */
```

must be corrected to:

```
if (islower(c))
```

Use codeset independent routines found in `<ctype.h>` such as `isalpha()`, `isprint()`, and so on. Software should have been using these functions all along, as they were always needed for portability to IBM's EBCDIC codeset. The SunOS system also provides wide-character equivalents such as `iswalpha()` and `iswprint()`.

Fix code that assumes characters fall in the range 0-127 by extending the range of such tables:

```
static int hashtable[127]; /* bad practice */
```

For example, the above declaration would be better coded as follows:

```
#include <limits.h>
static int hashtable[UCHAR_MAX];
```

`UCHAR_MAX` is defined in `<limits.h>` on all ANSI C conforming systems.

Watch for Sign Extension Problems

One issue that is sometimes invisible to the programmer is the way the C compilers default to using `signed` for all fundamental data types. This can sometimes cause substantial problems in both application and library code.

Code that casts `char` to other lengths may be dirty. Because the `char` data type is signed in SunOS, when a `char` variable holds an 8-bit character that has the most significant bit set, sign extension takes place during assignment. Needless to say, a negative integer might cause problems later on:

```
int i;
char c = 0xa0;
i = c; /* i is now negative */
```

Do not pass raw characters to functions that require `short`, `int`, or `long` arguments. This is bad practice because of the sign extension problem. For example, the following code is incorrect, as it produces a negative integer index into the C library `__ctype` table. This is because the functions are actually macros that generate stubs of in-line code, which assume the argument is an integer, and propagate the sign bit accordingly.

```
char ch;
isascii(ch);
```

The code above could be written like this:

```
unsigned char ch;
isascii(ch);
```

Watch for the use of unadorned `chars`. Unfortunately they have probably been used extensively throughout most code. It is therefore a nontrivial task to change all `char` data to `unsigned char`, especially as this might garner some `lint` or compiler warnings.

So,

```
char ch;
ch = 0xA0;
```

is better written as:

```
unsigned char ch;
ch = 0xA0;
```

On the other hand,

```
char *cp;
while (isspace(*cp)) {
```

is written as:

```
char *cp;
while (isspace((unsigned char)*cp)) {
```

Although all this may sound like a lot of work, in many cases existing code executes correctly in 8-bit mode *without* any changes to the code. You are primarily looking for lazy coding habits that assume ASCII is the only form of character encoding available. When you fix problems, they are usually easy to test using the Compose key of the Type-4, Type-5, PC-AT101, and PC-AT102 keyboard.

Note that the C compiler does not support 8-bit or multi-byte characters in object names—that is, names of routines, variables, and so forth—although it does allow you to initialize 8-bit or multi-byte data in strings.

Use ctype Library Routines

As mentioned previously, text processing software must avoid hard-coded character ranges. Upper- and lower-case letters, punctuation marks, numeric digits, and spaces should be defined using library routines under `<ctype.h>`, rather than with hard-coded character ranges:

TABLE 7-1 Library Routines for Codeset Independence

Routine	Character
<code>isalpha(c)</code>	Letter
<code>isupper(c)</code>	Capital letter
<code>islower(c)</code>	Lower case letter
<code>isdigit(c)</code>	Digit from 0-9
<code>isxdigit(c)</code>	Hexadecimal digit from 0-f
<code>isalnum(c)</code>	Alphanumeric (letter or digit)
<code>isspace(c)</code>	White space character
<code>ispunct(c)</code>	Punctuation mark
<code>isprint(c)</code>	Printable character
<code>isctrl(c)</code>	Control character
<code>isascii(c)</code>	7-bit character
<code>isgraph(c)</code>	Visible graphics character

Formats

Many different formats are employed throughout the world to represent date, time, currency, numbers, and units. These formats should not be hard-wired into your code. Instead, programs should call `setlocale()`, then the various locale specific format routines, leaving format design to localization work for each country or language.

Time and Date Formats

The secret to producing time and date formats valid in many locales is the `strftime()` library routine. First set the program clock by calling `time()`, then populate a `tm` structure by calling `localtime()`. Pass this structure to `strftime()`, along with a format for date and time, plus a holding buffer:

```
#include <locale.h>
#include <libintl.h>
#include <stdio.h>
#include <time.h>
main()
{
    time_t clock, time();
    struct tm *tm, *localtime();
    char buf[128];

    setlocale(LC_ALL, "");
    clock = time((time_t *)0);
    tm = localtime(&clock);
    strftime(buf, sizeof(buf), "%c", tm);
    printf("%s\n", buf);
}
```

Recommended formats are `%c` for the local short form of date and time, or `%C` for the local long form. Also, `%x` produces the local date form (numeric), and `%X` yields the local time form. If you try out the program above, your results will look something like this:

```
% setenv LC_TIME de
% a.out
Mo, 16. Mär 1992, 19:19:19 Uhr PST
% setenv LC_TIME fr
% a.out
lun, 16 mar 1992, 19:19:20 PST
```

Unfortunately many often-used combinations of date and time are missing from the standard. Neither short nor long form of the local date is available, and there is no abbreviation for time without seconds or time zone.

Currency and Number Formats

Use `localeconv(3)` function to obtain currency formats. It reads formatting conventions of the current locale to populate an `lconv` structure, then returns a pointer to the filled-in object.

The only way to properly represent monetary amounts using the facilities of Standard C is to laboriously build a string using information extracted from an `lconv` structure returned by `localeconv()`. Fortunately, XPG4 standardizes a function analogous to `strftime()`, named `strfmon()`, whose behavior depends on the `LC_MONETARY` category. This program uses `strfmon()` to format monetary amounts.

```
#include <locale.h>
#include <monetary.h>
#include <stdio.h>
int main()
{
    double cost;
    char buffer[100];
    setlocale(LC_ALL, "");
    scanf("%lf", &cost);
    strfmon(buffer, sizeof(buffer), "%n\t%i", cost, cost);
    printf("%s\n", buffer);
}
```

As with `strftime()`, the formatted string is placed in a buffer. The `%n` format item formats the amount in the locale's national format, and `%i` uses the international currency code specified in ISO 4217.

```
% echo 12345.678 | env LANG=en_US a.out
$12,345.68 USD12,345.68
% echo 12345.678 | env LANG=sv a.out
12.346 kr 12.346 SEK
```

The behavior of the `%f` format item for `scanf()` and `printf()` is affected by the `LC_NUMERIC` category. Swedish uses a comma (,) as the radix character and a period (.) as the thousands separator, so `scanf()` expects a comma where an English speaker would use a period. Be careful here: `scanf()` in the Swedish locale (or any similar locale) will stop reading upon encountering a period, just as it would stop at a comma in the C locale.

Note – The material in this section is used with permission from *Creating Worldwide Software: Solaris International Developer's Guide*, 2nd edition by Bill Tuthill and David A. Smallberg, published by Sun Microsystems Press/Prentice Hall. 1997.

Collation

For string collation, sort orders may vary for different languages. Programs should use the `strcoll()` or `strxfrm()` library routine to perform string comparisons, which use locale-specific collation order.

Replace `strcmp()` with `strcoll()`

Alphabetic ordering varies from one language to another. For example, in Spanish ñ immediately follows n, and digraphs ch and ll immediately follow c and l, respectively. In German the ligature ß is collated as if it were ss. Swedish has additional unique characters following z. Danish and Norwegian have additional characters æ, ø following z.

The traditional library routine for comparing strings, `strcmp()`, remains unchanged. Because it uses ASCII order, `strcmp()` places “a” after “Z” even in English. This ordering is often unacceptable.

By contrast, the new library routines `strcoll()` and `strxfrm()` can produce any sort order you want. Use `strcoll()` to compare strings, or `strxfrm()` to transform strings to ones that collate correctly.

Fortunately `strcoll()` takes the same parameters and returns the same values as `strcmp()`. Unfortunately `strcoll()` does a lot more work, and is consequently slower. To speed up applications that compare strings frequently, use `strxfrm()` to store transformed strings into arrays that collate more efficiently.

This program reads standard input, builds a binary tree in the correct order using `strcoll()` to compare strings, then prints out the binary tree. This code may be used for tasks such as listing files in a subwindow.

