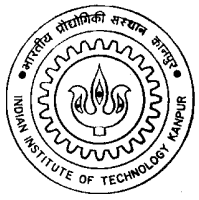
ISCII-HTML Document Support for Existing Internet Browsers

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Ashok Kumar Bhatt



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 "*ISCII-HTML Document Support for Existing Internet Browsers*", by *Ashok Kumar Bhatt*, 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

The Indian languages are non-linear in nature because of which there is no one-to-one correspondence between ISCII codes and emitted glyph codes. This is in contrast to ASCII encoded files. Because of this, ISCII encoded files are not meaningfully displayable using existing web-browsers.

To make it displayable, this tool converts ISCII encoded file to font encoded file on the fly. The conversion is done by an applet. Applet uses one configuration file for each encoding which specifies rules for converting ISCII codes to glyph codes. Since after conversion there is one-to-one correspondence between codes and glyphs to display, browsers can display it just like any other file.

Acknowledgements

I am greatly indebted to my thesis supervisor Dr. Rajat Moona for his constant support and valuable guidance throughout my work. In spite of his various involvements he was always available for help. His enthusiasm for all the work will always remain a motivating factor for me.

I acknowledge the MTech2000 batch for their exciting company. I would also like to thank ISCII group working under Dr. Rajat Moona for supporting me all the time, especially in last two hectic weeks. I wish I could express my thankfulness to all my friends for their love, support and encouragement.

I thank all the faculty and staff members of Department of Computer Science and Engineering at IIT, Kanpur for their technical help and support throughout my M.Tech. Programme.

I thank my parents, for their for their constant support , encouragement and patience with me. I am grateful to them for their efforts in building my career. Finally, I would also like to thank God for being kind to me.

Contents

List of Figures

Chapter 1

Introduction

In the past few years, the Internet revolution throughout the world, catalyzed largely by the World Wide Web (WWW), has enabled the information to be shared worldwide. However, much of this information is in English or in languages of Western origin. Presently, the Internet is positioned to be an international mechanism for communications and information exchange, the precursor of a global information superhighway. For this vision to be realized, one important requirement is to enable all languages to be technically transmissible via the Internet, so that in the absorbtion of Internet technology by any society, language poses no barriers.

India has a number of regional languages spoken in various parts of the country. And so, many people in India are currently excluded from actively participating in this so called information age just because of absense of softwares that can deal with information in the language which the majority of the Indians speak. This need was realized long back with Bureau of Indian Standards adopting ISCII(Indian Script Code for Information Interchange) as Indian Standard[4]. A lot of work has been going on since, but still we have to go a long way when it comes to information sharing on Internet.

1.1 Motivation

Presently, the approach for information sharing in Indian languages is to keep all the files encoded in specific fonts at the server end. But this approach fails if the client does not have the required font installed. This problem is also been taken care of by providing dynamic font downloading using pfr,*portable font represenation*. But, unfortunately it doesn't work with browsers running on a Linux system. Moreover, these methods do not exploit full capability of ISCII standard such as the transliteration of Indian languages. An ISCII coded file can be read in any Indian language. The motivation for this thesis was to develop a tool which enables ISCII-HTML browsing which does on the fly conversion from ISCII coded file to glyph coded file according to the fonts installed on client's end.

1.2 Related Works

One of the remarkable work done in this area is the development of MHTML(Multilingual HyperText Markup Language)[8]. Here the HTML files is converted to MHTML format which has all the font information embedded into it. On request an applet having the rendering engine is send along with the MHTML file. The benefit is that font is not required to be installed at client's end but MHTML tags are limited to a few tags in HTML. Tamilweb[6] a Bilingual Internet Information System, is developed by National University of Singapore which uses Tamil-English True Type Fonts developed specifically for the purpose.

1.3 Organisation of Thesis

In Chapter 2, we describe the peculiarities of Indian scripts. In Chapter 3, we describe our Configuration-driven Approach and Converter tool. In Chapter 4, we discuss design and implementation of our Applet. Chapter 5 gives the conclusion with the limitations and future directions.

