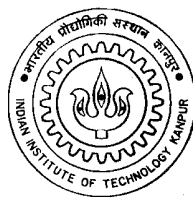


Item-Multilingual X-Windows Indian Script Terminal

*A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology*

by
Jyotirmoy Saikia



to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2002

Certificate

This is to certify that the work contained in the thesis entitled “*Iterm-Multilingual X-Windows Indian Script Terminal*”, by *Jyotirmoy Saikia*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2002

(Dr. Rajat Moona)

Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

In this thesis, **iterm** - a multilingual X windows based Indian script terminal has been developed. It allows entry and simultaneous display of ten Brahmi based Indian scripts as well as English. ISCII character coding has been used for Indian script characters.

To overcome the problems due to the non-linearity of Indian scripts, ISCII library has been developed. It provides configuration-driven conversion from code space to glyph space.

Iterm supports two types of keyboard overlays, viz. - Inscript and Roman phonetic keyboards. Fonts with any encoding can be used for display of Indian script characters. Currently, ISFOC fonts and Modular-infotech fonts are supported by iterm.

The software has been tested against various applications such as editors, filters, compilers etc.

Acknowledgements

I would like to express my deep gratitude to my thesis supervisor Dr Rajat Moona for his able guidance and valuable suggestions. Whenever I faced any difficulty, I just rushed to him. And he was very kind enough to listen to my problems patiently and then showing me my mistakes, teaching me many new things and also coming out instantly with better ideas. I take this opportunity to express my thankfulness to him.

I am thankful to each and every member of the ISCII group - Amit, Anil, Aravind, Ashok, Gaurav, Jyoti, Nitish, Ramesh, Sanatan, Saumen and Vivek. It was such a wonderful team!

I thank all my friends who made my stay here a memorable one. I would like to specially say thanks to Parthajit, Rajarshi, Suman, Rajib da and Nalin da.

I also thank my parents for their constant encouragement, help and support.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Terminal emulators for X	1
1.3	Features of iterm	2
1.4	Organization of thesis	3
2	Background	4
2.1	Terminal	4
2.2	Pseudo Terminal	5
2.3	ISCII Standard	6
2.4	Overview of Indian Languages	7
2.4.1	Indian Script Word Syntax	8
2.5	Support for Internationalization: Character, Bytes and Columns	9
3	ISCII Library	10
3.1	Introduction	10
3.2	Issues behind the development of ISCII library	11
3.3	Design and Implementation of ISCII library	11
3.3.1	How to write a configuration file?	13
4	Design and Implementation of Iterm	15
4.1	General Design of Iterm	15
4.2	Font Specification File	16
4.3	Support for Variable Width Fonts	17

4.4	Screen Buffer	17
4.5	Syllable Analyzer	17
4.6	Refresh Function	18
4.7	Cursor Movements	19
5	Results	20
6	Conclusion	25
6.1	Future Work	25
A	Man page of ISCII Library	27
B	Man page of iterm	31

List of Figures

2.1	Block Diagram of a Terminal	5
2.2	Block Diagram of a Pseudo-terminal	6
3.1	Design of the ISCII library	12
5.1	Results of alias and ls commands	21
5.2	Testing cat and cc commands	22
5.3	Editing a file in Vi editor	23
5.4	Testing locale	24

Chapter 1

Introduction

1.1 Motivation

Terminals provide an interface through which the computer and users communicate with each other. There are various terminals available under X windows for different languages of the world like English, Chinese, Japanese, Korean etc [4].

India is a multi-linguistic country, with 18 constitutionally recognized languages, written in various scripts. Out of these, Urdu and Kashmiri use Perso-Arabic scripts and rest all languages use Brahmi based scripts.

Despite the presence of so many different languages, there has not yet been a terminal under X which supports all the Indian languages. The motivation for this thesis was to develop a multilingual terminal software running under X window, which can allow input and output of Indian script characters.

1.2 Terminal emulators for X

Most commonly used terminal emulator for X which supports input and output of English texts is xterm [11]. The xterm program emulates VT102 and Tektronix4014 terminals. The character encoding used for xterm is ASCII.

Rxvt [8] is another commonly used terminal under X. Rxvt is a colour vt102 terminal emulator intended as an xterm replacement for users who do not require

features such as Tektronix 4014 emulation and toolkit-style configurability. As a result, rxvt uses much less swap space – a significant advantage on a machine serving many X sessions.

The kterm [4] program is a multilingual terminal emulator which emulates VT102 and Tektronix4014. It has support for input and output of Chinese, Japanese and Korean text. Multi-byte encoding is used for storing the text.

1.3 Features of **item**

The **item** program is an X windows based multilingual software, similar to rxvt, providing an interface for I/O of ten Brahmi based Indian scripts and English. The salient features of **item** are as follows:

- The **item** emulates VT102 terminal. The character encoding used for Indian language text is ISCII (Indian Script Code for Information Interchange) which is an eight-bit code. The lower 128 characters are same as in ASCII and the upper 128 characters caters to all the 10 Indian scripts based on the ancient Brahmi script.
- The entry and simultaneous display of text written in Indian languages and English is supported by **item**.
- Both fixed and variable sized fonts can be used to display text in English or any Indian scripts.
- The **item** supports Inscript keyboard and Roman phonetic keyboard overlays for Indian language text entry in addition to the normal English keyboard.
- A wide range of application program can run under **item**, without any modification.
- The cursor movement is over a syllable.
- Currently **item** is supporting the free fonts available from CDAC which are known as ISFOC standard fonts. But any single byte font encoding can be used for display of Indian script text.

1.4 Organization of thesis

Rest of the thesis is organized as follows. Chapter 2 introduces the terms and concepts relevant to discussions held in later chapters. Chapter 3 discusses the ISCII library used by the item program. Chapter 4 discusses the design and implementation details of the Indian language support in the item. Results of various tests are presented in chapter 4. Finally chapter 5 concludes the thesis. Appendix A is the man page of the ISCII library. Appendix B is the man page for item.

Chapter 2

Background

An attempt has been made in this thesis to develop an X windows based terminal emulation software which supports all the Brahmi based Indian scripts. Before discussing the actual design for this terminal, it is essential to understand the workings of a terminal and a pseudo-terminal, character encodings, issues in computer representation of Indian languages etc. This chapter provides these backgrounds which will be helpful while discussing the design of iterm.

2.1 Terminal

A terminal provides users an interface to communicate with application programs executing on host computer. Terminals consist of *transmit* and *receive blocks*. The transmit blocks interfaces with keyboard and sends the characters typed in to the computer. The receive block receives characters from the host and interfaces with the monitor for displaying them. Terminals support the standard I/O operations as well as terminal specific operations to control input/output behaviour and cursor editing. Figure 2.1 shows the basic building block of a terminal.

Apart from displaying the normal characters, terminals also receive control sequences, specifying some special action to be taken. Each terminal provides different set of control sequences.

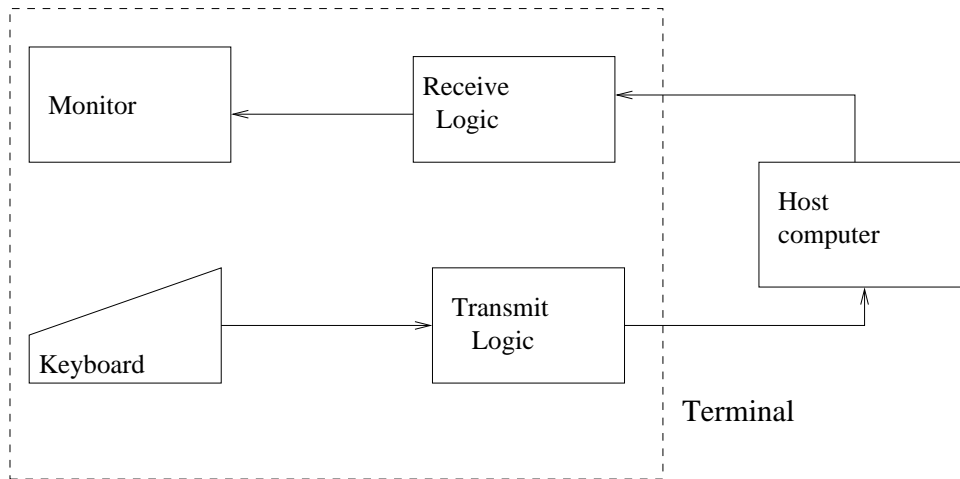


Figure 2.1: Block Diagram of a Terminal

2.2 Pseudo Terminal

The term *pseudo-terminal*[9] implies that it looks like a terminal to an application program, but it's not a real terminal. Pseudo terminals are pairs of *master* and *slave* terminals, acting together like one terminal unit. The slave terminal is the interface, acting like a terminal unit for the user programs, while the master represents the other end of the link. Figure 2.2 shows the basic building blocks inside a pseudo terminal.