```
#include <locale.h>
#include <stdio.h>
#include <string.h>
```

```

struct tnode { /* node of binary tree */
char *line;
int count;
struct tnode *left, *right;
};

main() /* collate: sort a list of lines using strcoll() */
{
    struct tnode *root, *tree();
    char line[BUFSIZ];
    root = NULL;
    (void)setlocale(LC_ALL, "");
    while (fgets(line, BUFSIZ, stdin))
        root = tree(root, line);
    treeprint(root);
}

struct tnode *
tree(p, line) /* install line at or below tree pointer */
struct tnode *p;
char *line;
{
    char *cp, *malloc(), *strcpy();
    int cond;
    if (p == NULL) {
        p = (struct tnode *)malloc(sizeof(struct tnode));
        if ((cp = malloc(strlen(line)+1)) != NULL)
            strcpy(cp, line);
        p->line = cp;
        p->count = 1;
        p->left = p->right = NULL;
    }
    else if ((cond = strcoll(line, p->line)) == 0)
        p->count++;
    else if (cond < 0)
        p->left = tree(p->left, line);
    else /* cond > 0 */
        p->right = tree(p->right, line);
    return(p);
}
treeprint(p) /* print tree recursively starting at p */
struct tnode *p;
{
    if (p != NULL) {
        treeprint(p->left);

```

```
while (p->count--)\n    printf("%s", p->line);\n    treeprint(p->right);\n}\n}
```

Messaging for Program Translation

One of the most critical tasks in software internationalization is providing messages that can be translated easily. Messages are what users see first: help text, button labels, menu items, usage summaries, error diagnostics, and so forth.

This chapter shows you how to write an application that produces internationalized messages. Your program consults an external catalog of messages to determine what strings to present to the user. You provide one message catalog for each locale you support, but you have only one version of the program.

The ease of message localization can vary greatly. In a well-designed application, nontechnical people can translate message files into their native languages. In a noninternationalized application, engineers fluent in a language must translate every explicit string that will be seen by a user, then recompile the code. In an internationalized application, a lookup function retrieves any such string from a message catalog: a database of text strings that is easy to compose, translate, and access. Because the contents of a message catalog are separate from application code, text can be selected by locale at runtime without altering the code itself.

Two similar (but incompatible) methods for international messaging in Solaris are `catgets()` from the XPG4 standard and `gettext()` from the POSIX.1b and UniForum proposals. The primary difference between them is the way that messages in the catalog are indexed: in essence, you pass `catgets()` a message number, but you pass `gettext()` a string.

If there are two messaging schemes to choose between, which should you use? Each has its strengths and weaknesses, and adherents to argue for it. There's a lot to be said for standardization, though. X/Open considered both and chose `catgets()`. For maximal portability of your application to other platforms, then, we recommend that you use that scheme.

This section presents the issues involved with messaging:

- **Messaging using `catgets()`.** For the steps involved in enabling messaging using the XPG4 scheme, see “Messaging Using `catgets()`” on page 125.
- **Messaging using `gettext()`.** For the steps involved in enabling messaging using the nonstandard scheme, see “Messaging Using `gettext()`” on page 132.

- **Problem areas.** Some common pitfalls are discussed in “Problem Areas” on page 136.
- **Messaging in languages other than C.** If you are writing applications in a language other than C, you can still create and access message catalogs. See “Other Programming Languages” on page 141.

Messaging Using `catgets()`

When creating internationalized applications, developers usually write text strings (error messages, text for buttons and menus, and so forth) in their native language, for later translation into other languages. Solaris lets you use any language as native.

Here are the steps to internationalize and localize text handling:

1. Change source code to `#include <nl_types.h>`, then call `catopen()` to open a message catalog and call `catgets()` to retrieve strings from the catalog.
2. Extract native language text strings from the `catgets()` calls and store them in a source message catalog. You must assign each message a unique number that will appear in both the source catalog and any `catgets()` call that refers to that message.
3. Translate the strings in the source message catalog into a target language.
4. Transform the translated source message catalog into a binary message catalog, using the `genocat(1)` utility. Install the binary catalog.

Locating Message Catalogs

After you have established the locale, you will want to open the appropriate message catalog immediately, so that any startup problems that produce error messages will do so in the proper language. Use `catopen()` for this:

```
#include <locale.h>
#include <nl_types.h>
nl_catd catd;
int main()
{
    (void) setlocale(LC_ALL, "");
    catd = catopen("demo", NL_CAT_LOCALE);
    ...
}
```

The `catopen()` function looks for the message catalog according to these rules:

1. The locale used is the value of `LC_MESSAGES` as established by `setlocale()`. (The only other choice for `catopen()`'s second argument is 0, meaning that locale used is the value of the `LANG` environment variable.)
2. The first argument and the `NLSPATH` environment variable are used to locate the catalog. (If the first argument contains `/`, then `LC_MESSAGES` and `NLSPATH` are ignored; instead, the first argument is the absolute path name of the catalog. You almost never want to do this.)

The `NLSPATH` variable is a colon-separated list of filename patterns, for instance:

```
/usr/lib/locale/%L/LC_MESSAGES/%N.cat:/tmp/%N.%L.cat
```

In these patterns, `catopen()` replaces `%N` with its first argument, and `%L` with the prevailing locale. If the locale is set to French, for example, then `catopen()` uses the file named `/usr/lib/locale/fr/LC_MESSAGES/demo.cat` if it exists. Failing that, it will try `/tmp/demo.fr.cat`. The first pattern in this example is the same one that `catopen()` uses if `NLSPATH` is not set. The second pattern is one a developer might use while testing an application's messaging ability.

Although you need not name a message file after its application, this convention is recommended. It simplifies maintenance to have `catopen()`'s first argument be the same as the application name.

The header `<nl_types.h>` defines the (integral) type `nl_catd`. The return value of `catopen()`, a *catalog descriptor*, should be stored in a variable of this type, since it will be passed to every `catgets()` call that looks up messages in the selected catalog. Because you use this variable throughout a program, declare `catd` globally.

If `catopen()` fails, it returns `(nl_catd)-1`. Of course, a good application should test for this and note the error. However, you can safely pass this failure value in calls to `catgets()`, which will simply return the default strings you provide instead of the localized strings.

An open catalog consumes system resources: a file descriptor and some memory for indexes into the catalog. When your program exits, these resources are automatically released. If you want to release them explicitly, call `catclose()`:

```
catclose(catd);
```


Using `catgets()`

To retrieve strings from a message catalog, you call `catgets()`, passing it the catalog descriptor returned by `catopen()`, an index into the catalog to select the message string, and a default string to use instead if there's a problem. The index is the most troublesome part of the `catgets()` interface.

In essence, to use `catgets()`, you must assign a number to each message your program will produce. This requirement alone accounts for the most noticeable change in appearance between a noninternationalized and an internationalized version of a program. It can also lead to a maintenance headache if these numbers are not well managed. The only support the XPG4 messaging scheme gives you is the ability to partition your messages into sets. You may, for example, decide that the button label “Edit” is message number 37 of set number 4. How many sets you use, and what you use them for, is up to you. On some projects, each developer uses a different set number; on others, each subsystem of an application is given its own set number.

Here is an example of how to use `catgets()`:

```
/* Assume catd is the return value of catopen() */
printf(catgets(catd, 3, 27, "Invoice\n"));
```

If all is well, `catgets()` will retrieve message number 27 of set number 3 from the message catalog referred to by `catd`, returning a `char *` value pointing to the message. If there is no message 27 in set 3, or if there is no set 3, or if `catd` is `-1`, then `catgets()` returns its last argument, the default string. The intent is that message 27 of set 3 in the catalog is a translation of “Invoice\n”; if the translation is unavailable, the program will use the English “Invoice\n”, since that’s better than nothing.

Although not true for Solaris, on some platforms `catgets()` returns a pointer to storage that may be overwritten on each call. This implies that for maximal portability, use or copy the value returned by one call of `catgets()` before you call it again:

```
char buffer[100];
char *p, *q;
/*
 * This is not portable:
 */
printf("%s %s", catgets(catd, 1, 1, "Name"),
        catgets(catd, 1, 2, "Age"));
/*
 * This is not portable either:
 */
p = catgets(catd, 1, 1, "Name");
q = catgets(catd, 1, 2, "Age");
printf("%s %s", p, q);
/*
 * This is portable, provided buffer is big enough:
 */
strcpy(buffer, catgets(catd, 1, 1, "Name"));
printf("%s %s", buffer, catgets(catd, 1, 2, "Age"));
```

Create the Source Message Catalog

Once you know what your messages are, create a source message catalog for your native language. Suppose the following program fragment shows all the messages some program will produce:

```
printf(catgets(catd, 1, 1, "Hello"));
printf(catgets(catd, 3, 4, "Age: %d\n"), age);
makeButton(catgets(catd, 1, 4, "Quit"));
```

XPG4 specifies a format for source message catalogs. For this program, here is a possible English source message catalog:

```
$ This line starts with "$ ", so it is a comment
$ We will use " as a delimiter for strings
$quote "
$ Notice that message numbers need not be in a contiguous range
$set 1
1 "Hello"
4 "Quit"
$ Notice that set numbers need not be in a contiguous range
$set 3
4 "Age: %d\n"
```

After each `$set` line, list the messages in that set in increasing order of message number. The set groups themselves must also be in ascending order of set number. The header `<limits.h>` defines `NL_SETMAX`, the maximum set number allowed; `NL_MSGMAX`, the maximum message number; and `NL_TEXTMAX`, the maximum number of bytes in a message text. The `genmsg(1)` manual page specifies the syntax of a source message catalog.

Notice that the English message texts in the source catalog are the same as the default strings in the `catgets()` calls in the program. This is almost always the case, of course: if the English message catalog could not be located, then the default messages would be the same as if the catalog had been successfully opened.

Whoever will be translating the messages in your catalog will probably not know the context in which those messages will appear. Usually, the translators will not be programmers, although you can expect that they will have some training in recognizing some common characteristics of message strings. For example, you can assume that in the following, the translators know that `%s` represents some string:

```
1 "%s cannot be opened."
```

However, you cannot assume the translator will know that the `%s` above will be replaced by a file name. In some languages, this may be significant, since the word for “opened” may be translated differently, depending on whether the element that

can't be opened is a file, a window, or a network connection. To enable good translations, you should include comments in your message catalogs for any strings that might cause difficulty:

```
1 "%s cannot be opened."  
$ %s is a file name  
  
2 "Read"  
$ This is a past participle, not a present tense verb
```

The `genmsg(1)` utility for creating source message catalogs became available in Solaris 2.6. This utility examines a source program file for calls to `catgets()` and builds a source message catalog from the information it finds. Here is an example:

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

Running `genmsg` on the program source file named `example.c` produced a source message catalog named `example.c.msg`. By specifying the `-c` option with an argument of our choosing (we chose the string `NOTE`), we caused `genmsg` to include comments in the catalog. If a comment in the source program contains the string we specified, that comment will appear in the message catalog after the next string extracted from a call to `catgets()`.

You can use `genmsg` to automatically number the messages within a message set. Refer to the `genmsg(1)` manual page for more information.

Translate the Source Message Catalog

For each language your application will support, you must have strings in the source message catalog translated to that language. For test purposes, you could change the message texts to a made-up language. Here's an example:

```
$quote "  
$set 1  
1 "XxxHelloyyY"  
4 "XxxQuityyY"  
$set 3  
4 "XxxAge: %dyyY\n"
```

These “translations” are readable by a tester who knows only English. The translated strings are longer than the English strings to simulate translation to a language where strings may be of a different length than in English. This lets you test to be sure that tables align, that button labels won't exceed the size of the button, and so forth. Another test file could be English with all the vowels deleted, to see if layouts are affected by shorter strings.

The `genmsg(1)` utility has options that cause it to automatically transform message strings as it produces a message catalog.

Generate the Binary Message Catalogs

For each translated source catalog, generate a binary message catalog. The binary catalog is the one your application will consult at runtime. Use the XPG4 `gencat` utility to generate the binary catalog. If your Korean source message catalog is named `demo.ko.msg`, you would say:

```
% gencat demo.ko.cat demo.ko.msg
```

The second argument is the source catalog, and the first is the binary catalog that will be created. Having successfully produced the binary catalog, you can install it in its final destination (`/usr/lib/locale/ko/LC_MESSAGES/demo.cat`).

While testing your application, you may not want to install the catalog in its production location; indeed, you may not have the permissions to do so. You can leave the binary catalog wherever you like, since you can set your `NLSPATH` so that

your application can find the catalog. Someone who knows only English and wants to test the `demo` application in Italian might first “translate” the English source message catalog as in the previous section, and then do the following:

```
% gencat demo.it.cat demo.it.msg
% env LANG=it NLSPATH=/tmp/%N.%L.cat demo
```

Italian locale rules will be used for date formats, collation, and so forth. However, the messages will still be readable by the Italian-illiterate tester, since they will be in English surrounded with “Xxx” and “yyY,” rather than in Italian.

If your application does not seem to be correctly finding the translated messages, as evidenced by your seeing the default strings or the wrong translated strings, consider the following common oversights:

- Did you establish the locale *before* you called `catopen()`?
- Are your `NLSPATH` environment variable and the arguments to `catopen()` correct? (For example, if the first argument to `catopen()` is “demo.cat” and `NLSPATH` is `./locale/%L/LC_MESSAGES/%N.cat`, then `catopen()` will look for `demo.cat.cat`.)
- Are your `catgets()` calls referring to the right set and message numbers? If you added, deleted, or changed message numbers in your `catgets()` calls but failed to revise, regenerate, and reinstall your message catalog, the numbers may be out of sync.

Messaging Using `gettext()`

Where `catgets()` uses numbers to index message catalogs, `gettext()` uses strings; that is the main difference in their approaches to the messaging problem.

The steps for text handling using `gettext()` are similar to those for `catgets()`:

1. Change source code to `#include <libintl.h>`, then call `textdomain()` to open the message catalog and call `gettext()` to retrieve strings from the catalog. In releases of Solaris prior to 2.6, the object program must be linked with the `-lintl` flag.
2. Use the `xgettext(1)` utility to extract native language text strings from the `gettext()` calls and store them in a source message catalog.
3. Translate the strings in the source message catalog into a target language.
4. Transform the translated source message catalog into a binary message catalog, using the `msgfmt(1)` utility. Install the binary catalog.

Locating Message Catalogs

Use `textdomain()` to open a message catalog. The pathname of `gettext()` message catalogs must end with `locale/LC_MESSAGES/domain.mo`, where `locale` is the current locale—the value of `LC_MESSAGES` as established by `setlocale()`—and `domain` is the argument you pass to `textdomain()`.

Unless you call `bindtextdomain()` to change the domain, the complete path is `/usr/lib/locale/locale/LC_MESSAGES/domain.mo`. In fact, this is where Solaris system messages for libraries and utilities that use `gettext()` reside.

This program fragment opens a message catalog named `/usr/lib/locale/locale/LC_MESSAGES/demo.mo`:

```
#include <locale.h>
#include <libintl.h>
int main()
{
    setlocale(LC_ALL, "");
    textdomain("demo");
    ...
}
```

Many applications do not require root permission for installation and thus cannot place their messages in `/usr/lib/locale`. Moreover, most applications need messages in their own directory hierarchy to simplify export across a network. So, most applications should use the Solaris routine `bindtextdomain()` to associate a path name with a message domain. Here's a sample invocation:

```
char *path;
#ifdef TEST
    path = "/tmp";
#else
    path = getenv("APPLICATIONHOME");
#endif
bindtextdomain("demo", path);
textdomain("demo");
```

If you compile the program with `TEST` defined, then the catalog will be found in `/tmp/locale/LC_MESSAGES/demo.mo`; if `TEST` is undefined, the catalog will be found in `$APPLICATIONHOME/locale/LC_MESSAGES/demo.mo`.

Surround Strings with `gettext()`

Although it is not portable, `gettext()` is much easier to use than `catgets()`. All you really have to do is go through your programs, enclosing literal strings inside `gettext()` calls. Here is `demo.c`, a short example:

```
#include <stdio.h>
#include <locale.h>
#include <libintl.h>
int main() /* demo.c */
{
    (void) setlocale(LC_ALL, "");
    bindtextdomain("demo", "/tmp");
    textdomain("demo");
    printf(gettext("Hello\n"));
    printf(gettext("Goodbye\n"));
    return 0;
}
```

The first `gettext()` looks in the catalog `/tmp/locale/LC_MESSAGES/demo.mo` for the translated string corresponding to the English string `"Hello\n"`. It returns a pointer to the translated string if it finds it; otherwise, it returns the index string `"Hello\n"`. You compile the program with

```
% cc demo.c -o demo
% cc demo.c -o demo -lintl
```

In the above example, `demo.c -o demo` is for Solaris 2.6 or later and `demo.c -o demo -lintl` is for versions of Solaris prior to 2.6.

You can partition your messages among different domains. When you call `textdomain()`, you establish the domain used by all calls to `gettext()` until you next call `textdomain()`. If you want to change domain for just the next call of `gettext()`, use `dgettext()` instead. This would be appropriate for a library product, as it is the best way to ensure a known domain. (Library calling sequence cannot be guaranteed, since different domains may be mixed together at random.) The library developer chooses the domain name.

The following two examples retrieve the same strings but have different effects on the text domain. The first example does not change the current text domain. The second example changes the current text domain to `library_error_strings`, then retrieves the alternate language string of `wrongbutton`.