Appendix A is user's manual. In Appendix B, we wrote a HowTo on writing configuration-file. In Appendix C, manual page for Converter Tool is written. In Appendix D, you will get a HowTo on installing true-type fonts in linux.

Chapter 2

Indian Languages: Addressing Problems

In this chapter we discuss the issues involved while working with Indian languages and the approach we have adopted for solving them.

2.1 Overview of Indian Languages

All Indian scripts are phonetic in nature. They all have common phonetic structure. But alphabet set of script and the rules for combining them are different. The character set of Indian scripts can be grouped into *vowels*,*consonants*,*matras*,*modifiers*,*numerals* and some other categories. These groups and their combining rules differ across the languages.

2.2 Indian Languages:Non-Linearity

The Indian languages are non-linear in nature. The most significant difference of these is that in English, each keystroke maps directly to a letter. So each letter can be coded uniquely. Contrary to that, in Indian langauges many characters can map to a single representation 'Syllable' of all constituent characters. Analogous to a letter in english, a *Syllable* is the Indian language equivalent unit of writing a letter.

There are too many of syllables to be encoded seperately.

A syllable is formed from the alphabets of the character set. There are certain rules by which these characters can be combined. The syntax for formation of a word is given in the following Backus-Naur Formation(BNF) [?].

```
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.
[]  elcloses items which are optional.
|   seperate items out of which only one can be present.
```

A syllable can have at the max four consonants. In the above syntax certain less frequently used syllables are ignored.

2.3 Design Issues

Each syllable has a unique visual representation. As there are so many of syllables, it's hard to design individual glyph for each of them. So a font generally contains different component glyphs, which when combined form a syllable glyph. Thus onscreen representation of a syllable is a composition of glyphs from the Indian language fonts.

There is no direct mapping of glyph codes to the consonant, vowel or modifier codes. However for every syllable (a sequence of consonant, vowels and modifiers) there is a corresponding sequence of glyphs. This forms many to many relationship between *character codes* and *glyph codes* as opposed to the one to one mapping in roman scripts.

2.4 Coding Standards for Indian Scripts

2.4.1 ISCII Standard

ISCII *Indian Standard Code for Information Interchange* is widely used for internal representation of Indian scripts. ISCII is an 8-bit encoding scheme while 7-bit ISCII is also available for the packages which don't allow the use of 8-bit codes. The lower 128 positions contains the ASCII character set, while upper 128 positions are used for indian script code. This allows roman characters to be used with Indain scripts.

ISCII codes has the advantage of spelling a word in phonetic order. This allows a document to be typed in the same way, regardless of the script it has to be displayed in, simplifying the transliteration process. This is one of the reason to keep document in ISCII codes instead of font-codes which is easily displayable. The detail description of ISCII standards can be found in [4].

2.4.2 Unicode Standard

Unicode provides a unique number for every character, no matter what the platform, no matter what the program,no matter what the language. Unicode is 16-bit encoding scheme. For all Indian scripts different code space isallocated unlike ISCII which is the superset of all Indian scripts.

2.5 Our Approach

To display multilingual ISCII-coded html files, we developed a tool which converts ISCII-coded files to font-coded file as soon as clients' browser send the request. The tool has two components: *Converter* and *htmlApplet* . Details about each component can be found in subsequent chapters.

Chapter 3

Design and Implementaion: Converter Tool

3.1 Configuration-Driven Approach

The problem encountered with any Indian Language is that there can't be any encoding scheme which can encode each syllable separately.This results in the problem of no one-to-one correspondence between *character codes* and the *glyph codes* to be displayed. This is in contrast to ASCII codes. That's what makes files encoded in any standard coding schemes for Indian Languages not meaningfully displayable using existing web-browsers.

To overcome with the problem of non-linearity of Indian-languages we use configuration-driven approach. Configuration files are written to deal with peculiarities of the languages. In general configuration file is the set of rules for transformation from one code space to another code space. We use ISCII encoding for Indian-Languages as our source code space and font-codes for the different standards as our target code space.