A process opens the pseudo-terminal master and then produces a child process by calling the *fork* system call. The child process establishes a new session and opens the corresponding pseudo-terminal slavs. Then the child duplicates it to be standard input, standard output and standard error and calls *exec* to run the shell process. The pseudo-terminal slave becomes the controlling terminal for the child process. For a user process running above the slave, it appears that its standard input, standard output and standard error are a terminal device. Anything written to the master appears as input to the slave and vice-versa.

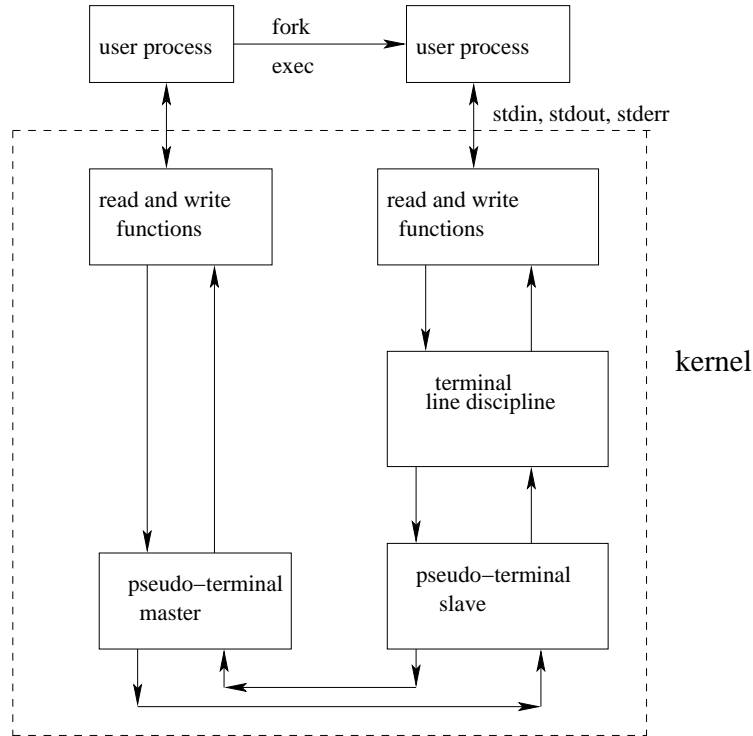


Figure 2.2: Block Diagram of a Pseudo-terminal

2.3 ISCII Standard

ISCII stands for Indian Standard Code for Information Interchange [7]. It is an 8-bit code and the lower seven bits are same as ASCII standards. So the lower 128 characters are same as ASCII and the upper 128 characters cater to the need of the Brahmi based Indian script characters. ISCII character set is a super set of all the characters required in all the ten Brahmi based script.

A code for all Indian language is made possible by their common origin from the Brahmi script. Similarly, an optimal keyboard overlay for all Indian languages is made possible by the phonetic nature of the alphabet.

There are many advantages for having a common code and keyboard overlay for all the Indian scripts. Any software, which allows ISCII codes to be used, can be used for any Indian script. Furthermore, immediate transliteration between different

Indian scripts becomes possible.

Two important properties of ISCII codes are:

- Phonetic sequence - The ISCII characters, within a word, are kept in the same order as they would get pronounced. For example,
- Display independence - ISCII codes allow a complete delinking of the codes from the displayed fonts. An ISCII syllable can be displayed using combinations of basic shapes. Different implementations can choose variant techniques in combination of these basic shapes.

2.4 Overview of Indian Languages

All Brahmi-based Indian scripts are phonetic in nature. The alphabet in each may vary somewhat, but they all share a common phonetic structure. The differences between scripts are primarily in their written forms, where different combination rules get used [7].

Devanagari character set can be categorized into *vowels*, *consonants*, *matras*, *modifiers*, *numerals*, *punctuation* and some special symbols like *halant* and *nukta*. In our following discussion, we will use VOWEL_A for the following vowel:

ॐ

In Devanagari scripts, consonant have an implicit VOWEL_A attached to it. A *pure consonant* is obtained by attaching a special symbol called *halant* to the *consonant*. Most of these *pure consonants* have a different display shapes.

Each *vowel* except VOWEL_A has a corresponding *matra* which can be attached to a *consonant* to form composite characters. The modifiers are *anuswar* (causing nasalaization), *visarg* (introducing apsriration), *chandrabindu* (causing prolongation). The diatric mark *nukta* is used along with some *consonants*, and is mostly used to represent some foreign sounds. All *punctuation* marks used in Indian scripts are

similar to the ones used in English, except for the full-stop, instead of which *viram* is used.

Devanagari script is a logical composition of its constituent symbols in two dimensions. The *matras*, *modifiers*, *halant*, and *nukta* can be attached to a *vowel* or a *consonant* to the right, left, top or bottom.

Two or more *pure consonants* combine to form a *conjunct*. *Conjuncts* can form composite characters by addition of *matras*, and *modifiers* in the same way as *consonants*. Shape of these *conjuncts* usually vary from those of the constituting *consonants*.

2.4.1 Indian Script Word Syntax

An Indian script word contains one or more syllable. The syllables are formed from the characters and there are certain rules how these some characters combine to form a syllable. The following rule in Backus-Naur Formalism (BNF) states syllable formation syntax [7]:

```

Word           ::= {Syllable} [Cons-Syllable]
Syllable       ::= Cons-Vowel-Syllable | Vowel-Syllable
Vowel-Syllable ::= Vowel[Modifiers]
Cons-Vowel-Syllable ::= [Cons-Syllable]Full-Cons[Matra][Modifiers]
Cons-Syllable  ::= [Pure-Cons][Pure-Cons]Pure-Cons
Pure-Cons      ::= Full-Cons Halant
Full-Cons      ::= Consonant [Nukta]

```

Following conventions are used in the syntax given above:

```

::= defines a relation.
{ } encloses items which may be repeated one or more times
[ ] encloses items which may or may not be present
| seperates items, out of which only one can be present

```

The above syntax ignores the followings cases:

1. Some vowels derived through Nukta and the Avagrah symbol
2. The Explicit Halant (Halant + Halant) and Soft Halant (Halant + Nukta)

2.5 Support for Internationalization: Character, Bytes and Columns

One ASCII character is always represented in one byte and it occupies one column in console or X terminal emulators (fixed font for X). But such an assumption should not be made for I18N programming and the distinction among number of bytes, characters, and columns should be made.

In multibyte encodings, two or more bytes are needed to express one character. In the case of CJK (Chinese, Japanese, Korean) scripts, character encoding is multibyte and each character occupies an integral multiple of number of columns[5].

One ISCII character is expressed in one byte. But number of columns in case of Indian script characters can never be predicted as Indian script characters combine in various ways in a two-dimensional space to form composite characters and conjuncts. This is one of the most important issues to be kept in mind while developing any Indian language software.

Chapter 3

ISCII Library

Indian languages are non-linear in nature. Besides, there is no one-to-one mapping between character codes and glyphs. So there was a requirement for a mechanism for converting the character codes to the glyphs, keeping in mind the various rules specific to that script. ISCII library have been developed for that purpose. This chapter discusses the design and implementation of the ISCII library that is used by the `item` program.

3.1 Introduction

ISCII library contains some code-conversion routines for converting from one coding mechanism to another, e.g. ISCII coded data to glyphs , glyphs to ISCII coded data etc. A configuration-driven approach is adopted for converting a source string to an array of target codes. The configuration file provides language and encoding specific rules.

The `item` program uses the ISCII library for converting the ISCII codes to the glyphs of a particular font.