```
message = dgettext("library_error_strings", "wrongbutton");

                                or

textdomain("library_error_strings");
message = gettext("wrongbutton");
```

Create the Source Message Catalog

After writing an application, create a text domain by extracting `gettext()` strings and placing them in a file with the alternate language equivalent.

Once you have enclosed all user-visible strings inside `gettext()` wrappers, you can run the `xgettext` command on your C source files to create a message file. This produces a readable `.po` file (the portable object) for editing by translators. For test purposes, you can use `xgettext`'s `-m` option to simulate a translation by adding a prefix string to each message.

```
% xgettext -m TRNSLT: demo.c
% cat messages.po
domain "demo"
msgid "Hello\n"
msgstr "TRNSLT:Hello\n"
msgid "Goodbye\n"
msgstr "TRNSLT:Goodbye\n"
```

The domain "*domainname*" line states that all following target strings until another domain directive belong to the *domainname* domain. Each `msgid` line contains the index string passed to `gettext()` and is followed by a `msgstr` line containing the translated string. The manual page for `msgfmt(1)` specifies the syntax of the `.po` file.

If you anticipate translators having difficulty translating a message, comment it, using lines starting with `#`. An effective way to do this is to place comments for the translator into your application source code, then use the `-c` tag option of `xgettext(1)` to place these comments into the `.po` file.

Create the Binary Message Catalog

Run `msgfmt` on the `.po` source file to produce a binary `.mo` file (the message object), which should be installed under the `LC_MESSAGES` directory. Here's a sample interaction on `demo.c`:

```
% msgfmt demo.po
% su
Password:
# mv demo.mo /usr/lib/locale/test/LC_MESSAGES
```

Problem Areas

Don't Overdo Messaging

You should not blindly wrap every string literal in your program in a call to `catgets()` or `gettext()`. In general, you only need to message those strings that users see. Do not message strings containing system commands or file names, such as `"sort"` or `"/dev/tty"`. Be careful when messaging strings inside `sprintf()`, which is often used to build up path names or command lines. You probably don't need to message strings used only for debugging. Because integers and decimal numbers are not strings, they don't need messaging, either.

Be Aware of Programming Language Restrictions

Not every context allows you to replace a string literal with a call to a function. Converting the noninternationalized declaration

```
static char *greeting = "Hello";
```

to

```
static char *greeting = catgets(catd,1,1,"Hello");
```

produces an illegal C declaration. One way to fix it is:

```
static char *greeting;
int main()
{
    /* establish locale and open catalog, and then: */
    greeting = catgets(catd,1,1,"Hello");
}
```

If this were a C++ program instead of a C program, the declaration with initialization would be legal. However, you must control the order of initialization of static objects so that `greeting` is not initialized until after the locale has been established and the message catalog opened.

Prepare for Variations in Text Length and Height

If strings must be stored in an array, be sure to declare arrays large enough to hold any possible translation. Messages in German are often longer than in English; messages in Chinese may be shorter, even accounting for multibyte encoding. A good rule of thumb is that a string might double in length, although very short strings might be even longer in translation (for example, English “Edit” is German “Bearbeiten”). Use `strncpy()` to avoid overrunning an array:

```
strncpy(msg, catgets(catd,1,1,"Hello"), sizeof(msg));
```

Displayed characters in translated messages may be of different length and height than the original messages. East Asian language ideographs are usually taller and wider than Roman characters.

Window system resource files specify height and width of elements such as panel buttons. The AppBuilder and DevGuide tools employ these facilities. In some cases, it's best to use implicit object positioning, letting the window system decide where to place things. See Chapter 9 for more details.

Avoid Compound Messages

Creating easily translated messages is an art form that involves more than just inserting `catgets()` calls around strings. Remember that word order varies from language to language, so complex messages can be very difficult to translate properly. A common-sense guideline is to avoid compound messages with more than two `%s` parts whenever possible.

There are two approaches to messaging: *static* and *dynamic*. Static messaging involves looking up strings in a message catalog, with no reordering taking place. Dynamic messaging also involves looking up strings in a message catalog, but those strings are reordered and assembled at runtime. International standards provide an ordering extension to `printf()` for implementing dynamic messaging.

The advantage of static messaging is simplicity. Use it whenever possible. However, avoid splitting strings across two `printf()` statements, which makes messages difficult to translate. Remember that the ANSI/ISO C preprocessor will paste together two consecutive string literals into one long literal:

```
/* bad */
printf(catgets(catd,1,1,"This is a very, very, very, very
"));
printf(catgets(catd,1,2,"long string that I want to
display"));
/* good */
printf(catgets(catd,1,1,"This is a very, very, very, very "
"long string that I want to display"));
```

Translation problems can arise with compound messages, especially when more than one sentence could be produced at runtime. Here is some code that would be difficult to translate:

```
/* poor practice: multipart compound message */
printf("%s: Unable to %s %d data %s%s - %s",
func, (alloc_flg ? "allocate" : "free"),
count, (file_flg ? "file" : "structure"),
(count == 1 ? "" : "s"), perror("."));
```

Quite apart from being poor programming practice, this fragment of code would be much clearer to the reader and much easier to translate if it were split into separate print statements inside an `if-else` block that would select the correct message at runtime:

```
if (alloc_flg)
    if (file_flg)
        printf("Unable to allocate %d file\n", count);
    else
        printf("Unable to allocate %d structure\n", count);
else
    if (file_flg)
        printf("Unable to free %d file\n", count);
    else
        printf("Unable to free %d structure\n", count);
```

The issue of making the objects plural is not addressed in this example because, in many languages, pluralization involves more than adding “s” to the end of a word.

Use Dynamic Messaging With Care

Dynamic messaging is used when the exact content or order of a message is not known until runtime. Unless done carefully, dynamic messaging causes translation problems. If the positional dependence of keywords is hardcoded into a program, code needs to be changed before messages can be successfully translated. Obviously, this defeats the purpose of internationalization.

XPG4 defines an extension to the `printf()` family that permits changing the order of parameter insertion. Solaris also supports this extension. For example, the conversion format `%1$s` inserts parameter one as a string, and `%2$s` inserts parameter two. The entire format string is parameter zero.

Here’s a small example of how these extensions can be used. This `printf` statement has position-dependent keywords because the verb must come before the object.

```
/* poor practice: position-dependent keywords */
printf("Unable to %s the %s.\n",
(lock_flg ? "lock" : "find"),
(type_flg ? "page" : "record"));
```

This could produce any of four messages in English:

```
Unable to lock the page.  
Unable to find the page.  
Unable to lock the record.  
Unable to find the record.
```

Here are those four messages translated into German. Note that the verb (“sperren” or “finden”) must follow, not precede, the object (“Seite” or “Rekord”).

```
Das Programm kann die Seite nicht sperren.  
Das Programm kann die Seite nicht finden.  
Das Programm kann den Rekord nicht sperren.  
Das Programm kann den Rekord nicht finden.
```

German syntax requires different word order, so the program’s keywords must be reversed. Here is that `printf` statement written for dynamic messaging:

```
printf(catgets(catd,1,1,"Unable to %s the %s\n"),  
       (lock_flg ? catgets(catd,1,2,"lock") :  
        catgets(catd,1,3,"find")),  
       (type_flg ? catgets(catd,1,4,"page") :  
        catgets(catd,1,5,"record")));
```

The German message catalog would then appear as follows:

```
1 "Das Programm kann %2$s nicht %1$s.\n"  
2 "sperren"  
3 "finden"  
4 "die Seite"  
5 "den Rekord"
```

This example might not work on other vendors’ systems because of multiple `catgets()` calls within one expression.

Consider carefully the effects of dynamic messaging. You might have to reposition parameters during translation. Often this fact isn’t recognized until translation actually begins, by which time it’s already too late—the software would have to be laboriously rereleased.

Manage Message Indices

When you use the `catgets()` messaging scheme, you must ensure that you don't assign the same set number/message number combination to different messages. This can be a problem in a multiperson project. Here are some guidelines for managing the message numbers.

- Use a different message set number for each subsystem or for each developer. This localizes potential conflicts, making them easier to find and fix.
- Do not change a message number after it has been assigned to a message. If a message is deleted, do not reuse its number. This makes successive versions of a message catalog more consistent. Suppose that a localizer has already translated a source message catalog. If a new version of that catalog arrives for translation, much less work needs to be done if unchanged messages can be quickly identified.
- Use a tool to assign message numbers. An automated process is less likely to assign duplicate numbers than a manual one. The `genmsg(1)` tool that became available in Solaris 2.6 has an option that automatically numbers those messages in each set that have not already been assigned numbers.
- Appoint a central numbering authority. Making one entity responsible for managing message numbers helps ensure that consistent procedures are followed.

Other Programming Languages

The Desktop Korn Shell, `dtksh`, in CDE has built-in `catopen`, `catgets`, and `catclose` commands. Here is an example:

```
catopen CATD demo
catgets msg1 $CATD 3 7 'Hello there'
catgets - $CATD 3 7 'Hello there'
catclose $CATD
```

Using the `LANG` and `NLSPATH` environment variables and the name `demo` (which will be substituted for `%N` in `NLSPATH`), `catopen` opens the message catalog and sets `CATD` to the catalog ID. The calls to `catgets` look for message 7 of set 3, returning `Hello there` if it can't find it. The message is stored in the variable `msg1` in the first call and written to standard output in the second. The `catclose` command releases the resources acquired by `catopen`.

Solaris provides a `gettext(1)` command to retrieve translated messages from a catalog for use in shell programming. This command reads the `TEXTDOMAIN` environment variable for the domain name and the `TEXTDOMAINDIR` environment variable for the path name to the message database.

Summary

To internationalize and localize text handling in an application, follow these steps:

1. Decide whether you will use the standard `catgets()` scheme or the nonstandard `gettext()` scheme.
2. Open the message catalog after establishing the locale.
3. Call `catgets()` or `gettext()` to retrieve strings from the catalog.
4. Extract native language text strings to form the source message catalog. Comment those strings that may cause translation difficulty.
5. Translate the strings in the source message catalog into a target language.
6. Transform each translated source message catalog into a binary message catalog.
7. Install the binary message catalogs when you install the application.

Note – The material in this section is used with permission from *Creating Worldwide Software: Solaris International Developer's Guide*, 2nd edition by Bill Tuthill and David A. Smallberg, published by Sun Microsystems Press/Prentice Hall. 1997.

X/DPS

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

Internationalization and localization issues using Adobe's PostScript™ are documented in several books from Adobe:

- *PostScript Language Reference Manual, Second Edition*. Adobe Systems Inc., Addison Wesley, 1990.
- *PostScript Language Reference Manual Supplement*. Adobe Systems Inc., December 1994.
- *Programming the Display PostScript System with X*. Adobe Systems Inc., Addison Wesley, 1993.

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

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

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

Localization Resource Category

The localization resource category specifies which natural language (for example, English or Japanese) is supported. This category is made up of dictionaries that contain the keys `Language`, `Country`, `CharSet`, and others. These keys are in the `%Console%` device parameter set.

```
<</Language/EN /Country/US /CharSet/ISO-646-ISV>>  
<</Language/JA /Country null /CharSet/JIS-...>>
```

In the example with Japanese, the `null` value shows that no dialect was selected for Japanese.

Unique names should be used for each item in the localization resource category.

Information on Language Interpreters

Page Description Language (PDL) interpreters can be assigned to a PostScript product. An application or printer driver uses the `PDL` resource category to see which PDL interpreter has been assigned.

Control languages can also be assigned. An application or printer driver can use `ControlLanguage` to see which control languages are available on a PostScript product.

The `PDL` and `ControlLanguage` resource categories have been available since version 2015.

The `PDL` and `ControlLanguage` resource categories are made up of key/value pairs. See the Adobe PostScript documentation for more information.

Desktop Environments

The Common Desktop Environment (CDE) is the standard GUI desktop interface for Solaris 2.6. Not only is it the user's main interface to the system, it is also the interface in which many of the user's locale settings are apparent. The German user sees a German interface; the French user sees a French interface.

The *Common Desktop Environment: Internationalization Programmer's Guide* provides information for internationalizing the desktop to enable applications to support various languages and cultural conventions in a consistent user interface.

Overview

The following is a synopsis of the *Common Desktop Environment: Internationalization Programmer's Guide*.

- **Chapter 1, "Introduction to Localization,"** contains an overview of internationalization and localization within CDE and Solaris, including locales, fonts, drawing, inputting (including preedit area, offthespot, overthespot, and root), interclient communications standards (ICCC), and extracting visual text. A discussion of internationalization standards is also included.
- **Chapter 2, "Internationalization and the Common Desktop Environment,"** explains topics which developers need to consider when internationalizing their applications. This includes locale management, localized resources, font management, drawing localized text, inputting localized text, extracting localized text, message guidelines, message extraction functions, localized resources, and operating system internationalized functions.
- **Chapter 3, "Internationalization and Distributed Networks,"** discusses the handling of encoded characters across distributed networks. Basic principles and examples for interclient interoperability are provided. Since the user will work not only in multiple languages, but also perhaps across various borders, this chapter discusses the principles of interclient interoperability in international

environments. The chapter discusses interchange concept, simple text basic interchange, mail basic interchange, encodings and codesets, and ISO EUC codesets.

- **Chapter 4, “Motif Dependencies,”** covers internationalized applications, locale management, font management, drawing localized text, inputting localized text, the internationalized User Interface Language (UIL), and localized applications.
- **Chapter 5, “Xt and Xlib Dependencies,”** discusses locale management, font management, font set matrix, drawing localized text, inputting localized text, interclient communications conventions for localized text, messages, charset and font set encoding, and registry information.
- **Appendix A, “Message Guidelines”** contains tips and suggestions for writing messages.

CDE is fully internationalized so that any application can run using any locale that has been installed in the system. By keeping the language- and culture-dependent information separate from the application source code, the application does not need to be rewritten or recompiled to be marketed in different countries. Instead, the external information only has to be localized to match the target language and customs.

The application interface has been standardized to allow functionality in any locale, including East Asia. Solaris 2.6 complies with the Portable Operating Systems Interface for Computer Environments (POSIX and X/Open specifications, which are also referred to as XPG4).

It is important that each layer within the desktop use the proper internationalization interface standards which are described in the following sources:

- *X Window System, The Complete Reference to Xlib, Xprotocol, ICCM, XLFD-X Version, Release 5*, Digital Press, 1992.
- *IEEE Std. 1003.1-1990. Information Technology-Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API)*. ISO/IEC 9945-1:1990.
- *OSF™ Motif 1.2 Programmer' Reference, Revision 1.2*, Open Software Foundation, Prentice Hall, 1992.
- *X/Open CAE Specification Commands and Utilities*, Issue 4, X/Open Company Ltd., 1992.

Common Desktop Environment: Programmer's Guide, Addison Wesley, 1995. The Solaris 2.6 updated version is supplied online with the CDE AnswerBooks. See “Related Books” on page xvi for more information.

Locales

Most single-display clients operate in a single locale. This is set with the environment variable, usually `$LANG` or a set of `LC_` environment variables including `$LC_CTYPE`.

The `LC_CTYPE` category of the locale is used by the environment to identify the locale-specific features used at runtime. The fonts and input methods are determined by the `LC_CTYPE` category.

Programs that are enabled for internationalization are expected to call the `XtSetLanguageProc()` function (which calls `setlocale()` by default) to set the locale.

Integrating Fonts

Your application may be used by someone sitting at an X terminal or by someone at a remote workstation across a network. In these situations, the fonts available to the user's X display from the X window server might be different than your application's defaults, and some fonts may not be available.

The standard interface font names defined by CDE are guaranteed to be available on all CDE-compliant systems. These names do not specify actual fonts. Instead, they are aliases that each system vendor maps to its best available fonts. If you use only these font names in your application, you can be sure of getting the closest matching font on any CDE-compliant system.

See *Solaris Common Desktop Environment: Programmer's Guide*, Chapter 2 "Integrating Fonts," and also the CDE man pages `DtStdInterfaceFontNames(5)` and `DtStdAppFontNames(5)` for additional information.

Input Methods

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

The preedit area displays the string that is being preedited. This can be done in four modes: OffTheSpot, OverTheSpot (default), Root, and None. In OffTheSpot mode, the location is just below the MainWindow area at the right of the status area. In OverTheSpot mode, the preedit area is at the cursor point. In Root mode, the preedit and status areas are separate from the client's window.

Internationalization and CDE

Multiple environments may exist within a common open system to support various languages. Each of these is called a *locale*. A locale specifies the language, fonts, and customs to display data. CDE is fully internationalized so that any application can run in any locale. Any application should be code-set-independent and include support for any multibyte codeset.

All components are shipped as a single, worldwide executable. These support the USA, Europe (Western and Eastern), Japan, Korea, Taiwan, and China.

Matching Fonts to Character Sets

Various sets of fonts are used to render a locale's characters, various sets of fonts are used. The specific font charset depends on the locale. This information should be in a locale specific `app-defaults` file. It will contain *font sets*, *fonts*, and *font lists*.

`XmFontSet` specifies the locale-dependent fonts. The resource name is `*fontSet`. Fonts should not be specified specifically. The resource name for `XFontStruc` is `*font`. Font lists contain lists of fonts and font sets. `XFontList` specifies the fonts.

Storage of Localized Text

Text strings in each language should be stored outside of the application and in directories which are identified by locale names. These strings are stored in three types of files: resource files, message catalogs, and private files.

Resource files and message catalogs are both files that deliver text strings. Resource files are compiled when they are loaded and message catalogs are precompiled and ready to be accessed. Any application should be code-set-independent and include support for any multibyte codeset. Private files may be databases of information that may include some text strings. Ideally, text strings should be in resource files or message catalogs. If text strings are supplied in a private file, then a tool should also be developed to extract and replace text strings.

Xlib Dependencies

X locale supports one or more of the locales defined by the host environment. Direct Xlib™ conforms to the American National Standards Institute (ANSI) C library and the locale announcement method is the `setlocale()` function. This function configures the locale operation of both the host C library and Xlib. The operation of Xlib is governed by the `LC_CTYPE` category; this is called the current locale. The `XSupportsLocale()` function is used to determine whether the current locale is supported by X.

Message Guidelines

Message guidelines should be developed and used to create a consistent format and style for text. Use clear and simple English so that all users, including those whose command of English is minimal, can understand every message. The book *Common Desktop Environment: Internationalization Programmer's Guide* ends with a number of guidelines for producing clear, concise, translatable messages. Messages should explain the problem and suggest how to perform the action successfully. Comments to the translators should also be included, that explain concepts, variables, and so forth. The book includes several lists of suggestions for the format style of the message catalogs and the style of the messages themselves.

Before sending out the message catalogs to be translated, it is useful to have the message catalogs translated from English into international English, that is, into a simplified English that can be easily translated into other languages. This speeds up the translation process, reduces the translator queries, and saves costs.

Internationalization and Distributed Networks

This section of the book covers the exchange of information between applications on different hosts. The transfer of data has to consider several parameters:

- The sender's and receiver's codeset
- Whether the protocol is 7-bit or 8-bit
- The type of interchange encoding allowed by the protocol

If the remote host uses the same codeset as the local host, and if the protocol allows 8-bit data, no conversion is needed. If the protocol allows only 7-bit data, the 8-bit code points must be mapped onto 7-bit ASCII values. There are various strategies for conversion.

If the remote host's codeset is different from that of the local host, the following two cases may apply. The conversion depends on the specific protocol. If the protocol allows 8-bit data, the protocol will need to specify which side does the conversion. If the protocol allows only 7-bit data, a 7-bit interchange encoding is needed along with an identifying character repertoire.

Mail Interchange

With the rise of the Internet and the ease of communicating with people around the world, an email message can be viewed on many platforms and dozens of locales. Standards for email interchange, however, are restricted by desktop machines for which the default email standard is Simple Mail Transfer Protocol (SMTP), which supports only 7-bit transmission channels.

The sending agent converts the body of the message into a standard format and labels it as body. The receiving agent looks at the body and if it supports the character encoding, it converts the body into the local character set.

Due to the fact that `dtmail` now uses the Language Conversion Library (LCL), `dtmail` now has the capacity to support multibyte characters in both the subject line, the mail body, and in attachments. There is also the ability for `dtmail` to have characters of different encodings within the same mail, for example, SJIS and EUC encodings for the Japanese (`ja`) locale.

OpenWindows

Solaris 2.6 does not have any changes in OpenWindows with regard to internationalization. Applications that were developed for previous versions of Solaris will run in Solaris 2.6 without any changes.

The XView toolkit is not codeset independent. Applications that use the XView toolkit are not supported in non-EUC locales, such as `ja_JP.PCK`, `en_US.UTF-8`, or `ko.UTF-8`.

For information on international XView, see the internationalization portions of the *XView Developer's Notes*.

For information on international OLIT, see the internationalization chapter of the *OLIT Reference Manual*.

Printing

Localization Printing Support Under Solaris 2.6

Solaris provides support for PostScript printers. Custom print filters are available to convert localized text to PostScript. See `mp(1)` and `postprint(1)` man pages for further details. The ability to download fonts onto a printer is also present.

For more details see the `download(1)` man pages. This support is configured for PostScript printers.

No internationalization-specific changes were made to printing with Solaris 2.6. Look for printing information in the AnswerBook; the *System Administration Guide* has several chapters that discuss printing.

European Printing Support

For European non-iso-8859-1 locales, such as Greek and Russian, `prolog.ps` files are supplied. The files are located in `/usr/openwin/lib/locale/print`.

When you print in one of these locales, the files are automatically downloaded to the printer. These fonts are PostScript Type1. They include Times, Helvetica, and Courier.

These are in normal, bold, italic, and bold-italic styles.

This allows printing on PostScript printers from both CDE and OpenWindows desktops. From a command line, use `/usr/openwin/bin/mp <filename> | lp` in each non-iso8859-1 locale.

For the Eastern European locales such as Russian, non iso-8859-1 encoded, `prolog.ps` files are supplied. The files are located in:

```
/usr/openwin/lib/locale/locale/directories/print/prolog.ps
```

for each relevant locale. At directories, insert of the following

```
/iso8859-10/  
/iso8859-2/  
/iso8859-4/  
/iso8859-5/  
/iso8859-7/  
/iso8859-9/
```

The files are downloaded automatically when you print in one of the Eastern European locales. A minimum set of fonts allow printing.

The fonts in the `prolog.ps` files are:

TABLE 10-1 `prolog.ps` Fonts

<code>/LC_Courier</code>	<code>CourierCyr AliasFont</code>
<code>/LC_Courier-Italic</code>	<code>CourierCyr Inclined AliasFont</code>
<code>/LC_Courier-Bold</code>	<code>CourierCyr Bold AliasFont</code>
<code>/LC_Courier-BoldOblique</code>	<code>CourierCyr BoldInclined AliasFont</code>
<code>/LC_Times-Roman</code>	<code>TimesNewRomanCyr</code>
<code>/LC_Times-Italic</code>	<code>TimesNewRomanCyr-Inclined Aliasfont</code>
<code>/LC_Times-Bold</code>	<code>TimesNewRomanCyr-Bold AliasFont</code>
<code>/LC_Times-BoldOblique</code>	<code>TimesNewRomanCyr-BoldIncl AliasFont</code>
<code>/LC_Helvetica</code>	<code>LucidaSansCyr AliasFont</code>
<code>/LC_Helvetica-Italic</code>	<code>LucidaSansCyr ItalicFont</code>
<code>/LC_Helvetica-Bold</code>	<code>LucidaSansCyr-Bold AliasFont</code>
<code>/LC_Helvetica-BoldOblique</code>	<code>LucidaSansCyr-BoldItalic AliasFont</code>

Asian Printing Support

The `xetops` and `xutops` utilities convert Asian text into a bitmapped graphics printed image. This allows you to print Asian characters on a PostScript-based printer.

A typical command line for printing such a file would be as follows:

```
system% pr <filename> | xetops |lp
```

or

```
system% pr <filename> | xutops |lp (for the ko.UTF-8 locale)
```

Japanese Solaris 2.6 supports the following Japanese-specific printers:

- Japanese PostScript printer
- Epson VP-5085 (based on ESC/P)
- NEC PC-PR201 (based on 201PL)
- Canon LASERSHOT (based on LIPS)

Japanese texts can be printed with these printers through the LP print service. TABLE 10-2 shows the relation between these printers and used components. See JFP User's Guide for further details.

TABLE 10-2 Japanese Printer Support

Printer	terminfo(-T)	interface(-i)	content(-I)	filter
Japanese PS	PS	jstandard	postscript	jpostprint
Epson VP-5085	epson-vp5085	jstandard	None	jprconv
NEC PC-PR201	nec-pr201	jstandard	None	jprconv
Canon LASERSHOT	canon-ls-a408	jstandard	None	jprconv

Use the following to set up a Japanese PostScript printer.

In the following example, the PostScript printer name is `lw`. The `/dev/lp1` is the device that is associated with the printer. For more information, see the `lpadmin` man page.