Using configuration file ISCII-coded file is converted to font-coded file, which contains index of the glyphs to be displayed, thus it has to be an one-to-one mapping. Configuration file differs from one language to another and from one font-encoding to

another, but the basic rules are similar across different font encodings. Configuration file is read by java parser class, generated from *javaCC*, a java parser generator. Rules for writing the configuration files are unambiguously defined. A HowTo on writing a configuration file may be found in Appendix B.

3.2 javaCC: The java Parser Generator

Java Compiler Compiler(javaCC)[9] is a top-down parser generator which reads LL(k) grammar specification and generates java programs which recognize matches to the grammar. All lexical specifications and grammar specification appears in a single file. The generated parser includes a public method declaration corresponding to each non-terminal symbol in the grammar file. Parsing with respect to a non-terminal is achieved by calling the method corresponding to that non-terminal.

3.3 Design and Implementation

Figure 3.1 depicts the design of the parser and converter used for the purpose of transforming ISCII coded files to font-coded files.

A parser suite is generated using Java Compiler Compiler, which contians the details about parser action. It uses ParserAction class to build the data-structures, which converter class needs to consult for conversion from source code space to target code space. Javacc generates *TokenManager.java*, *ParseException.java*, *Parser.java*,*Token.java* classes which reads in encoding file, and throws ParseException, if encoding file is not compliant to the rules written in javacc Lexical and Grammar Specification file.

Converter class takes a source code space string as input along with either the encoding file or fonts to be used for the conversion. It outputs target code space string. A manual page describing the details of **API's** available may be found in Appendix C.