3.2 Issues behind the development of ISCII library

English has the advantage of linearity, that is, the characters are typed in and displayed in the same sequence as it is written. Besides, there is one-to-one mapping between ASCII character codes and glyphs of any English fonts.

Contrary to English language, Indian language have some peculiar features which are discussed below:

- Indian languages are non-linear in nature. The display order of Indian language characters are not necessarily the same as they were typed from the keyboard. The symbols of the script can be attached either to the left, right, top or bottom of the previous symbol.
- One character code may be represented by several display shapes available in the font. On the other hand, several character codes may combine to form a single display shape.
- As the characters are typed, they may combine with earlier characters to form an entirely new display shape.
- There are some fixed rules which determines how several character codes can combine to form a new display shape. These rules are specific to the particular script.
- The fonts for Indian scripts may follow different encoding techniques. There is no standardisation for font encodings of Indian scripts. There should be flexibility to support any font.

3.3 Design and Implementation of ISCII library

To overcome the challenges thrown by Indian languages as discussed in the previous section, the ISCII library have been developed. The ISCII characters are converted into the glyphs using the functions in this library which is configuration driven, that is, configuration files guide these conversion. Various language and font encoding rules have been incorporated into these configuration files.

Each configuration file deals with a source space and a target space. For example, for converting ISCII codes to the ISFOC fonts of Assamese language, ISCII code space is the source space and Assamese ISFOC font is the target space. This particular configuration file will contain information entailing the rules about how the various characters are combined to form composite characters and conjuncts in Assamese language.

A parser has been developed using Lex and Yacc, which parses the configuration file and generates various tables. The conversion routines consults these tables for converting the source space codes to target space codes. Figure 3.1 shows the basic design of the ISCII library.

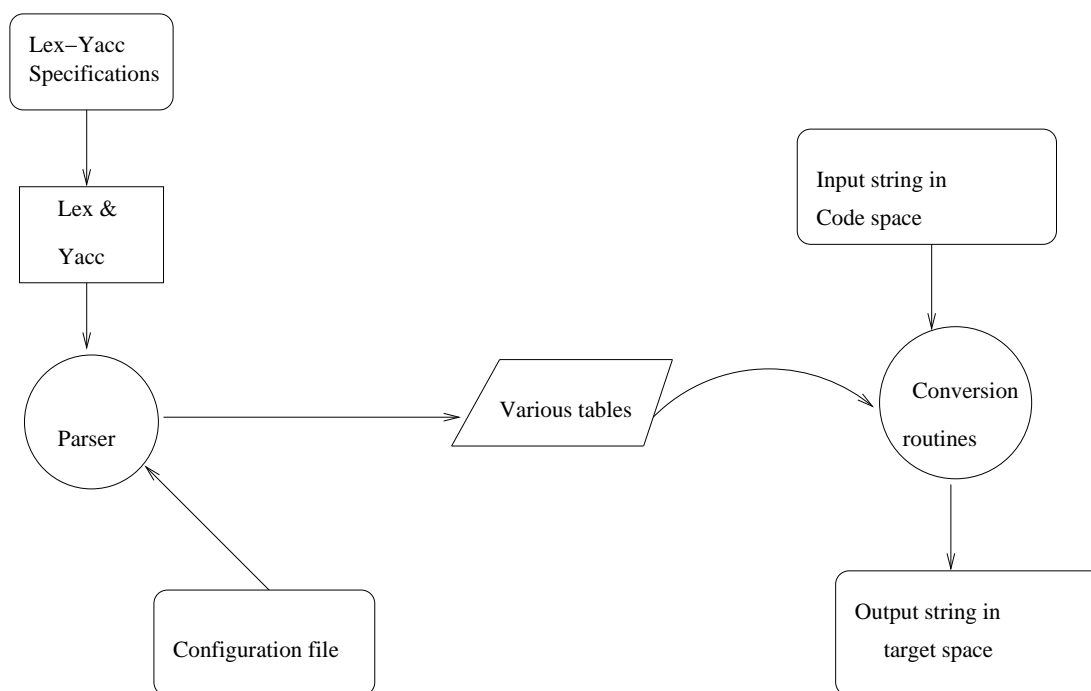


Figure 3.1: Design of the ISCII library

3.3.1 How to write a configuration file?

The configuration file basically consists of three different types of statements: definition, grouping rules and rewriting rules. Each statement is terminated by semi-colon.

In source space, numbers are preceded by `#` symbol and in target space the numbers are preceded by `%` symbol. The numbers can be written in decimal, hexadecimal or octal number format of C language. For example, in the configuration file for `iscii2glyph` conversion, we may have two definitions as given below:

```
KA = #179;
G_AA = %0x61 %0x41 ;
```

which means KA is the ISCII code 179 and G_KA is the glyphs 61h and 41h.

Grouping rules specify the various categories in the source space and target space. Each category consists of one or more constituent symbols (identifiers or numbers), separated by `|` (the vertical bar). For example, in the configuration file for `iscii2glyph` conversion, we may have a grouping rule like the one given below:

```
CONSONANT = KA | KHA | #181;
```

which states that KA, KHA and 181 constitute the category CONSONANT in the codespace.

Definitions and grouping rules can be written in terms of identifiers or numbers of the same space. But source and target space numbers/identifiers cannot be mixed together in the same definition or grouping statement.

Rewriting rules provide the language and encoding specific rules. The left hand side of a rewriting rule consists of source space identifiers and source space numbers only. It cannot contain any target space identifiers or numbers in the left hand side. The right hand side of a rewriting rule may contain any sequence of source and target space identifiers and numbers.

The source string is compared with the left hand side of each rewriting rule. The rule that matches first is taken and the part of the input string that matches with the rule are replaced by the right hand side of the rule. The rules are applied

repeatedly until the output string consists of only target space numbers.

Let's illustrate rewriting rules with an example from ISCII to glyph conversion:

```
CONSONANT HALANT CONSONANT LEFT_MATRA -> G_LEFT_MATRA($4) $1 $2 $3;  
CONSONANT HALANT CONSONANT -> G_HALF_CONSONANT($1) $3;  
CONSONANT -> G_CONSONANT($1);
```

The \$n in right hand side means the n-th symbol in the left hand side of the rule. Referring to the first rule, G_LEFT_MATRA(\$4) means corresponding G_LEFT_MATRA of \$4, which is LEFT_MATRA.

Suppose that the input ISCII string consists of a consonant, halant, consonant and a left_matra(e.g. matra_i) in that order. Then the first rule will match first and the input string will be replaced by the right hand side of the first rule, that is, the glyph code of the left-matra, ISCII codes of the first consonant, halant and ISCII code of the other consonant. Now the second rule will match and part of the input string matching this rule will be replaced by the right hand side of the rule. So this time, the output string will contain glyph code of the left matra, glyph code of the half-consonant of the first consonant and the ISCII code of the third consonant. The remaining ISCII code of the consonant will match with the third rule and the ISCII code of the consonant will be replaced by the glyph code of the consonant. Now the output string consists entirely of glyphs and so rules will not be applied any more. This becomes the final output string.

In ISCII to glyph conversion, conjuncts are treated just like consonants. In order to do that, each of the conjuncts is first given an intermediate code in the source space and then this code is included in the grouping rule for consonants. Correspondingly the glyph code of the conjunct is included in the grouping rule for the category of glyph of consonants. So whenever a conjunct is found in the input string, it is first rewritten with an intermediate code in the source space which makes it as a consonant. This same approach is taken for nukta consonants also, i.e, for the consonants after which a nukta character can be attached.

Chapter 4

Design and Implementation of Iterm

The **iterm** program has been developed by modifying **rxvt** [8] - a VT102 terminal emulator for X. Thus **iterm** inherits all the features of **rxvt** and in addition provides entry and display of all Brahmi based Indian scripts and variable width English fonts. This chapter discusses how **rxvt** was modified to provide support for Indian language.

4.1 General Design of Iterm

Iterm, being derived from **rxvt**, emulates VT102 terminal. It supports all the ten Brahmi based Indian scripts in addition to English. ISCII character coding [7] has been used for Indian script characters.