```
# lpadmin -p lw -v /dev/lp1 -T PS -I postscript
# lpadmin -p lw -i /usr/lib/lp/model/jstandard
# cd /etc/lp/fd
# lpfilter -x -f postprint
# lpfilter -f jpostprint -F jpostprint.fd
# accept lw
# enable lw
# /etc/init.d/lp stop
# /etc/init.d/lp start
```

You will be able to print with the following operation:

```
% lp -d lw Japanese Text File
```

Note – These features are supported only on Japanese Solaris. Supported input codesets are Japanese EUC (Default) and PCK. The locale setting of Command line in `jpostprint.fd` (or `jprconv.fd`) should be changed to `ja_JP.PCK`, if you want PCK.

Index

SYMBOLS

.cshrc, 47
/bin/stty directory, 46
/sbin/sh command, 105
/usr/bin/ldd command, 105
/usr/ucb/stty directory, 47

NUMERICS

8-bit clean software, 116

A

adding packages, 65
addresses, formats, 13
Adobe Type Manager (ATM) fonts, 29
alphabets, 9, 10
APIs, 108 to 112
 using to develop applications, 104
applications
 developing, 104
 FontSet/XmFontList definitions, 62
 internationalizing, 62
 linking to system libraries, 104 to 105
 XPG4, 107
architectures (SPARC and x86), xv
Asian
 packages, 80
 printing support, 152
ATM fonts, 29

ATOK8, 34

B

base Solaris 2.6, 17 to 20
 locales supported, 20
Big-5, 33
 codeset, 101
/bin/stty directory, 46
binary message catalog, 131
bindtextdomain, 133
bitmap
 font editor, 38
 fonts, 30
books@sun.com, xvi
bopomofo in Chinese, 11
breve, 21

C

caron, 21
catalog
 binary messages, 131
 source messages, 128
catalog descriptor, 126
catclose(), 126
catgets(), 112, 125
catopen(), 125
CD
 installing software from, 67

- CDE, 145
 - en_US.UTF-8 locale support of, 20
 - input methods, 147
 - localization packages, 97
 - using fonts for locales, 21
- Central European languages, character support, 20
- character classification macros, 107
- character support, 20
- character transformation macros, 107
- characters
 - number, 9
- Chinese
 - package files, 86
- Chinese Solaris, simplified, 2
- Chinese Solaris, traditional, 2
- Chinese text
 - bopomofo, 11
 - Hanzi, 11
 - linguistic introduction, 11
 - pinyin, 11
 - zhuyin, 11
- CNS-11643, 33
- code conversion STREAMS modules, 43
- code conversions, 47 to 49
- codeset
 - Big-5, 101
 - character support, 20
 - Extended UNIX Code (EUC), 101
 - Shift-JIS, 101
- Codeset Independence, 1, 102
- collation and formats, 119, 122
- command examples, xviii
- command names, xviii
- command-line placeholder, xviii
- commands
 - CSI-capable, 102
 - Japanese, 37
- Common Desktop Environment* (book), xvi
- Common Desktop Environment Internationalization Programmer's Guide*, 145
- Compose c c sequence, 58
- Compose g g sequence, 59
- compose sequences
 - Latin-1, 50 to 54
 - Latin-2, 54 to 56
 - Latin-4, 56 to 57
 - Latin-5, 58
- compose sequences, for new locales, 21
- compound messages, 137
- conversion
 - multibyte and wide character process code, 107
- conversions, 47 to 49
- converting characters, 24
- core locales, 18
- creating
 - message catalogs, 112
- Creating Worldwide Software*, xvi, 14
- cs00, 34
- CSI, *See* Codeset Independence
- CSI-capable commands, 102
- CSI-enabled libraries, 103
- ctype
 - library routines, 119
 - macros, 107
- currency
 - formats of, 121
 - presentation order of, 8
 - sizes of, 9
 - symbols of, 8
 - units of, 8
- currency symbol, 22
- Cyrillic input mode, 58
- Czech
 - character support, 20
 - keyboards, 21

D

- date and time formats, 120
- date formats, 6
- Daylight Savings Time (DST), 6
- decimal places, 7
- degree symbol, 22
- delimiters
 - numeric, 8
 - thousands, 7
 - word, 9
- descriptions of European package files, 74
- desktop environments, 145
- desktop layers, 146

- deutsche mark, 8
- developer's cluster, in Solaris 2.6, 20, 41
- developing international applications, 104
- dgettext, 134
- diacritical marks, 21
 - in English input mode, 50
- diaeresis, 21
- directories, xviii
- disk space
 - Asian packages, 100
- documentation, ordering, xvii
- dollar, 8
- doubleacute, 22
- DST (Daylight Savings Time), 6
- dtlogin command, 21
- dtmail, 150
- dtterm, 44
- dynamic linking, 104 to 105
- dynamic messaging, 139

E

- Eastern European package files, 73
- en_US.UTF-8, 20
 - code conversions, 47
 - fontset definitions, 62 to 63
 - overview, 17, 41 to 60
 - printing utility, 61 to 62
- en_US.UTF-8 locale, 17
- English
 - character support, 20
 - input mode, 50
 - language locales, 24
 - Solaris, 2
- English Solaris 2.6, *See* base Solaris 2.6
- environment
 - LANG, 116
 - LC_COLLATE, 122
 - LC_MONETARY, 121
 - LC_TIME, 120
- EUC, *See* Extended Unix Code
- European Codesets, 79
- European font packages, 79
- European printing support, 151

- European Solaris, 2
- extended locales, 19
- Extended UNIX Code (EUC), 101

F

- file code, 102
- file names, xviii
- font editor
 - bitmap, 38
 - Type1, 38
 - Type3, 38
- fonts
 - across different platforms, 147
 - adding or removing, 30
 - formats, 29
 - location, 30
 - packages for Europe, 79
 - SUNixxf format for new locales, 21
 - X11 bitmaps, 61
- FontSet definitions, 62 to 63
- FontSet/XmFontList definitions, 62
- formats
 - addresses, 13
 - and collation, 122
 - currency, 8, 121
 - dates, 6, 120
 - monetary, 121
 - numeric, 7
 - set with `setlocale()`, 119
 - sort orders, 122
 - string collation, 122
 - time, 5, 120
- franc, 8
- French package files, 69

G

- GB2312-80, 32
- gender in language, 13
- genmsg utility, 112 to 113, 130, 131
- German
 - character support, 20
 - package files, 70
- gettext(), 132

- insertion, 134
- surround strings, 134
- GMT offset, 6
- Greek
 - character support, 20
 - input mode, 59 to 60
 - keyboards, 22
- Greenwich Mean Time offset, 6

H

- Hangul in Korean, 10
- Hanja in Korean, 10
- Hanzi in Chinese, 11
- head side module, 43
- Hiragana in Japanese, 10
- Hungarian
 - character support, 20
 - keyboards, 21

I

- IBM DOS 437, 11
- `iconv`, 28
 - command, 47
 - how to use, 24
 - Japanese character code conversion, 36
- imperial system, 13
- input modes
 - Cyrillic, 58
 - English, 50
 - Greek, 59 to 60
- installation, 65 to 68
- internationalization, 2
 - ISO Latin-1, 3
 - Java, 102
- internationalization APIs, 108 to 112
- internationalizing applications, 62
- introduction, xv
- ISO 8859, 41
- ISO 8859-*n* character support, 20
- ISO Latin-1, 3
- ISO/IEC 10646-1, 41
- ISO-10646, 1

- Italian package files, 71

J

- `ja`, 34
- `ja_JP.PCK`, 34
- Japanese
 - package files, 87
 - Solaris, 2
- Japanese text
 - Hiragana, 10
 - Kanji, 10
 - Katakana, 10
 - linguistic introduction, 10
- Japanese-specific commands, 37
- Japanese-specific printer support, 38
- Java internationalization, 102
- JLE Binary, 38
- Jumpstart, 24

K

- Kanji in Japanese, 10
- Katakana in Japanese, 10
- key compose sequences, 21
- keyboard layouts
 - Greek, 60
 - Russian, 59
- keyboards, 12
 - Changing keyboards on x86, 23
 - Changing on SPARC, 22
 - Czech, 21
 - Greek, 22, 60
 - Hungarian, 21
 - Latvian, 21
 - Lithuanian, 21
 - Polish, 21
 - Russian, 22
 - Support in Solaris 2.6, 22
 - Turkish, 21
- Korean package files, 85