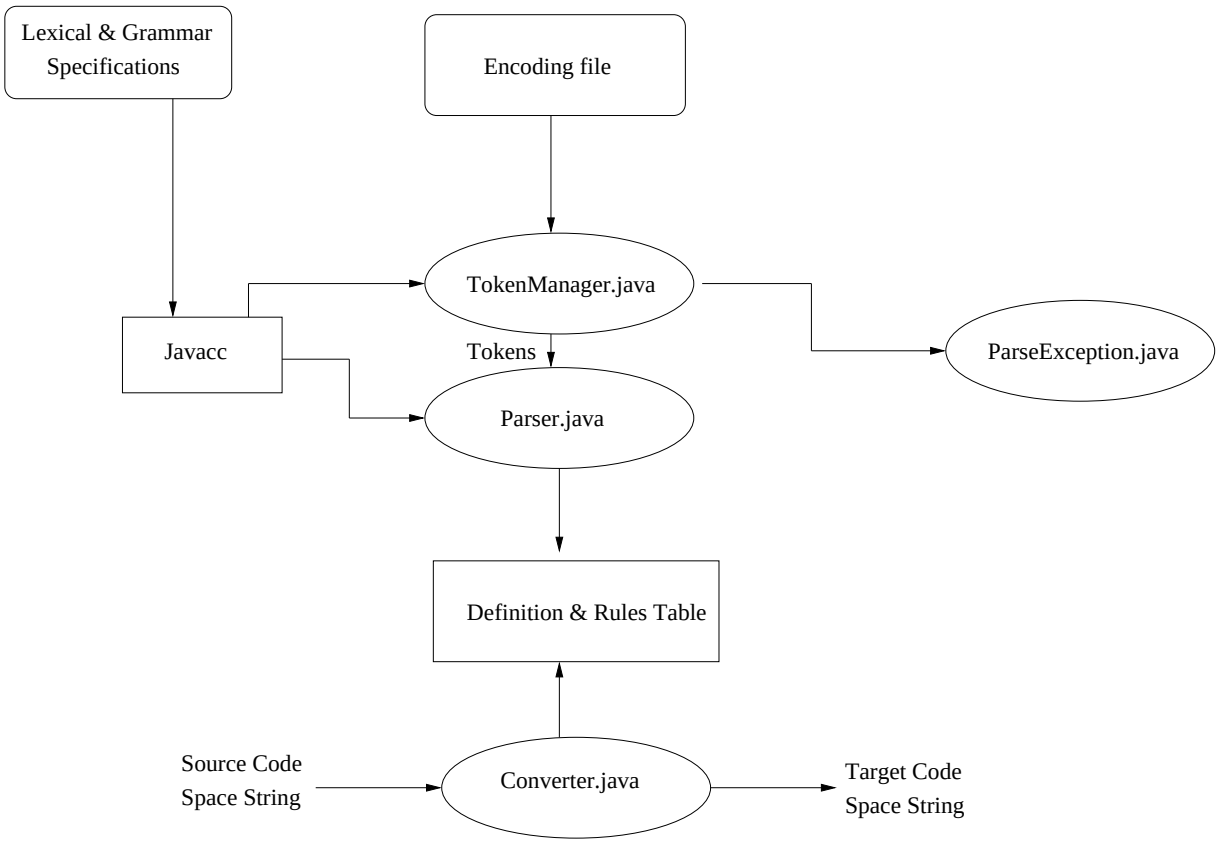


Figure 3.1: Design of the Converter Tool

Chapter 4

Design and Implementation: htmlApplet

In this chapter we discuss the design of Applet which does on-the-fly conversion.

4.1 Approach

We use applet to run on the client's machine and converts ISCII-coded file to font-coded file. One of the main reason to do conversion on the client's machine is to check for the availability of the required fonts. The flow of events is depicted in the Figure 4.1:

4.2 Implementation

As client's browser requests for ISCII-coded file, the embedded applet gets loaded onto the client's browser. We have written htmlApplet to do on the fly conversion. htmlApplet is a sub class of *JApplet*, an extension of applet from which java swing component can be called. Before JApplets gets loaded, it check for java plugin[1]. If java plugin is installed on the client's machine, JApplet get's loaded or else java plugin downloaded.