A menu has been provided to select the script. On choosing the script, the terminal generates the ATR [7] character and the script attribute and send these to the application. In ISCII standard, an ATR character followed by a valid script attribute determines the script for the whole row or till another ATR character in the same row. Besides, two boxes has been provided just beside the menu to select the display attribute of the current script. One can select a bold, italics and bold-italics font by clicking on these boxes. A status bar has also been provided which displays the current character under the cursor.

There are two types of keyboard overlays supported by **iterm**, viz - Inscript

keyboard and Roman phonetic in addition to the normal English keyboard. Initially the keyboard is in English mode. By pressing the scroll lock key, one can cyclically change the keyboard from English to Roman Phonetic, Roman Phonetic to Inscript and Inscript to English. Besides, there is provision for temporary switching of keyboard modes for typing a single character by pressing the right-ALT key.

4.2 Font Specification File

This specification file provides the name of the fonts corresponding to various display attributes, like normal, bold, italics etc for a particular script. The name of this specification file can be supplied from command line . If it is not given in command line option, this is searched in the X resource database. If it is npt found in the X resource also, then the default specification file is read from `/usr/share/iterm/spec`

A sample font specification file is shown below:

```
<DEV>
NOR = -altsys-dv_ttyogesh-medium-r-normal--20-0-50-50-p-0-iso8859-1;
BLD = -altsys-dv_ttyogesh-bold-r-normal--20-0-50-50-p-0-iso8859-1;
ITA = -altsys-dv_ttyogesh-medium-i-normal--20-0-50-50-p-0-iso8859-1;
BLD ITA = -altsys-dv_ttyogesh-bold-i-normal--20-0-50-50-p-0-iso8859-1;
</DEV>

<TML>
NOR = -altsys-tm_ttvalluvar-medium-r-normal--20-0-50-50-p-0-iso8859-1;
BLD = -altsys-tm_ttvalluvar-bold-r-normal--20-0-50-50-p-0-iso8859-1;
ITA = -altsys-tm_ttvalluvar-medium-i-normal--20-0-50-50-p-0-iso8859-1;
BLD ITA = -altsys-tm_ttvalluvar-bold-i-normal--20-0-50-50-p-0-iso8859-1;
</TML>
```

4.3 Support for Variable Width Fonts

Indian scripts characters are of variable widths by their nature. Some matras and modifiers have zero width; some matras have a width of a few pixels and consonants and vowel etc have width of of a few more pixels. Original **rxvt** used to support fixed width fonts only and it used to draw each fixed with character in a column position. The concept of column has been removed in **iterm**. Instead of finding the xpixel position for drawing a character by calculating it from its column, **iterm** calculates the xpixel position by adding the pixel width of the previous characters. While calculating the widths of previous characters, it is to be kept in mind that each character may belong to a different font because of ATR characters.

4.4 Screen Buffer

Codes received by the program are stored in a screen buffer. This screen buffer contains the entire text displayed in the current screen as well as the text saved for scrolling.

The screen buffer is basically an array of pointers and each of these pointers point to a dynamically allocated memory space. Each of these pointers correspond to a row in the screen and each of these memory space pointed by these pointers contain the text for each row.

Each of these rows are considered to be of infinite length (that is, a very high value) instead of some fixed values as in original **rxvt**. So there is no restriction on the number of characters per row.

4.5 Syllable Analyzer

For Indian languages, a worst case consonant syllable can contain[7]:

C N H C N H C N H C M D

A worst case vowel syllable can contain:

V D
where C = Consonant
V = Vowel
N = Nukta
H = Halant
M = Matra
D = Modifier

But in actual case, the consonant syllable may be more complexed as the above notation does not take into consideration Halant + Halant (Explicit Halant) and Halant + Nukta (Soft Halant) and the vowels derived through Nukta and the Avagrah symbol.

A parser for a syllable analyzer has been developed which takes into consideration all possible cases of a syllable for Indian scripts. This syllable analyzer is run for each row of the characters that is being drawn. The syllable analyzer gives the boundaries of each syllable. For the case of an invalid syllable, each invalid symbol is considered to be a syllable in itself.

This syllable analyzer is used by the refresh function of **iterm**.

4.6 Refresh Function

The **rxvt** code has two screen buffers: current one and previous one. While refreshing the screen, each row was compared between these two buffers and only the difference of characters between them were displayed in the screen. The **iterm** code has been modified in such a way that whenever some characters were coming for refresh, the syllable analyzer was consulted first and the entire syllable to which these characters belong is refreshed. Again, any text being refreshed are first passed through the conversion functions provided by the ISCII library to get the appropriate glyphs.

4.7 Cursor Movements

In English there is one to one correspondence between the input character and the display shape and so the cursor always shows the actual character. However in case of Indian scripts, multiple characters may combine together to form a single glyph. In **iterm**, the cursor is drawn in reverse video mode over the entire syllable to which the current character belongs to. At the same time, the status bar shows the current character on which the cursor is actually present.

Chapter 5

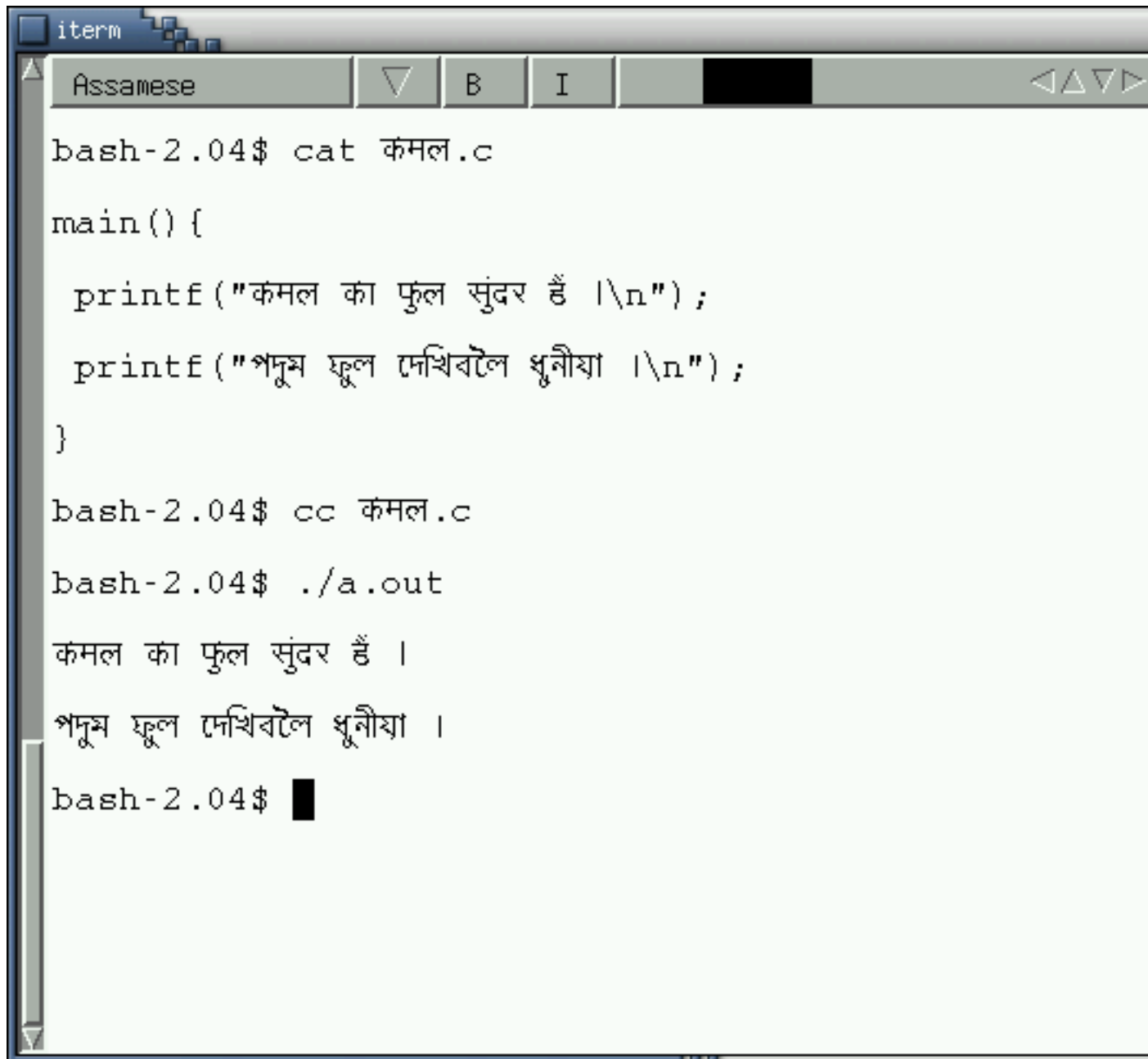
Results

The **iterm** supports all application programs that run on **rxvt**. The software has been tested against some application programs like **vi**, **more**, **cat**, **ls** and C compiler. Screenshots of the terminal has been taken while using different applications.

