Complex Text Layout Language Support in the Solaris Operating Environment

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

Part Number 806-5583
May 2000

Contents

Preface

The *Complex Text Layout Language Support in the Solaris™ Operating Environment* white paper presents information and software features for internationalizing software in Complex Text Layout language markets.

Complex Text Layout (CTL) languages require text layout transformations on text input to properly display text output on the screen or printed page. CTL functionality is available in all Unicode locales as well as the Thai, Arabic, and Hebrew locales.

Who Should Use This Book

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

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

How This Book Is Organized

Chapter 1 describes characteristics of CTL languages and how they are managed by the Solaris CTL language engines.

Chapter 2 provides an overview of the Solaris CTL architecture and explains how developers can add CTL functionality to an application.

Chapter 3 provides helpful resource information.

Related Books

The following books are related to software internationalization:

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

The following books are related to CTL language support:

- *CAE Specification: Portable Layout Services* ISBN 1–85912–142–X.
- *Portable Layout Services: Context-dependent and Directional Text* ISBN 1–85912–075–X.

Ordering Sun Documents

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

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

Accessing Sun Documentation Online

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

CHAPTER **1**

Complex Text Layout Languages

This chapter introduces features of Complex Text Layout languages and their support in the Solaris operating environment.

- Complex Text Layout language support in Solaris
- Bidirectional scripts (Arabic and Hebrew)
- Character representation
- Input and text handling
- Numbers and dates

1.1 Introduction

Storing and displaying text has traditionally followed the basic structure of writing systems in Western languages (languages based on the Roman, Cyrillic, and Greek alphabets). Input characters are stored in the order in which they are typed, and displayed in the same order. There is no logical difference between how text is stored and how it is displayed.

Some writing systems, however, are different than those of Western languages. The direction in which characters are displayed can be different and the shape of a character or word can be modified depending on the adjoining text. Complex text layout transformations are required for display. A Complex Text Layout (CTL) language is any language which stores text differently from how it is displayed.

Particularly, many CTL languages use *bidirectional script*. Words and sentences are written from *right to left*, while some text, such as numbers and Roman-based words, are written from *left to right*. Arabic and Hebrew use bidirectional script. Furthermore, some CTL languages are *context dependent*. In Arabic, characters can be modified depending on the preceding and following characters. Written Thai uses character clusters, a collection of syllabic elements denoting consonants, vowels, and tonal values.

1.2 CTL Language Support in Solaris

The Solaris operating environment supports CTL easily and conveniently in all Unicode and Thai, Arabic, and Hebrew locales. CTL scripts are properly rendered as screen output or in printed format. Most applications need only be recompiled without changing the source code to run under these locales. A CTL-supported application enables users to input, view, and display CTL text without any additional development work.

The Solaris operating environment contains a standard API, the Portable Layout Services (PLS) from The Open Group, which provide applications with a consistent interface to the CTL language engines. Developers set various CTL attributes in the PLS configuration file, including cursor positioning and character and cell deletion.

Note - Unless otherwise stated, the CTL language engines manage complex text layout transparently to the user.

1.3 Bidirectional Scripts

Arabic and Hebrew use bidirectional script. In a bidirectional script, the default text direction is from right to left. However, numbers and Western-language text (such as quotations or proper names) are written from left to right. Text justification depends on line direction—typically, right-to-left text is right justified and left-to-right text is left justified.

Because input characters are stored in order, an application must manage the display of blocks of right-to-left and left-to-right text in the proper order. In Figure 1–1, text direction is left to right for the English text and right to left for the Arabic text. Note that the cursor is to the left of the Arabic text after input.

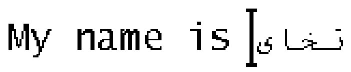


Figure 1–1 Bidirectional text

Note - In the Solaris operating environment, complex text layout issues are managed by *Motif*. Applications using Motif do not need to address these issues directly. For more information, see Section 2.1, *Architecture*.