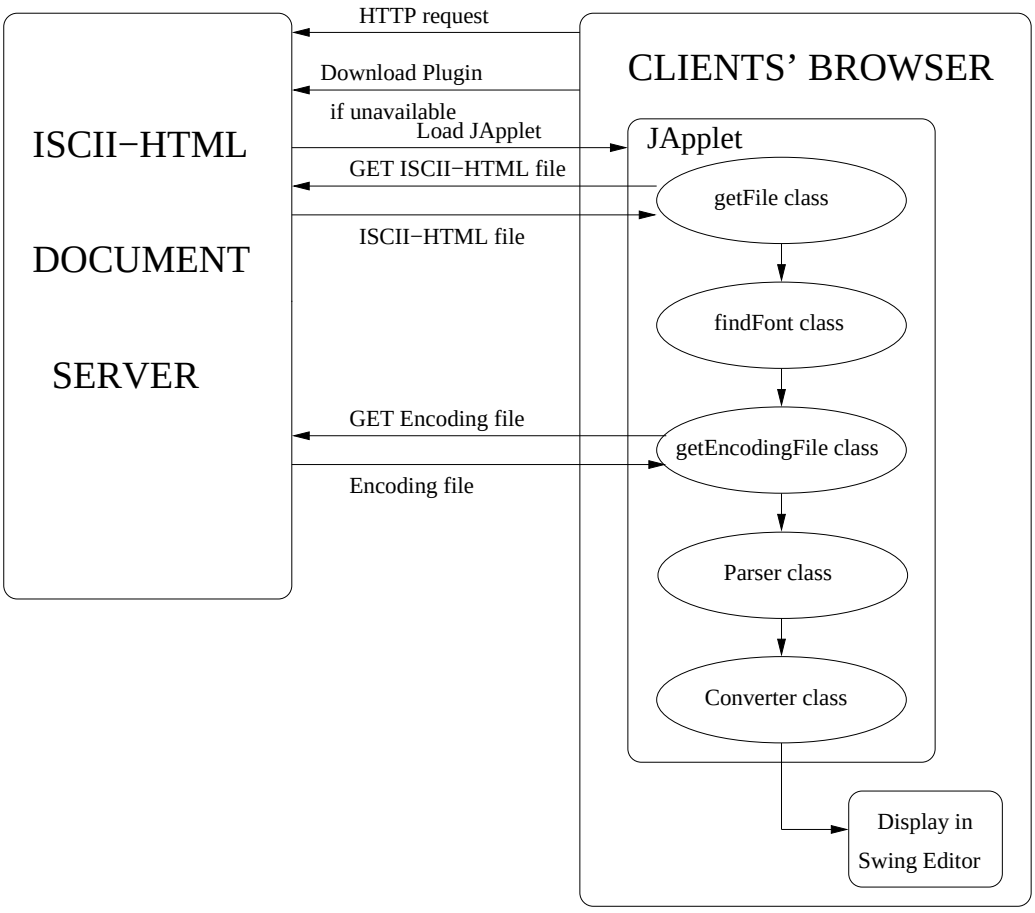


Figure 4.1: Flow of Events

htmlApplet on execution, checks for the *font face* tags in the html file. When it finds face attribute it checks for the fonts availabilitiy. If the fonts are available, htmlApplet sends back a request to the server through java.net.URL class to get the encoding file to do required conversion.

Once the conversion is done, htmlApplet creates an instance of Swing JEditorPane and gives the font-coded HTML stream for display.

Chapter 5

Conclusion and Further Work

In this work, we have shown that reading multilingual html-texts on a World Wide Web page from the Internet is indeed possible, even without any font change or additional helper tools on the client browser side. Even we can list a number of fonts in html file. If client have any of them installed, he'll be able to view the document. Right now we have written encoding files for *modular* and *isfoc* font standards for all Brahmi based Indian scripts. The server is also free from any extra load as all conversions are done on client side. This takes the toll by increasing download time as client needs to download java plugin, if it is not installed already.

Display of multilingual data alone is not enough. Users must be able to input search strings or fill in forms through the Web, and retrieve from multilingual databases. One solution is to make use of Java applets to control keyboard mappings of a different language and display the script as it is keyed in. This requires implementation of virtual keyboard.

Appendix A

User's Manual

This appendix explain about HowTo use this tool to enable ISCII-browsing. All the files can be copied to a common directory or it can be kept at different sub-directories. In case of sub-directories, name must be passed to applet using <param> tag.

A.1 Embedding Swing Applet in HTML file

A.1.1 Using htmlConverter

htmlConverter converts normal html file having applet tag to include *swing* applet. htmlConverter is available on
http://java.sun.com/products/jfc/tsc/articles/converter/converter.html
Example file to use *isciiweb*:

```
<html>
<body>
<applet code="htmlApplet.class" archive="isciiweb.jar" codebase="Appletdir/"
        width=500 heigth=20>
<param name="url" value="isciiFile.html">
</applet>
</body>
</html>
```

Where parameter *name="url"* specifies the ISCII coded html file to be displayed on client's browser. This file is passed to htmlConverter to include swing applet. After conversion example file will be as shown below:

```
<html>
<body>
<!--"CONVERTED_APPLET"-->
<!-- CONVERTER VERSION 1.0 -->
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 500  codebase="http://java.sun.com/products/plugin/1.1.1/jinstall-111-win32.c
<PARAM NAME = CODE VALUE = "htmlApplet.class" >
<PARAM NAME = CODEBASE VALUE = "Appletdir/" >
<PARAM NAME = ARCHIVE VALUE = "isciiweb.jar" >
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
<PARAM NAME = "url" VALUE ="isciiFile.html">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2" java_CODE = "htmlApplet.class" ja
</NOEMBED></EMBED>
</OBJECT>
<!--
<APPLET  CODE = "htmlApplet.class" CODEBASE = "Appletdir/" ARCHIVE = "isciiweb.jar" W
<PARAM NAME = "url" VALUE ="isciiFile.html">
</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->
</body>
</html>
```

A.1.2 Manually Changing the File

html-File to include Swing applet can also be written manually. A exapmle file is shown below:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
```