In Figure 5.1, the output of **alias** and **ls** commands are seen. In figure 5.2, it is tested for **cat** and **cc** commands. Figure 5.3 shows a file being edited with **vi** editor. Figure 5.4 shows **date** and **ls** commands after setting the locale to Bengali locale.

```
bash-2.04$ alias सूची='ls -l'
bash-2.04$ सूची
total 92
-rw-r--r--  1 jms      nscd         468 Apr 19
-rw-r--r--  1 jms      nscd       21395 Apr 19
-rw-r--r--  1 jms      nscd         300 Apr 19
-rw-r--r--  1 jms      nscd         219 Apr 19
-rw-r--r--  1 jms      nscd       1797 Apr 19
-rw-r--r--  1 jms      nscd          60 Apr 19
-rw-r--r--  1 jms      nscd          90 Apr 26
-rw-r--r--  1 jms      nscd      25650 Apr 22
-rw-r--r--  1 jms      nscd     15248 Apr 19
bash-2.04$
```

Figure 5.1: Results of alias and ls commands



```
bash-2.04$ cat कमल.c
main() {
    printf("कमल का फुल सुंदर हैं |\n");
    printf("পদুম ফুল দেখিবলৈ ধুনীয়া |\n");
}
bash-2.04$ cc कमल.c
bash-2.04$ ./a.out
कमल का फुल सुंदर हैं ।
পদুম ফুল দেখিবলৈ ধুনীয়া ।
bash-2.04$
```

Figure 5.2: Testing cat and cc commands

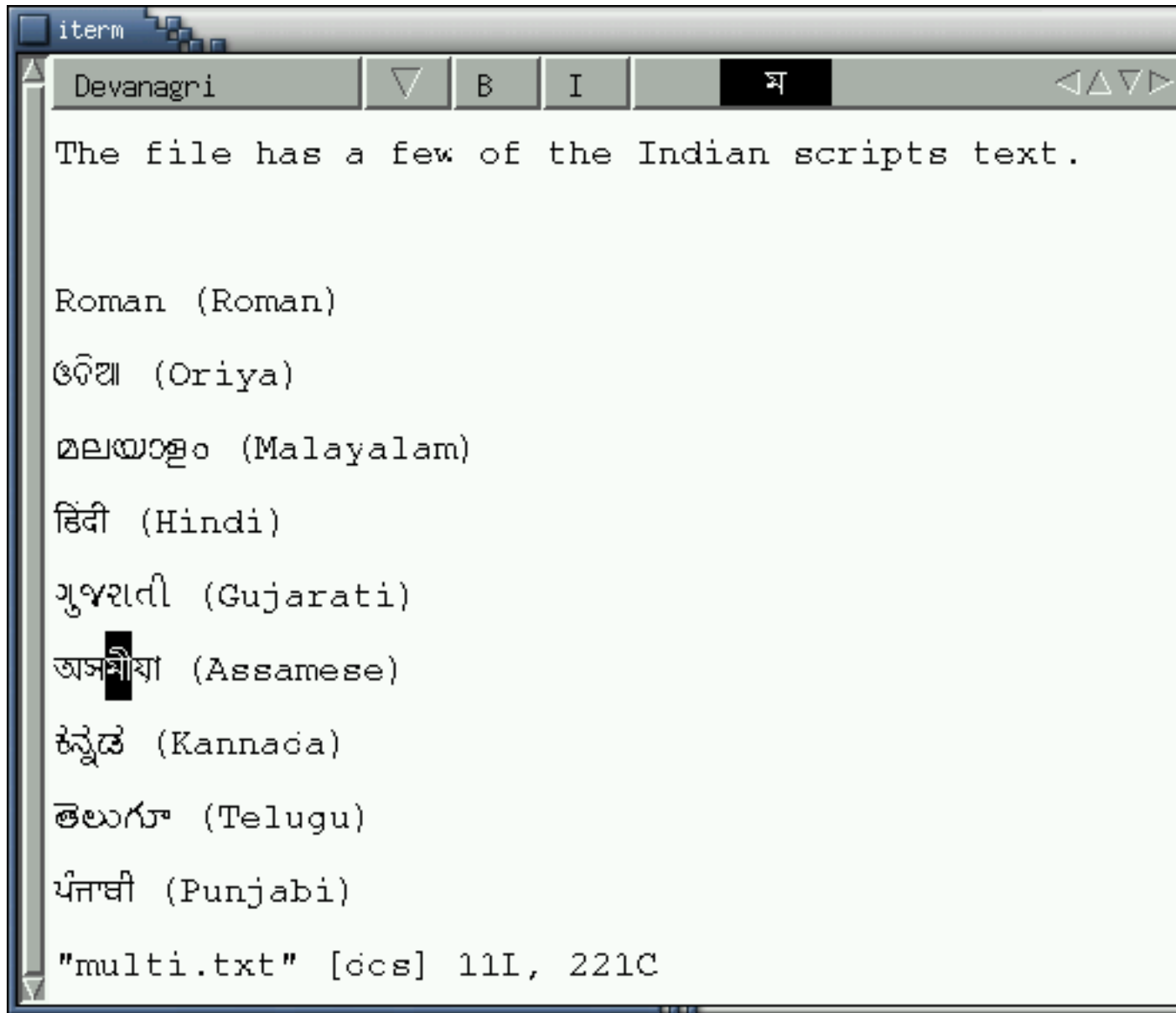


Figure 5.3: Editing a file in Vi editor

The image shows a terminal window titled 'iterm' with a menu bar containing 'Assamese', a dropdown arrow, 'B', 'I', a black rectangle, and navigation arrows. The terminal content is as follows:

```
bash-2.04$ export LC_ALL=bn
bash-2.04$ date
শনি এপ্ৰিল 27 00:01:56 EDT 2002
bash-2.04$ ls -l
total 44
-rw-r--r--  1 jms      nscd          468 এপ্ৰিল 26
-rw-r--r--  1 jms      nscd       21395 এপ্ৰিল 26
-rw-r--r--  1 jms      nscd         300 এপ্ৰিল 26
-rw-r--r--  1 jms      nscd         219 এপ্ৰিল 26
-rw-r--r--  1 jms      nscd        1797 এপ্ৰিল 26
-rw-r--r--  1 jms      nscd          60 এপ্ৰিল 26
bash-2.04$ █
```

Figure 5.4: Testing locale

Chapter 6

Conclusion

The **iterm** has underline been developed from **rxvt**, a VT102 terminal emulator for X. It can support simultaneous display and entry of all the ten Brahmi based script. The character coding used is ISCII[7].

Item supports both fixed width and variable width fonts. It can work with any type of font. To change fonts, the font specification file needs to be modified and the corresponding encoding file in the ISCII library needs to be written, if it is not already present. Currently ISCII library supports all ISFOC[3] and modular[6] fonts.

Two types of keyboard overlays are provided: Inscript and Roman Phonetic. Scroll-lcok key is used to permanently switch the keyboard from one mode to another and right-ALT key is used for a temporary switch.

The **iterm** and **ISCII** library are part of the iLinux distribution from the TDIL[10] group, IIT Kanpur.

6.1 Future Work

The **iterm** works only for ISCII character coding. It does not support Unicode standards. Iterm should be given Unicode support also. Besides, a horizontal scroll bar is required as there is no restriction on column length and so long lines go out of the screen. A horizontal scroll bar will help in scrolling the terminal window

horizontally to view any long lines that go beyond the screen width.