- Korean Solaris, 2
- Korean text
 - Hangul, 10
 - Hanja, 10

- linguistic introduction, 10
- krona, 8
- krone, 8
- kroner, 8
- KSC-5700, 31

L

- LANG, 42
- LANG environment
 - default behavior, 116
- LANG environment variable, 42, 147
- Language Conversion Library, 150
- Latin-1 compose sequences, 50 to 54
- Latin-2 compose sequences, 54 to 56
- Latin-4 compose sequences, 56 to 57
- Latin-5 compose sequences, 58
- Latin-*n* terminals, 45
- Latvian keyboards, 21
- LC_COLLATE, 5
- LC_COLLATE environment, 122
- LC_CTYPE, 5
- LC_MESSAGES, 5
- LC_MONETARY, 5
- LC_MONETARY environment, 121
- LC_NUMERIC, 5
- LC_TIME, 5
- LC_TIME environment, 120
- LCL, 150
- libc, 104 to 105, 107
- libintl, 106
- libraries, linking applications to, 104 to 105
- library routines
 - ctype, 119
- libw, 106
- linking, 115
- linking applications, 104 to 105
- lira, 8
- list separators, 8
- Lithuanian keyboards, 21
- loading
 - STREAMS modules, 43 to 44
- locale utility, 42

- locale(1), 42
- locales, 2, 4, 20
 - categories of, 4
 - compose sequences, 21
 - core, 17, 18
 - database, 101, 104
 - en_US.UTF-8, 17, 20
 - English, 24
 - environment variables, 42, 147
 - extended, 17, 19
 - font format, 21
 - full, 3
 - localized text handling, 125
 - operating system, 17
 - partial, 3, 17
 - what is..., 3
 - window system, 17
- localization resource category, 144
- lpadmin command, 61
- lpfilter command, 61
- lpr command, 61

M

- macros
 - ctype, 107
- mail interchange, 150
- markka, 8
- mbtowcs, 107
- mbtwoc, 107
- message catalogs, creating, 112
- message files, creating, 136
- messages
 - compound, 137
 - creating message database, 135, 136
 - dynamic, 139
 - location of database, 125, 133
 - static, 138
 - text length and height variability, 137
 - window system resource files, 137
- metric system, 13
- modinfo command, 43
- modload command, 44
- monetary formats, 121
- mp(1), 151

multibyte file code, 107
multi-byte Unicode representation, 17, 41
mystreams file, 47

N

NULL (0x00), 102
number of characters, 9
numeric conventions, 7

O

ogonek, 21
OLIT Reference Manual, xvii
on-screen computer output, xviii
OpenWindows
 changes, 150
 font editor, 38
 using fonts for locales, 21
operating system locale, 17
order for sorting, 9
ordering documentation, xvii
OSF/Motif Programmer's Guide, xvii
OSF/Motif Programmer's Reference, xvii
outline fonts, 29

P

packages
 adding, 65
Page Description Language (PDL) interpreters, 144
page sizes, 13
paper sizes, 13
paper trim size, 14
partial locales, 17
PDL interpreters, 144
peseta, 8
pinyin in Chinese, 11
pkgadd command, 66
pkgchk command, 66
Polish
 character support, 20
 keyboards, 21

POSIX, 146
postprint(1), 151
PostScript, 29, 143
 output, 61
 support under Solaris, 151
 Type 1 fonts, 29
PostScript Language Reference Manual, xvii, 143
PostScript Language Reference Manual Supplement, xvii, 143
pound, 8
printing, 61 to 62
printing support
 Asian, 152
 European, 151
 Japanese, 38
process code format, 104
Programming the Display PostScript System with X, xvii, 143
prompts, *See* shell prompts
punctuation, 12

R

radix characters, 7
remote package server
 installing software from, 67 to 68
Russian
 character support, 20
 keyboard layout, 59
 keyboards, 22

S

saving
 STREAMS modules settings, 47
/sbin/sh command, 105
Scandinavian and Baltic language character support, 20
script selection, 50 to 60
scripts, in multiple languages, 20
separators
 list, 8
 thousands, 7
 word, 9

- setenv command, 42
- setlocale man page, 42
- setlocale(), 115
- setting
 - terminal options, 46
- setup
 - TTY environment, 42
- shell prompts, xviii
- Shift-JIS codeset, 101
- shortcuts. *See* compose sequences
- sign extension problems, 117
- Simple Mail Transfer Protocol, 150
- single-display clients, 147
- Slash (0x2f), 102
- Smallberg, David, xvi, 14
- SMTP, 150
- software developers, xv
- Solaris
 - Asian, 30
 - Austrian, 27
 - base product, 17 to 20
 - Chinese, 32
 - contents, 25
 - Czech, 27
 - Eastern European, 2
 - English, 2, 25
 - Estonian, 27
 - European, 25
 - French, 2, 25
 - German, 2, 25
 - Greek, 27
 - Hungarian, 27
 - Italian, 2, 25
 - Japanese, 2, 34
 - Japanese printing support, 153
 - Korean, 2, 31
 - Latvian, 27
 - Lithuanian, 27
 - localized products in, 2
 - Polish, 27
 - PostScript support, 151
 - Russian, 27
 - Simplified Chinese, 2
 - Spanish, 2, 25
 - Swedish, 2, 25
 - Turkish, 27
- sort order, 9
- source message catalog, 128
- Spanish
 - character support, 20
 - package files, 72
- SPARC architecture, xv
- SPARC keyboards, 22
- standalone system
 - adding packages to, 65 to 66
- standards
 - interface, 146
 - internationalization, 146
- stateless file code encodings, 102
- static linking, 105
- static messaging, 138
- strchg command, 44, 46
- strcmp(), 122
- strcoll(), 122
- strconf command, 46
- STREAMS modules
 - loading, 43 to 44
 - saving settings, 47
- string collation, 124
- strxfrm(), 122
- stty command, 47
- stub entry points, in libw and libintl, 106
- su command, 43
- SunDocs program, xvii
- SunOS 5.6, xv
- SUNWpldte, 21
- SUNWploc, 17
- SUNWploc1, 17, 21
- SUNWplow, 17
- SUNWplow1, 17, 21
- Swedish package files, 73
- symbols, 12
- system libraries
 - linking applications to, 104 to 105

T

- tail side module, 43
- terminal options, setting, 46
- terminal support for Latin-1, Latin-2, or KOI8-R, 45
- terminals

- Latin-*n*, 45
- Latin-*n* terminals, 45
- text
 - height, 137
 - length, 137
- textdomain()
 - environment variable, 141
 - opens message catalogs, 133
- thousands separators, 7
- time and date formats, 120
- time formats, 5
- time zones, 6
- titles, xviii
- titles in language, 13
- TrueType fonts, 1
- TTY environment setup, 42
- TTY STREAMS, 38
- Turkish
 - character support, 20
 - keyboards, 21
- Tuthill, Bill, xvi, 14
- Type 1 fonts, 29
- Type3 font editor, 38

U

- u8lat1 STREAMS module, 45
- u8lat2 STREAMS module, 45
- UDC support, 38
- Unicode 2.0, 1 to 17
 - support, 1
- UniForum standards, 124
- Universal Character Set Transformation Format for
 - 8 bits encoding, *See* UTF-8 encoding
- user type, xviii
- User-Defined Character support, 38
 - /usr/bin/ldd command, 105
 - /usr/ucb/stty directory, 47
- UTF-8 encoding, 20
- utilities
 - genmsg, 112 to 113
 - locale, 42
 - printing, 61 to 62

W

- wcstombs, 107
- wctomb, 107
- Western European alphabets, 10
- Western European languages, character
 - support, 20
- wide character
 - expression, 101
 - process code, 107
- window system locale, 17
- Wnn6, 34
- words
 - delimiters, 9
 - order of, 9, 11
- writing internationalized code, 115

X

- X Display PostScript, 143
- X Window System, 143
- X/DPS, 143
- X/Open-Uniform Joint Internationalization
 - Working Group, 20
- X11 bitmap fonts, 61
- x86
 - architecture, xv
 - keyboards, 23
- xetops, 152
- XFontStruc, 148
- Xlib dependencies, 149
- XmFontSet, 148
- XoJIG, 20, 41
- XPG4 applications, 107
- xutops, 152
- xutops utility, 61 to 62
- XView Developer's Notes*, xvii
- XView toolkit, 150

Y

- yen, 8

Z

zhuyin in Chinese, 11