```
width="500" height="20"
codebase="http://java.sun.com/products/plugin/1.2/
jinstall-12-win32.cab #Version=1,1,2,0">
<PARAM NAME="code" VALUE="htmlApplet.class">
<PARAM NAME = ARCHIVE VALUE = "isciiweb.jar" >
<PARAM NAME="codebase" VALUE="Appletdir/">
<PARAM NAME = "url" VALUE ="isciiFile.html">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2"
width="500" height="20"
code="htmlApplet.class"
archive="isciiweb.jar"
codebase="Appletdir/"
url="isciiFile.html"
pluginspage="http://java.sun.com/products/plugin/
1.2/plugin-install.html">
<NOEMBED>
</COMMENT>
</NOEMBED>
</EMBED>
</OBJECT>
```

where parameters *name="code"* is used to include htmlApplet, *name="archive"* is used for applet jar file and *name="codebase"* gives the relative path of applet class files. Parameter *name="url"* is the ISCII coded html file to be displayed on client's browser.

Appendix B

Howto Write Configuration File

The configuration file is just the set of rules for converting one set of codes to another set of codes. Depending upon the transformation needed, configuration file contains from very simple rules to complex rules. Configuration file can be divided in three set of statements: definition, grouping rules and rewriting rules.

Source code space is represented by placing a # symbol before the number, target space numbers are preceded by % symbol. The numbers can be written in decimal, hexadecimal and octal number systems using C language format.

B.1 Definitons

Definitions are just there to give symbolic name to source and target code space to improve readability of theconfiguration file. For example,we may have two definitions as given below:

```
LA = #0xd1;
G_LA = %0x61 %0x41 ;
```

which means LA is the ISCII code d1h and G_LA is the glyphs 61h and 41h.

B.2 Grouping Rules

Grouping rules specify the various categories in the source and the target code space. For example, Indian language character set has *consonant*,*vowel*,*modifiers* and other categories. Grouping the elements reduces the number of rules needed as all the codes on which common rules can be applied are place in a group. Each group consists of one or more constituent elements (identifiers or numbers), seperated by | (the vertical bar). For example, in the configuration we may have a grouping rule like the one given below:

```
VOWELS =  A | AA | I | II | #0xa8;
```

which tells that A, AA, I, II and #0xa8 are in a single category VOWELS.

A code space number may belong to several different category depending upon the rules it share with other code space numbers. 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 defintion or grouping statement.

B.3 Rewriting Rules

Rewriting rules do the actual transformation from one code space to another code space. 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. It simply states that whenever a pattern matching with left hand side pattern appears in input string, replace it with right hand side patten.

The rules are mathched from top to bottom as it appears in configuration file. The rules are applied recursively which gives us the freedom to use source space numbers on the right hand side of the rewriting rules. The converter reads the configuration file, matches the pattern of input string with left hand side of the rules in order as they appears in configuration file, and if a rule matches, it replace

the input pattern with right hand side of the rule. This is repeated untill input string contains all target codes.

Now we'll explain one example to illustrate conversion method.

```
CONSONANT HALANT CONSONANT MODIFIER -> $1  $2  $3 G_MODIFIER($4) ;
CONSONANT HALANT CONSONANT -> G_HALF_CONSONANT($1) $3;
```

The $n in right hand side means the n-th symbol in the left hand side of the rule. Referring to the first rule, right hand side $1 $2 $3 says whenever input pattern matches with this rule, copy 1st, 2nd anf third symbol from input pattern as it is to the output pattern. And replace fourth symbol with corresponding number from G_MODIFIER group. Now since three code space numbers are as it it copied, rule matching will start from that point onwards.

Some intermediate codes and be used when those numbers can also belong to other groups. For example,