Appendix A

Man page of ISCII Library

NAME

codeconversionbyname, codeconversionbyencoding, getencoding - library functions for handling ISCII data

SYNOPSIS

```
#include <isciilib.h>
```

```
unsigned char *codeconversionbyencoding(const unsignedchar *sourcestr,  
int sizeofsourcestr, unsignedchar *targetstr, int *sizeoftargetstr, const  
char *encoding, int *errorcode);
```

```
unsigned char *codeconversionbyname(const unsigned char *sourcestr,  
int sizeofsourcestr, const char *sourceformat, unsignedchar *target-  
str, int *sizeoftargetstr, const char *fontname, int *errorcode);
```

```
int getencodingbyname(const char *fontname, const char *source-  
format, char *encoding); const char *fontname, int *errorcode);
```

```
int getaltnamebyname(const char *indianfontname, char *matching-  
fontname, float *mfactor);
```

DESCRIPTION

The function **codeconversionbyencoding()** converts an sourcestring pointed by sourcestr to an array of target codes pointed by targetstr according to the encoding name encoding. The size of the source string is provided in the integer value sizeofsourcestr. The size of the target array is provided in the integer value pointed by sizeoftargetstr and this size should be large enough to accomodate all the target codes generated from the sourcestr. Upon return, sizeoftargetstr is set to the number of target codes translated into the targetstr. The return value of sizeoftargetstr never exceeds the value provided at the entry. The encoding name encoding refers to a configuration file where various rules for the translation from source space to target space are defined. The environment variable ENC_PATH contains the name of the directories, separated by semi-colons, where the encoding file is searched in that order. If ENC_PATH is not defined, then a default value is used as `/usr/lib/X11/fonts/encodings;/usr/share/fonts/encodings;/etc/X11/fs/encodings;$HOME/encodings`. Examples of some of the encodings are "ISFOC_DEV", "ISFOC_ASM", "ISFOC_BNG", "ISFOC_TML", "ISFOC_TLG", "ISFOC_ASM", "ISFOC_ORI", "ISFOC_KND", "ISFOC_MLM", "ISFOC_GJR", "ISFOC_PNJ", "ISFOC_RMN", "DEVATEX" etc.

The errorcode returns the error number .

The function **codeconversionbyname()** is similiar, except that the name of the font fontname is passed instead of the encoding. Besides, the format of the source(whether ISCII, UNICODE or UTF-8 etc) sourceformat is also passed. This function uses **getencodingbyname()** function to retrieve the corresponding encoding associated with the fontname.

The function **getencodingbyname()** fills the array encoding with the name of the encoding associated with the font whose name is passed as an argument to it. For doing this, the function uses a *fonttable* which provides a mapping between font name and encoding. The first column of the fonttable contains font names which can be written in X font naming convention. The second column corresponds to the encoding associated with the font. The *fonttable* is

stored in a file example of which is given below:

<code>*dv_tt*</code>	ISFOC_DEV	<code>*courier*</code>	1.1
<code>*as_tt*-*-*</code>	ISFOC_ASM	<code>*helvetica*</code>	1.2
<code>*or_tt*</code>	ISFOC_ORI	<code>*courier*</code>	1.1
<code>*bn_tt*</code>	ISFOC_BNG	<code>*courier*</code>	1.2

This function appends the sourceformat to the name of the encoding retrieved from the second column of the fonttable. For example, if the sourceformat is ISCII and the encoding name mentioned in the fonttable is ISFOC_ASM, then the array encoding will be filled with ISCII_ISFOC_ASM.

The environment variable FONT_ENC contains the name of the directories, separated by semicolons, where the *fonttable* is searched in that order. If FONT_ENC is not defined, then a default value is used as `/usr/lib/X11/fonts;/usr/share/fonts;/etc/X11/fs;$HOME`.

The function **getaltnamebyname()** stores the name of the matching English font name corresponding to the Indian font name indianfontname, from the third column in the fonttable, into the array matchingfontname and stores multiplying factor from the fourth column of the fonttable into the value pointed by mfactor.

RETURN VALUES

codeconversionbyencoding() and **codeconversionbyname()** return the first byte address of sourcestr that could not be converted because of lack of space in the target array. These return (`sourcestr + sizeofsourcestr`) when the entire sourcestr could be successfully converted and returns sourcestr if nothing could be converted.

getencodingbyname() and **getaltnamebyname()** returns 0 when it could find a corresponding encoding for the font fontname. It returns 1 if the fonttable could not be located or the encoding was not found. It returns 10001 to 20000 for syntax error in line number (x-10000) in fonttable, where x lies in between 10001 and 20000.

ERROR NUMBERS

The integer value `errorcode` may have the following values:

- 0 : No error
- 1 : Encoding file not found
- 2 : Fonttable or encoding of a font not found
- 3 : Not enough space
- 4 : Target string is not big enough to store all the target codes
- 5 : No rule matched
- 10001 to 20000 : Syntax error in line number (x - 10000) in fonttable,
where x lies in between 10001 and 20000
- 20001 to 30000 : Syntax error in line number (x - 20000) in
encoding file, where x lies in between 20001 and 30000
- 30001 to 40000 : Undefined symbol in line number (x - 30000) in
encoding file, where x lies in between 30001 and 40000
- 40001 to 50000 : Illegal grouping in line number (x - 40000) in
encoding file, where x lies in between 40001 and 50000
- 50001 to 60000 : Redefinition of a symbol in line number (x - 50000)
in encoding file, where x lies in between 50001 and 60000

Appendix B

Man page of `iterm`

NAME

`iterm` - a VT102 terminal emulator for Indian languages based on the X window system

SYNOPSIS

```
iterm [options] [-e command [ args ]]
```

DESCRIPTION

The `iterm`, version 1.1 is a colour vt102 terminal emulator intended for users intending to use Indian languages for their computing purposes.

USAGE

The terminal in addition to the normal terminal behaviour displays indian language scripts. It uses the ISCII(Indian Script Code for Information Interchange) character coding standard to display Indian scripts. The terminal allows you to work in the following Indian scripts : Assamese, Bengali, Devanagari, Gujarati, Kannada, Malayalam, Oriya, Punjabi, Tamil and Telugu, apart from the usual Roman script. The terminal is set to the script Devanagari at the first invocation. This can be altered by selecting from the drop down menu that appears on clicking on the arrow by the side. The style of the

script in which you wish to type can be chosen by clicking on the menubar buttons showing 'B'(for Bold) and 'I'(for Italics). These can be activated simultaneously too. The default font of the scripts is "Normal". The 'B' and 'I' buttons also indicate the font attribute of the character that is presently under the cursor, and therefore change accordingly assuming the user will be typing characters in the present script with the same attributes. The present script though is not indicated in the script bar. Assuming the user will be using/typing in one particular script usually, the selection of the same has been facilitated by having it displayed in the menubar and the user need only to click on it for reactivating the printing in the displayed script. While moving the cursor in a text from one script to the other the attributes and the script change as per the character under the cursor. The associated attributes are accordingly indicated by the corresponding status of the menubar buttons, the script being implicit is not indicated in the menubar box. Any further typing would be in the same script with the same attributes until they be explicitly altered by clicking on the menubar boxes accordingly as mentioned above. Also the script and the attributes by default are set to Devanagri and Normal when on a new line. The blackened area in the menubar that comes next, displays the character(Indian Language or Roman) under the current cursor location. It could also be "ATR"(short for Attribute) or a single char indicating the attribute code associated with the respective Indian language which are for use by the terminal and are not shown while displaying the Indian scripts. The arrows on the side facilitate cursor movement in a text as per the direction they indicate and also facilitate in switching to the earlier or next command in the sequence of commands previously given.

OPTIONS The following options are recognized by `itern`:

Note that `itern` permits the resource name to be used as a long-option (`-/++` option). For example: `'itern -loginShell -color1 Orange'`.

`-fs fontspecificationfilename`

Specify the font specification file to be used; resource `fontspec`.

-help, -help

Print out a message describing available options.

-display *displayname*

Attempt to open a window on the named X display (-d still respected). In the absence of this option, the display specified by the DISPLAY environment variable is used.

-geometry *geom*

Window geometry (-g still respected); resource `geometry`.

-rv|+rv

Turn on/off simulated reverse video; resource `reverseVideo`.

-ip|+ip

Turn on/off inheriting parent window's pixmap; resource `inheritPixmap`.