1.3.1 Managing Bidirectional Text

In an application supporting bidirectional script, the *global direction* determines text flow and alignment. For example, in an Arabic word processor, the global direction is right to left for justification, tabulation, and related formatting operations. Roman words and phrases within an Arabic text string, however, remain left-to-right aligned. When a line break occurs in the middle of an Arabic string imbedded in an English sentence, however, the break must still respect the left-to-right text flow as shown in Figure 1–2.

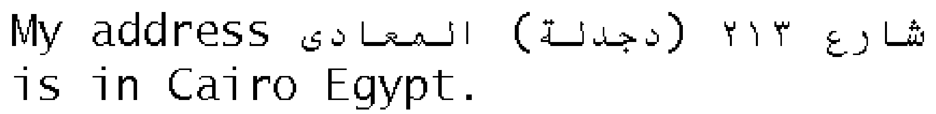

My address شارع ٢١٣ (دجلة) المعادى
is in Cairo Egypt.

Figure 1–2 English-Arabic-English left-to-right text flow

1.3.2 Symmetrical Swapping

Some characters have a meaning related to their direction, such as greater than (>) and less than (<). Bidirectional text strings containing these characters may need to replace one character with its complement to retain the correct meaning. For example, the expression 3 < 4 (three is less than four) is stored in the sequence in which it is typed. To display this expression correctly in a right-to-left string, however, it must appear as 4 > 3 (read right to left as 3 less than 4). Because a direct transposition of the three characters in storage would give 4 < 3, which is incorrect, the transformation process must swap < and >.

Replacing complementary symbols to retain their correct meaning is called *symmetrical swapping*. Characters requiring symmetrical swapping include parentheses: (and), angle brackets: < and >, square brackets: [and], and curly brackets: { and }.

1.4 Character Representation

In CTL languages, the basic input character is often modified before being displayed. *Character representation* is the final display form of the screen or printed character and depends on context, ligatures, diacritical marks, and character clusters.

Note that any modified input will not have a one-to-one correspondence between the number of input characters and the number of output glyphs. A single glyph occupying one display cell can represent two or more typed characters. Internal algorithms determine the character representation from the number and sequence of input characters. By using the PLS library, an application uses the appropriate character representation in each CTL language.

1.4.1 Contextual Analysis

The shape of a character in its final display form can depend on its position in a word or its position relative to neighboring characters. Changing the shape of a character is called shaping or *contextual analysis*. For example, in English handwriting, letters can have different shapes in isolation or connected to other letters: the letter “r” appears differently at the beginning, in the middle, or at the end of a word. In English printing, however, character representation is unaffected by position—a character in isolation or connected to other letters is identically represented. For example, lowercase “a” is represented as “a”—either by itself, at the start, in the middle, or at the end of a word as in: a, all, lap, formula.)

Arabic uses a cursive script, connecting one character to another as in English handwriting. All characters are affected in many different ways by their context. Arabic characters can have up to four final display forms: *initial*, *final*, *medial*, or *isolated*. During input, the keystrokes are stored as basic code–point values. The Arabic language engine reads the code-point values and selects the appropriate final display form from the context. Ligatures and diacritics, however, must also be considered.

Note - Only the basic shape of each character appears on an Arabic keyboard.

1.4.2 Ligatures

A ligature is the combination of two or more characters to create a single character or syllable. For example, in English, a diphthong is a ligature which unites two vowel characters to create another sound: a + e = æ.

In Arabic, ligatures are the combinations of two and, sometimes, three characters into one glyph. The resulting glyph replaces the characters composing it. For example, an Arabic letter typed twice is stored in memory as two distinct keystrokes, occupying two display cells. However, the Arabic language engine recognizes the context of each character and returns one glyph as shown in Figure 1–3.

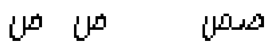


Figure 1–3 Arabic ligature

In the Arabic language engine, a ligature occupies the same number of display cells as input characters, with one exception: the combination character *lamalif*.