```
KA HALANT HARD_SHA              -> #2000;
SHA HALANT RA                   -> #2001;
```

Now these number can be placed in any category where they share the common set of rules.

Appendix C

Manual Page for Converter Tool

NAME
doConversionByEncoding, doConversionByFont, GetEncodingFile - java classes for handling ISCII data
SYNOPSIS

public static int doConversionByEncoding(char isciiString[], int start, int isciiLength, char glyphString[], IntObject glyphLength , String encodingFile, IntObject errorCode)

public static int doConversionByFont(char isciiString[], int start, int isciiLength, String sourceFormat, char glyphString[], IntObject glyphLength, String fontName, IntObject errorCode)

public static String GetEncodingFile(String fontName,String sourceFormat) throws FileNotFoundException

DESCRIPTION

These are the java methods defined in the class Conversion. **doConversionByEncoding()** converts an source string pointed by isciiString to an

array of target codes pointed by glyphString according to the encoding name encodingFile. The size of the source string is provided in the integer value isciiLength. The size of the target array is provided in the IntObject glyphLength and this size should be large enough to accomodate all the target codes generated from the isciiString. Upon return, glyphLength is set to the number of target codes translated into the glyphString. The return value of glyphLength 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 encoding file is searched in the path /usr/lib/X11/fonts/encodings; /usr/share/fonts/encodings; /etc/X11/fs/encodings; $HOME/encodings in order. 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 error Code returns the error number. doConversionByFont() is similiar, except that the name of the font fontName is passed instead of the encoding.Besides, the format of the source encoding(ISCII, UNICODE or UTF-8). This function uses **GetEncodingFile** method to retrieve the corresponding encoding associated with the fontName. This java method is defined in the class parseFonttable. GetEncodingFile returns the name of the encoding file corresponding to String fontName 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:

```
*dv_tt*          ISCII_ISFOC_DEV
*as_tt*          ISCII_ISFOC_ASM
*ml_tt*          ISCII_ISFOC_MLM
*pn_tt*          ISCII_ISFOC_PNJ
```

The fonttable is searched in the path /usr/lib/X11/fonts; /usr/share/fonts; /etc/X11/fs; $HOME in order.

RETURN VALUE

doConversionByEncoding() and **doConversionByFont()** returns the index of first unconverted byte in isciiString. It returns 0 if nothing could be converted and isciiLength if the entire string got converted.

GetEncodingFile() returns reference to the encoding file if it could find corresponding encoding for the font fontName and null if the fonttable could not be located. It throws *FileNotFoundException* if encoding file couldn't be located.

Appendix D

Howto install true-type fonts in Linux

D.1 Installing True-Type Fonts Using Font-Server

There are two ways to add true-type fonts. One can use Xserver with true type support or alternatively a Font server with true-type support can be installed and configured to serve fonts. If one decide to use font-server to serve the true-type fonts, following is HOWTO install and configure font-server.

D.1.1 HowTo Use Font-Server

- Install package Xfsft (or Xfs):
  This package contains xfs font server and various utilities to support the fonts. Once xfs font server has been installed some changes may be needed.

- Install utility ttmkfdir:
  This utility is used to create fonts.scale file for TrueType fonts. This file in each font directory is used by xfs to find out available fonts in that directory.

- Configure X server to use font server
  Now we need to configure X server so that it uses font server for getting fonts.

For this purpose the XF86Config file is modified to include the font server details in the fontpath.

```
# default FontPath: misc, Speedo, Type1, 75dpi, 100dpi
#  FontPath   "/usr/X11R6/lib/X11/fonts/misc:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/75local:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/75dpi:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/100local:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/100dpi:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/TrueType"
#  FontPath   "/usr/X11R6/lib/X11/fonts/Type1"
#  FontPath   "/usr/X11R6/lib/X11/fonts/Speedo"
#  FontPath   "/usr/X11R6/lib/X11/fonts/iso_8859.2/misc:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/iso_8859.2/75dpi:unscaled"
#  FontPath   "/usr/X11R6/lib/X11/fonts/iso_8859.2/100dpi:unscaled"
   FontPath   FONTSERVER NAME.
```

In this example only font server is in font path. We can include additional fontpath where xserver will look for the fonts.

FONTSREVER Name:
One of the following forms can be used to name a font server that accepts TCP connections:

```
tcp/hostname:port
tcp/hostname:port/cataloguelist
```

The port is the decimal TCP port on which the font server is listening for connections. The catalogue list specifies a list of catalogue names, with '+' as a separator.