-bg *colour*

Window background colour; resource `background`.

-fg *colour*

Window foreground colour; resource `foreground`.

-pixmap: *file[;geom]*

Specify XPM file for the background and also optionally specify its scaling with a geometry string. Note you may need to add quotes to avoid special shell interpretation of the ';' in the command-line; resource `backgroundPixmap`.

-cr *colour*

The cursor colour; resource `cursorColor`.

-pr *colour*

The mouse pointer colour; resource `pointerColor`.

-bd *colour*

The colour of the border between the xterm scrollbar and the text; resource `borderColor`.

-km *mode*

Multiple-character font-set encoding mode; eucj: EUC Japanese encoding. sjis: Shift JIS encoding. big5: BIG5 encoding. gb: GB encoding. kr: EUC Korean encoding. noenc: no encoding; resource multichar_encoding.

-grk *mode*

Greek keyboard translation; iso: ISO-8859 mapping. ibm: IBM-437 mapping; resource greek_keyboard.

-name *name*

Specify the application name under which resources are to be obtained, rather than the default executable file name. Name should not contain '.' or '*' characters. Also sets the icon and title name.

-ls|+ls

Start as a login-shell/sub-shell; resource loginShell.

-ut|+ut

Inhibit/enable writing a utmp entry; resource utmpInhibit.

-vb|+vb

Turn on/off visual bell on receipt of a bell character; resource visualBell.

-sb|+sb

Turn on/off scrollbar; resource scrollBar.

-si|+si

Turn on/off scroll-to-bottom on TTY output inhibit; resource scrollTTYOutput has opposite effect.

-sk|+sk

Turn on/off scroll-to-bottom on keypress; resource scrollTTYKeypress.

-sr|+sr

Put scrollbar on right/left; resource scrollBar_right.

-st|+st

Display normal (non XTerm/NeXT) scrollbar without/with a trough; resource scrollBar_floating.

-iconic

Start iconified, if the window manager supports that option.

-sl *number*

Save number lines in the scrollback buffer; resource `saveLines`.

-b *number*

Internal border of number pixels; resource `internalBorder`.

-w *number*

External border of number pixels. Also, `-bw` and `-borderwidth`; resource `externalBorder`.

-tn *termname*

This option specifies the name of the terminal type to be set in the TERM environment variable. This terminal type must exist in the termcap(5) database and should have `li` and `co` entries; resource `termName`.

-e *command [arguments]*

Run the command with its command-line arguments in the `iterm` window; also sets the window title and icon name to be the basename of the program being executed if neither `-title` (`-T`) nor `-n` are given on the command line. If this option is used, it must be the last on the command-line. If there is no `-e` option then the default is to run the program specified by the SHELL environment variable or, failing that, `sh(1)`.

-title *text*

Window title (`-T` still respected); the default title is the basename of the program specified after the `-e` option, if any, otherwise the application name; resource `title`.

-n *text*

Icon name; the default name is the basename of the program specified after the `-e` option, if any, otherwise the application name; resource `iconName`.

-C Capture system console messages.

RESOURCES (available also as long-options)

If compiled with internal Xresources support (i.e. `iterm -h` lists `.Xdefaults`) then `iterm` accepts application defaults set in `XAPPLLOADDIR/Rxvt` (compile-time defined: usually `/usr/lib/X11/app-defaults/Rxvt`) and resources set in `/.Xdefaults`, or `/.Xresources` if `/.Xdefaults` does not exist. Note that when

reading X resources, `iterm` recognizes two class names: `XTerm` and `Rxvt`. The class name `XTerm` allows resources common to both `rxvt` and `xterm` to be easily configured, while the class name `Rxvt` allows resources unique to `rxvt`, notably colours and key-handling, to be shared between different `rxvt` configurations. If no resources are specified, suitable defaults will be used. Command-line arguments can be used to override resource settings. The following resources are allowed:

fontspec: *fontspecificationfilename*

`iterm` uses several fonts. The fonts are for various Indian Scripts and for their display variations (like bold, italic etc.). The file containing the font names can be specified and provided by the user. The font specification (if) provided in the command line has the highest priority. The file can also be provided in the X-resource database which will be used in the case of absence of command line specifications. The "spec" file in the directory - `/usr/share/iterm` is the default font specification file in the absence of the above two; option `-fs`.

geometry: *geom*

Create the window with the specified X window geometry [default 80x24]; option `-geometry`.

background: *colour*

Use the specified colour as the window's background colour [default White]; option `-bg`.

foreground: *colour*

Use the specified colour as the window's foreground colour [default Black]; option `-fg`.

color*n:* *colour*

Use the specified colour for the colour value `n`, where 0-7 corresponds to low-intensity (normal) colours and 8-15 corresponds to high-intensity (bold = bright foreground, blink = bright background) colours. The canonical names are as follows: 0=black, 1=red, 2=green, 3=yellow, 4=blue, 5=magenta, 6=cyan, 7=white, but the actual colour names used are listed in the

COLORS AND GRAPHICS section.

colorBD: *colour*

Use the specified colour to display bold characters when the foreground colour is the default.

colorUL: *colour*

Use the specified colour to display underlined characters when the foreground colour is the default.

cursorColor: *colour*

Use the specified colour for the cursor. The default is to use the foreground colour; option **-cr**.

cursorColor2: *colour*

Use the specified colour for the colour of the cursor text. For this to take effect, `cursorColor` must also be specified. The default is to use the background colour.

reverseVideo: *boolean*

True: simulate reverse video by foreground and background colours; option **-rv**, False: regular screen colours [default]; option **+rv**. See note in COLORS AND GRAPHICS section.

inheritPixmap: *boolean*

True: make the background inherit the parent window's pixmap, giving artificial transparency. False: do not inherit the parent window's pixmap.

scrollColor: *colour*

Use the specified colour for the scrollbar [default B2B2B2].

troughColor: *colour*

Use the specified colour for the scrollbar's trough area [default 969696]. Only relevant for normal (non XTerm/NeXT) scrollbar.

backgroundPixmap: *file[;geom]*

Use the specified XPM file (note the `.xpm` extension is optional) for the background and also optionally specify its scaling with a geometry string `WxH+X+Y`, in which `"W"` / `"H"` specify the horizontal/vertical scale (percent) and `"X"` / `"Y"` locate the image centre (percent). A scale of 0 displays

the image with tiling. A scale of 1 displays the image without any scaling. A scale of 2 to 9 specifies an integer number of images in that direction. No image will be magnified beyond 10 times its original size. The maximum permitted scale is 1000. [default 0x0+50+50]

path: *path*

Specify the colon-delimited search path for finding files (XPM and menus), in addition to the paths specified by the RXVTPATH and PATH environment variables.

multichar_encoding: *mode*

Set the encoding mode to be used when multicharacter encoding is received; eucj: EUC Japanese encoding [default for Kanji]. sjis: Shift JIS encoding. big5: BIG5 encoding. gb: GB encoding. kr: EUC Korean encoding. noenc: no encoding; option -km.

greek_keyboard: *mode*

Set the Greek keyboard translation mode to be used; iso: ISO-8859 mapping (elot-928) [default]. ibm: IBM-437 mapping (DOS codepage 737); option -grk. Use Mode_switch to toggle keyboard input. For more details, see the distributed file README.greek.

selectstyle: *text*

Set mouse selection style to old which is 2.20, oldword which is xterm style with 2.20 old word selection, or anything else which gives xterm style selection.

title: *text*

Set window title string, the default title is the command-line specified after the -e option, if any, otherwise the application name; option -title.

iconName: *text*

Set the name used to label the window's icon or displayed in an icon manager window, it also sets the window's title unless it is explicitly set; option -n.

mapAlert: *boolean*

True: de-iconify (map) on receipt of a bell character. False: no de-iconify (map) on receipt of a bell character [default].

visualBell: *boolean*

True: use visual bell on receipt of a bell character; option -vb. False: no visual bell [default]; option +vb.

loginShell: *boolean*

True: start as a login shell by prepending a '-' to argv[0] of the shell; option -ls. False: start as a normal sub-shell

default

; option +ls.

utmpInhibit: *boolean* True: inhibit writing record into the system log file utmp; option -ut. False: write record into the system log file utmp [default]; option +ut.