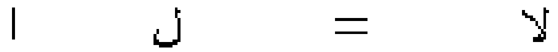


Figure 1–4 Lamalif combination character

The rules governing ligatures in Arabic text are very complex and do not depend solely on individual characters. Certain fonts define as many as 200 ligatures, while other fonts do not use ligatures at all.

1.4.3 Diacritics

Diacritical marks (or diacritics) are added to a character to show pronunciation. For example, in French, an accent is a diacritical mark above a vowel which alters the stress of the vowel: à á è é.

In Hebrew, diacritical marks represent vowel sounds and appear above, below, or inside characters . Typically, however, words are written without diacritics and vowel sounds are determined from the context.

In Arabic, diacritical marks can appear above (single or double diacritic) or below (single diacritic) any character. In the Arabic language engine, diacritics are entered as separate keystrokes and occupy the same cell as their associated character.

In Thai, diacritical marks appear above or below the base line containing consonants, vowels, symbols, and numbers. In the Thai language engine, diacritics are entered as separate keystrokes and occupy the same cell as their associated character.

1.4.4 Character Clusters

A *character cluster* is a collection of alphabetic characters forming a single word or syllable.

A Hebrew character cluster can contain characters and diacritics, though diacritics are not often used.

Character clusters are integral to written text in Thai. A Thai character cluster consists of up to four parallel lines:

- Base line of consonant, vowel, or symbol
- Lower line of lower vowel or diacritic
- Upper line of upper vowel or diacritic
- Line above the upper line for tone marks and upper diacritics

One or more elements may change shape in the presence of other elements. In the Thai language engine, a character cluster occupies one display cell. Depending on the composition, one display cell can contain up to three input characters, as shown in Figure 1–5.

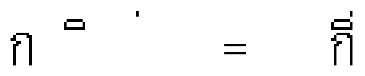


Figure 1–5 Thai-language character cluster

1.5 Input and Text Handling

The Solaris operating environment locales support text input in a number of different writing systems. In the UTF-8 locale, to choose an input mode, press the `Compose` key and a two-letter code as in Table 1–1. For example, to input text in Thai, press `Compose+tt`. Alternatively, click the status area and select an input mode. (To select the English/European input mode, press `Control+Space`.)

Note - In a CTL locale, to toggle between the English/European and CTL locale, press `Control+Space`.

The bottom left corner of the text window displays the current input mode, as shown in Figure 1–6.

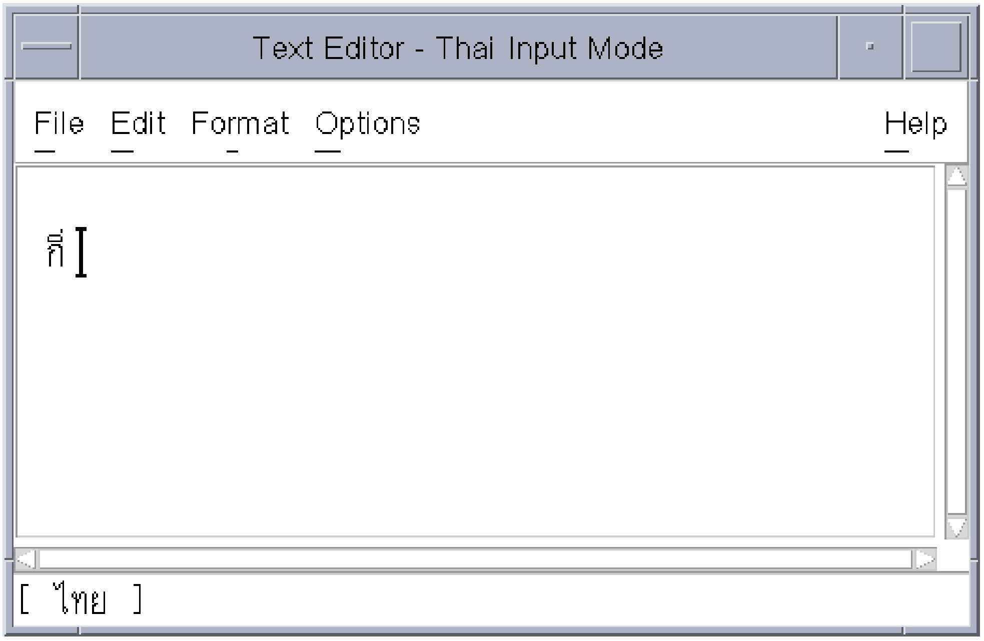


Figure 1–6 Current input mode (Thai)

The user can mix text between English and the CTL language locale (Arabic, Hebrew, Thai) by switching input modes. The formatting rules governing the input stream are managed by the application.

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

Language	**Code**
Cyrillic	`cc`
Greek	`gg`
Thai	`tt`
Arabic	`ar`
Hebrew	`hh`
Unicode Hex	`uh`
Unicode Octal	`uo`

TABLE 1–1 UTF-8 Input Mode two-letter codes *(continued)*

Language	Code
Lookup	`ll`
English / European	`Control+Space`

Note - To switch modes, press `Compose+code`. On a X86 machine, use `Ctrl+Shift+F1` as the `Compose` key.

CTL text can also be input with a lookup table as shown in Figure 1–7.

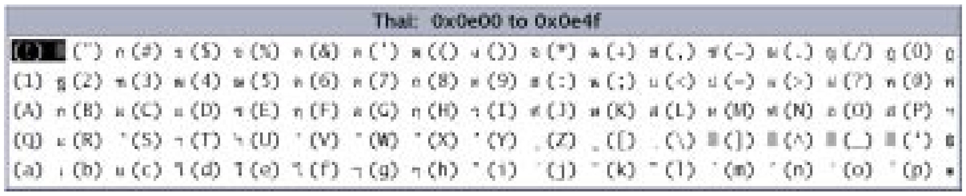


Figure 1–7 Lookup Thai input mode

To view a lookup table, click the status area, [Lookup], and a Lookup input mode. To input a character, click the character.

1.5.1 Cursor Positioning

The default cursor position is the upper left corner of the screen. The cursor direction depends on the input mode (left-to-right for Western languages and right-to-left for Arabic and Hebrew). The cursor position is determined after the second keystroke.

In English and other Western languages, the cursor redisplays to the right of each keystroke. In Hebrew and Arabic, the cursor remains stationary, with the character stream appearing to the right or left of the cursor, depending on the page alignment.

In Thai, the character cluster is composed to the left of the cursor. If a set of two or three keystrokes belong to the same cluster, the language engine values these keystrokes as zero and the cursor remains in place. The cursor moves to the right as each cluster is composed.

Note - Cursor attributes can be reconfigured in the application resource file. For more information, see Step 4 in Section 3.1 *CTL Motif Resources.*

1.5.2 Character and Cell Deletion

To manage editing character clusters, the CTL language supports both *character-based deletion* and *cell-based deletion*. In character-based deletion, the user deletes the separate elements of the character cluster with each press of the `Delete` key. In cell-based deletion, the user deletes the entire cluster with one keystroke.

The two deletion modes can be assigned to separate keys on the keyboard using translation and resources. For more information, see Step 3 in Section 3.1 *CTL Motif Resources*.

1.5.3 Processing Stored Text

In the Solaris operating environment, all CTL text is stored in *logical* order—the order of the character input stream. The CTL language engines manage the transformation from CTL scripts to final display form on screen.

Most legacy systems containing CTL text store data in *physical* order—the order represented on screen. This was done because there was no processing capability at the terminal for reading and correctly rendering logical input. Screens of text were stored and then read again to screen.

Applications can read legacy data using the CTL language engines by configuring the appropriate language engine modifier. For more information, see Step 1 in Section 3.1 *CTL Motif Resources*.

For other language-engine configuration parameters, see *CAE Specification: Portable Layout Services (ISBN 1-85912-142-X)*.

1.6 Numbers and Dates

For numbers, most countries use the "Arabic" or Western format of ten digits: 1 2 3 4 5 6 7 8 9 0. Some Arabic countries, however, use *Hindi* digits.