print-pipe: *string*

Specify a command pipe for vt100 printer [default lpr(1)]. Use Print to initiate a screen dump to the printer and Ctrl-Print or Shift-Print to include the scrollback as well.

scrollBar: *boolean*

True: enable the scrollbar [default]; option -sb. False: disable the scrollbar; option +sb. Note that the scrollbar type (with/without arrows) is compile-time selected. **smallfont_key:** *keysym*

If enabled, use Alt-*keysym* to toggle to a smaller font [default Alt-<]

bigfont_key: *keysym*

If enabled, use Alt-*keysym* to toggle to a bigger font [default Alt->]

saveLines: *number*

Save number lines in the scrollback buffer [default 64]; option -sl.

internalBorder: *number*

Internal border of number pixels; option -b.

externalBorder: *number*

External border of number pixels; option -w, -bw, -borderwidth.

termName: *termname*

Specifies the terminal type name to be set in the TERM environment

variable; option -tn.

meta8: *boolean*

True: handle Meta (Alt) + keypress to set the 8th bit. False: handle Meta (Alt) + keypress as an escape prefix [default].

backspacekey: *string*

The string to send when the backspace key is pressed. If set to DEC or unset it will send Delete (code 127) or, if shifted, Backspace (code 8) - which can be reversed with the appropriate DEC private mode escape sequence.

deletekey: *string*

The string to send when the delete key (not the keypad delete key) is pressed. If unset it will send the sequence traditionally associated with the Execute key.

THE FORMAT OF THE FONT SPECIFICATION FILE

The font specification file contains several scripts related font information in the following format.

```
<Indian_Script_mnemonic>
Display_Attribute(s)_1 = Corresponding_font_name;
Display_Attribute(s)_2 = Corresponding_font_name;
.
.
.
Display_Attribute(s)_n = Corresponding_font_name ;

</Indian_Script_mnemonic>
```

Following mnemonics are used for Indian Scripts:

RMN: Roman

DEV: Devanagri

BNG: Bengali
TML: Tamil
TLG: Telugu
ASM: Assamese
ORI: Oriya
KND: Kannada
MLM: Malayalam
GJR: Gujarati
PNJ: Punjabi

The following mnemonics are used for the Display_Attribute_name:

BLD: Bold
ITA: Italic
UL: Underline
EXP: Expanded
HLT: Highlight
OTL: Outline
SHD: Shadow
NOR: Normal

Display Attributes can be combined together to give composite attributes.

Bold + Italic = BoldItalic

An example of the font specification file is:

```
<DEV>  
NOR = -altsys-dv_ttyogesh-medium-r-normal--20-0-50-50-p-0-iso8859-1;  
BLD = -altsys-dv_ttyogesh-bold-r-normal--20-0-50-50-p-0-iso8859-1;  
ITA = -altsys-dv_ttyogesh-medium-i-normal--20-0-50-50-p-0-iso8859-1;  
BLD ITA = -altsys-dv_ttyogesh-bold-i-normal--20-0-50-50-p-0-iso8859-1;  
</DEV>
```



```

<TML>
    NOR = -altsys-tm_ttvaluvar-medium-r-normal--20-0-50-50-p-0-iso8859-1;
    BLD = -altsys-tm_ttvaluvar-bold-r-normal--20-0-50-50-p-0-iso8859-1;
    ITA = -altsys-tm_ttvaluvar-medium-i-normal--20-0-50-50-p-0-iso8859-1;
    BLD ITA = -altsys-tm_ttvaluvar-bold-i-normal--20-0-50-50-p-0-iso8859-1;
</TML>

```

Here DEV is the mnemonic used for Devanagari Script. As shown above the begin and end tags for the scripts must be same except terminating symbol in the end tag. NOR is the Display Attribute for Normal fonts, Display Attributes "BLD ITA" together is for Bold+Italic text. "-altsys-dv_ttyogesh-medium-r-normal-20-0-50-50-p-0-iso8859-1" is the corresponding font name followed by a semicolon. The mnemonics for the Script and Display Attributes can be in mixed case.

THE SCROLLBAR

Lines of text that scroll off the top of the iterm window (resource: saveLines) and can be scrolled back using the scrollbar or by keystrokes. The normal iterm scrollbar has arrows and its behaviour is fairly intuitive. The xterm-scrollbar is without arrows and its behaviour mimics that of xterm

Scroll down with Button1 (xterm-scrollbar) or Shift-Next. Scroll up with Button3 (xterm-scrollbar) or Shift-Prior. Continuous scroll with Button2.

MOUSE REPORTING

To temporarily override mouse reporting, for either the scrollbar or the normal text selection/insertion, hold either the Shift or the Meta (Alt) key while performing the desired mouse action.

If mouse reporting mode is active, the normal scrollbar actions are disabled – on the assumption that we are using a fullscreen application. Instead, pressing Button1 and Button3 sends ESC[6 (Next) and ESC[5 (Prior), respectively. Similarly, clicking on the up and down arrows sends ESC[A (Up) and ESC[B (Down), respectively.

TEXT SELECTION AND INSERTION

The behaviour of text selection and insertion mechanism is similar to `xterm(1)`.

Selection:

Left click at the beginning of the region, drag to the end of the region and release; Right click to extend the marked region; Left double-click to select a word; Left triple-click to select the entire line. Insertion:

Pressing and releasing the Middle mouse button (or Shift-Insert) in an `iterm` window causes the current text selection to be inserted as if it had been typed on the keyboard.

LOGIN STAMP

`iterm` tries to write an entry into the `utmp(5)` file so that it can be seen via the `who(1)` command, and can accept messages. To allow this feature, `iterm` must be installed setuid root on some systems.

COLORS AND GRAPHICS

If graphics support was enabled at compile-time, `iterm` can be queried with ANSI escape sequences and can address individual pixels instead of text characters. Note the graphics support is still considered beta code.

In addition to the default foreground and background colours, `iterm` can display up to 16 colours (8 ANSI colours plus high-intensity bold/blink versions of the same). Here is a list of the colours with their `rgb.txt` names.

It is also possible to specify the colour values of foreground, background, `cursorColor`, `cursorColor2`, `colorBD`, `colorUL` as a number 0-15, as a convenient shorthand to reference the colour name of `color0-color15`. Note that `-rv` ("reverseVideo: True") simulates reverse video by always swapping the foreground/background colours. This is in contrast to `xterm(1)` where the colours are only swapped if they have not otherwise been specified. For example, would yield White on Black, while on `xterm(1)` it would yield Black on White. `iterm` sets the environment variables `TERM`, `COLORTERM` and

COLORFGBG. The environment variable WINDOWID is set to the X window id number of the iterm window and it also uses and sets the environment variable DISPLAY to specify which display terminal to use. iterm uses the environment variables RXVTPATH and PATH to find XPM files. System file for login records. Color names.

Written by Saumen Mandal. Report bugs to <saumen@cse.iitk.ac.in>.

Copyright 2001 by iLinux Software Products IIT Kanpur, All Rights Reserved.

Bibliography

- [1] Jeffrey D. Ullman Alfred Aho, Ravi Sethi. *Compilers - Principles, Technique and Tools*. Addison Wesley.
- [2] Nabajyoti Barkakati. *X Window System Programming*. Prentice Hall, second edition.
- [3] Cdac. World Wide Web,<http://www.cdac.org.in>.
- [4] Kterm, multilingual terminal emulator for x. World Wide Web,<http://packages.debian.org/unstable/x11/kterm.html>.
- [5] Tomohiro Kubota. Introduction to i18n. World Wide Web,<http://www.debian.org/doc/manuals/intro-i18n/>.
- [6] Modular-infotech. World Wide Web,<http://www.modular-infotech.com>.
- [7] Bureau of Indian Standards. *Indian Script Code for Information Interchange - ISCII Standard*. Manak Bhawan, 9, Bahadur Shah Zafar Marg, New Delhi, December, 1991.
- [8] Rxvt. World Wide Web,<http://www.rxvt.org>.
- [9] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison Wesley.
- [10] Tdil. World Wide Web,<http://www.tdil.gov.in>.
- [11] Xterm. World Wide Web,<http://www.cs.utk.edu/~shuford/terminal/xterm.html>.