٩ ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١ ٠

Figure 1–8 Hindi digits

The Solaris operating environment supports both Arabic and Hindi digits. By default, Western language input modes display Arabic numbers while the Arabic input mode displays Hindi numbers, though this is configurable by the user. To change the default, see Step 1 in Section 3.1 *CTL Motif Resources*.

Arabic countries use both the Gregorian (Western) and Hijri (Islamic) date systems. These and other numeric data, such as time and monetary formats and currency symbol, are managed by the Solaris operating environment locale settings.

1.7 Summary

CTL languages require complex transformations to properly display or print characters. The fundamental characteristics of CTL languages are:

- Bidirectional text—Arabic and Hebrew are written right to left, but numbers and Roman-based characters are written left to right.
- Contextual analysis—each Arabic character has up to four display representations. The representation glyph depends on its position in the text.
- Ligatures—combinations of two or more Arabic characters can form a different, single character.
- Diacritics—diacritical marks placed above, below, or inside vowels and consonants modify their tonal value.
- Character clusters—syllables and cells, especially in Thai, are composed of several alphabetic elements, including vowels, consonants, diacritics, and tone marks.
- Number and date formats—some Arabic countries use Hindi digits rather than Arabic digits, and the Islamic calendar rather than the Western calendar.

Motif interfaces with PLS which interfaces with the CTL language engines. Applications that make use of Motif services need not address CTL issues directly because all CTL transformations and rendering are handled transparently.

CHAPTER 2

Technical Considerations

This chapter outlines the implementation of CTL support in the Solaris operating environment.

- Architecture
- Multiscript support
- Representation management
- CTL locales

2.1 Architecture

CTL language support in the Solaris operating environment comprises a set of extensions supporting transformations between text input and text output.

The CTL language engines enable the CDE/Motif drawing and measurement API for character shaping, such as ligatures and diacritics, the transformation of static and dynamic text widgets, and right-to-left and left-to-right text alignment and tabbing. Because text rendering is handled through the rendition layer, other widget libraries can be extended to support CTL.

The new CTL features require the Portable Layout Services (PLS) library and the appropriate language engine. CTL uses PLS as the interface to the language engines to transform text before rendering.

Note - 1. There are no changes to the Motif public APIs. A few resources have been added, but all old resources still work.

2. Existing applications can inherit CTL support by adding the resources without changing any source code.

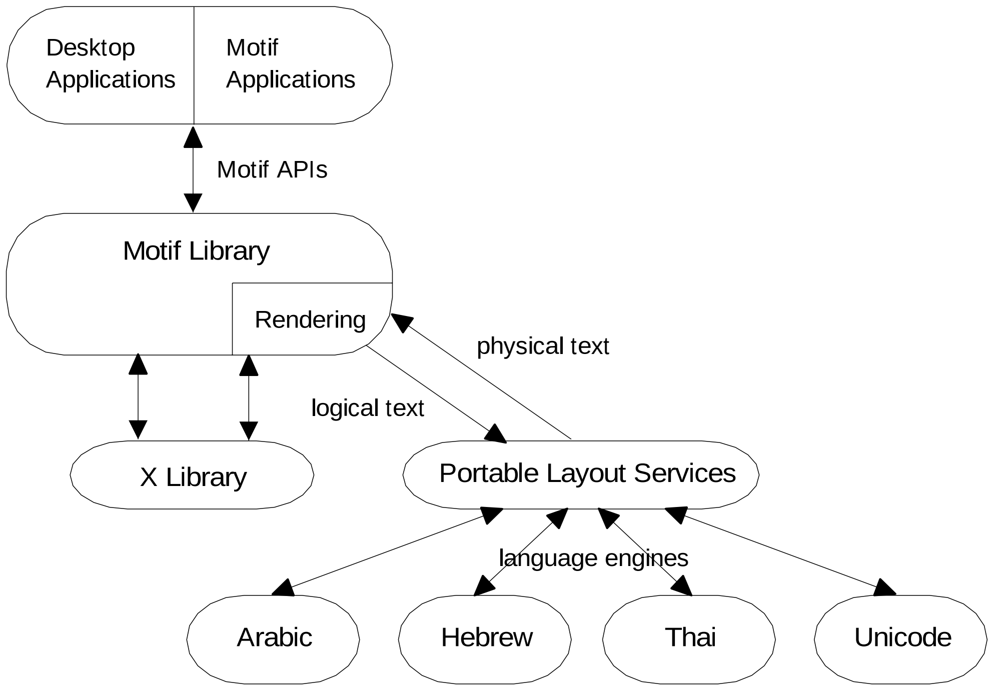


Figure 2–1 CTL language architecture

Desktop and Motif applications, at the top of the stack, use Motif functionality to render text. Motif interfaces with the language engines using PLS. PLS needs to know the current locale only when calling the appropriate language engine.

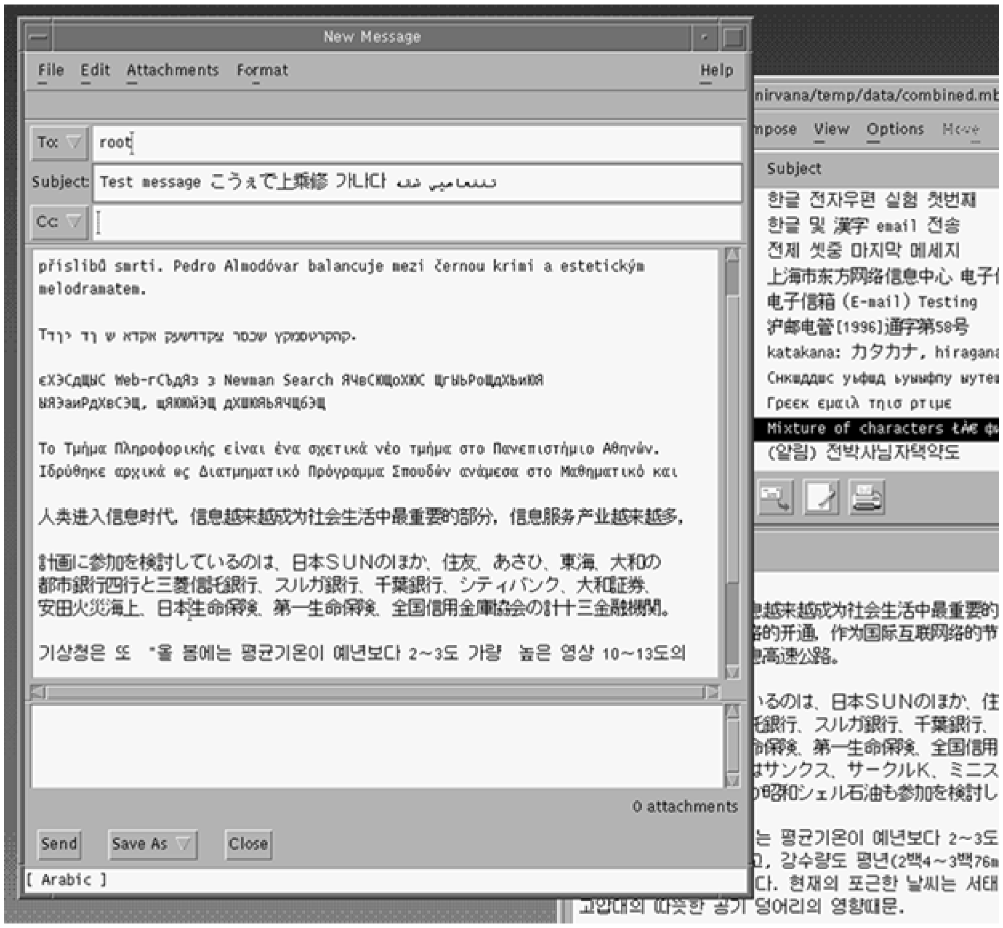
The language engines perform character transformations for shaping and reordering text. The character stream can come from keyboard input, an application’s text strings, or a file.

Note - Because the CTL architecture easily supports new languages, Sun may develop new language engines in future.

2.2 Multiscript Support

Because of the CTL framework, Unicode characters and CTL features can easily be presented together in a Unicode locale.

The UTF-8 locales use the Universal Multiscript Language Engine (UMLE). UMLE is controlled by the utility `genlayouttbl(1)` which can be used to add/customize the script layout. For more information about this utility and the Universal Multiscript Language Engine, see the `genlayouttbl(1)` man pages. Figure 2–2 shows a sample multiscript e-mail in `dtmail` (`en_US.UTF-8` locale).



To: root

Subject: Test message こうぇで上乗修 가나다 ﻨﻨﻨﻌﺎﻤﻴﻲ ﺸﻘﻪ

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

Tיךי דך ש אקדא קעשדדקצ שכסר ץקמסטקקקק.

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

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

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

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

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

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

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

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

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

Figure 2–2   Multiscript e-mail in `dtmail` (en_US.UTF-8 locale)

2.3 Representation Management

The various components of the CTL stack manage different functions of the string, processing, and display of CTL language characters and symbols.

The language engines are responsible for all aspects of character formation and rendering in the appropriate language. The language engines support the following shaping and reordering features:

- Bidirectionality
- Symmetrical swapping
- Numeral shaping
- Contextual analysis
- Positional variation
- Ligation
- Diacritics
- Character clusters

The language engine algorithms apply language-specific rules to the input stream and return the appropriate glyphs for rendering by the `X11` rendition layer.

Motif is responsible for all on screen text manipulations, including:

- Cursor positioning
- Highlighting
- Text selection
- Cut and paste
- Insert/delete

Motif also manages the flow of the character input stream so that the language engine receives only CTL characters for processing.

The application manages the physical layout of text, including alignment, justification, word demarcation, and other page-formatting features. The application also manages the collation of CTL text, non-processed text, and locale-specific information. For example, when displaying currency strings, the application collates the processed number stream and the current locale's monetary format and currency symbol.

2.4 CTL Locales

The Solaris operating environment provides the following CTL locales.

TABLE 2–1 CTL Locales

Locale	Codeset	Typeface	Conversion
Arabic: ar_AR	iso8859-6	Akhbar, Shayyal, Naskh (Regular and Bold)	8859-6 <-> UTF-8 8859–6 <-> Win1256
Hebrew: he_HE	iso8859-8	Arial, CourierNew, TimesNewRoman, LucidaSans (Regular, Bold, Italic, and BoldItalic)	8859-8 <-> UTF-8
Thai: th_TH	iso8859-11 (TIS620.2533)	AngsanaUPC, BrowaliaUPC, CordiaUPC, Lucida (Medium, Bold, Italic, and BoldItalic)	8859-11 <-> UTF-8

*.UTF–8	Unicode 3.0	All of the above	UTF-8 <-> ISO 646(ASCII)
			UTF-8<->ISO 8859-1 ~ -11, -15
			UTF-8<->KOI8-R<->UCS-2
			UTF-8<->UTF-7
			UTF-8<->UTF-16<->UCS-4,
			UTF-8<->PCK(aka SJIS)
			UTF-8<->various euc encodings
			UTF-8<->ISO-2022-JP ~ -TW, -KR, -CN
			UTF-8<->BIG5
			UTF-8<->GBK
			UTF-8<->IBM MBCS CP933, CP935, CP937

Note - CTL locales must be installed before CTL applications can be used.

CHAPTER **3**

Adding CTL Support to an Application

Adding CTL support to an application requires little or no change to the source code. Certain resources are set in the resource file to control the behavior of the language engine for all `Text`, `TextField`, and other static text.

If an application was compiled in an earlier version of Solaris, the code should be recompiled in Solaris 8. Ensure that the application is linked to the default Motif library in Solaris 8 (`libXm.so`) which is linked to the latest Motif library (`libXm.so.4`). Add the flag `-DSUN_CTL` to Makefiles for compilation.

For finer control over CTL attributes, such as cursor positioning and character and cell deletion, use the APIs provided in PLS with its configuration file. For more information about the Portable Layout Services APIs, see *CAE Specification: Portable Layout Services: ISBN 1-85912-142-X*

3.1 CTL Motif Resources

To control CTL functionality in an application, use the following Motif resources:

1. `layoutModifier`
2. Visual mode
3. Default keystroke override
4. `layoutDirection`
5. Dynamically switch screen layout

Step 1. Control the behavior of the language engine through the `layoutModifier` resource in the application resource file. For example, the default numeral type, text type, and text orientation can be changed using the following resource:

```
*layoutModifier: @ls numerals=nominal:contextual,
typeoftext=implicit:visual, orientation=ltr:rtl
```

TABLE 3–1 Common Modifiers

Modifier	Values	Description
orientation=in:out	ltr	left-to-right global orientation (default in and out)
	rtl	right-to-left global orientation
	contextual	context-sensitive orientation
numerals=in:out	nominal	Arabic numerals (Latin) (default in)
	national	national shaping of numerals
	contextual	numeral shaping based on the context of the surrounding text (default out)
swapping=value	yes	enable symmetric swapping (default)
	no	disable symmetric swapping
typeoftext=in:out	visual	visual order (default out)
	implicit	logical order (default in)
	explicit	logical order with embedded control

For a complete list of layout modifiers, see *Portable Layout Services: Context-Dependent and Directional Text: ISBN 1-85912-075-X*.

Step 2. Configure Hebrew and Arabic locales to support Visual mode for editing in the resource file as follows:
`*editPolicy: XmEDIT_VISUAL` (default is `XmEDIT_LOGICAL`)

Step 3. To *override* the default keystrokes in an application, make the necessary changes in the resource file, as shown in the following example:

```
*XmText.translations: #override \n\
c <Key>q:right-word() \n\
c <Key>w:left-word() \n\
c <Key>g:forward-word(extend) \n\
c <Key>h:backward-word(extend) \n\
c <Key>f:next-line(extend) \n\
c <Key>b:previous-line(extend) \n\
c <Key>>:right-character() \n\
c <Key><:left-character() \n\
c <Key>t:prev-cell() \n\
c <Key>v:forward-cell() \n\
c <Key>+:delete-right-word() \n\
c <Key>-:delete-left-word() \n\
c <Key>r:right-character(extend) \n\
c <Key>l:left-character(extend)
```

Step 4. The Motif resource `layoutDirection` can be used to set the right to left screen layout. In this screen mode, the screen origin is the top right corner and all application Motif widgets are mirrored. The following example shows how to set this in the resource file:

```
*layoutDirection: XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
```

This resource changes the layout of the whole application. To change the layout of some widgets only, such as `Text` or `TextField`, use the following:

```
*XmText*layoutDirection: XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
*XmTextField*layoutDirection: XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
```

Step 5. The screen layout can be dynamically switched for the `Text` and `TextField` widgets by defining the dynamic switching hot keys in the Motif translation table. The following example shows how to add this function to the resource file:

1-`Text`:

```
 *XmText.translations: #override \n\
Ctrl<Key>m:toggle-rtl-mode()
```

2-`TextField`:

```
*XmTextField.translations: #override \n\
Ctrl<Key>m:toggle-rtl-mode()
```

Note - `CTRL+m` is defined as the key combination to dynamically switch the screen layout. Any key combination which doesn’t interfere with the application hot keys can be defined instead. This entry can also be appended to the original key translation table as discussed in Step 3 above.

3.2 CTL Printing

Each locale provides text input and display in the locale language and in English. Solaris 8 also includes the print utilities `ctlmp` and `ctlconvert_text` which can be used to print the CTL text.

- `ctlmp` creates a PostScript file from a text file created in a CTL locale. The utility performs the necessary CTL transformations and ensures WYSIWYG printing support. For more information on `ctlmp`, see `man ctlmp`.
- `ctlconvert_text` creates a standard print file from a text file created in a CTL locale for printing on non-PostScript devices.