```
Examples:
       tcp/csews15.cse.iitk.ac.in:7000, tcp/172.31.18.15:7001/all.
```

```
Other way to specify FontServer Name is
unix:/-1
unix:/7100
```

This creates the socket at the filesystem level, and is perhaps a bit faster than TCP/IP, and definitely more secure. These are implementation specific and this fontserver can serve fonts only on local machine.The form that is used must be the one on which the xfs is configured to listen for the font requests.

- Configure font server
  You will need to find the way the font directories are specified to the X font server. We describe one such method in which the font server looks at the font directories in /usr/X11R6/lib/X11/fs/config directory. Now to add previous fontpaths Modify catalogue in /usr/X11R6/lib/X11/fs/config to include all previous paths.

```
# where to look for fonts
catalogue = /usr/X11R6/lib/X11/fonts/misc:unscaled,
  /usr/X11R6/lib/X11/fonts/75local:unscaled,
  /usr/X11R6/lib/X11/fonts/75dpi:unscaled,
  /usr/X11R6/lib/X11/fonts/100local:unscaled,
  /usr/X11R6/lib/X11/fonts/100dpi:unscaled,
  /usr/X11R6/lib/X11/fonts/TrueType,
  /usr/X11R6/lib/X11/fonts/Type1,
  /usr/X11R6/lib/X11/fonts/Speedo,
  /usr/X11R6/lib/X11/fonts/iso_8859.2/misc:unscaled,
  /usr/X11R6/lib/X11/fonts/iso_8859.2/75dpi:unscaled,
  /usr/X11R6/lib/X11/fonts/iso_8859.2/100dpi:unscaled
```

This specifies the directories where font files are residing. Font server searches these directories to find out different fonts.

- Restart X server with font server

  As X server needs a font server running before X server starts (because of FontPath definition), we can either:

  - Manualy start xfs before X server
  - Start xfs at system startup (/etc/rc.local, ...)
  - Start xfs before X server in /usr/X11R6/lib/xinit/xserverrc (best way):

```
# start X (TrueType) font server
killall xfs 2>/dev/null
xfs &
# start X server
exec /usr/X11R6/bin/X -dpi 100 -deferglyphs all :0
```

D.1.2 Adding Scalable True-Type Fonts To Font-Server

These instructions will make font-server to serve true-type fonts.

- Make a directory for storing fonts. In this discussion we assume that the directory is $HOME/fonts.
- Put the ttf fonts in $HOME/fonts.
- Change current directory to $HOME/fonts.
- Run ttmkfdir -c > fonts.dir and

  ttmkfdir -c > fonts.scale.

  This will create 'fonts.scale' and 'fonts.dir' files in the directory.
- Now run the following commands to tell the font-server the location of your font files.

  chkfontpath –add $HOME/fonts.

  This will add font-path to the file /usr/X11R6/lib/X11/fs/config.

Restart font-server.

/etc/rc.d/init.d/xfs restart

Now tell XServer to rebuild it's font-paths

xset fp rehash

- The fonts are installed. You can now start netscape and check whether the fonts are visible in the list when you click edit-> preferences->fonts.
- To see the list of fonts served by font-server use the command

  fslsfonts -server FONTSERVER NAME e.g. fslsfonts -server unix/:-1.

  or use

  xlsfonts.

D.2 Installing True-Type Fonts Without Font Server

If you don't want to use Font-server, following are the instructions to install true-type fonts. Only XFree86-4.x and higher has support for true-type fonts. To check which version you are using currently type the command

rpm -qa | grep XFree.

If we have root permissions then the following steps will install fonts:

- Make a directory for storing fonts. say /usr/X11R6/lib/X11/fonts/ttfonts.
  mkdir /usr/X11R6/lib/X11/fonts/ttfonts
- Put all the ttf fonts you want to install, in that directory.
- Change current directory to /usr/X11R6/lib/X11/fonts/ttfonts.
  cd /usr/X11R6/lib/X11/fonts/ttfonts
- Run ttmkfdir -c > fonts.dir and

  ttmkfdir -c > fonts.scale.

This will create 'fonts.scale' and 'fonts.dir' files in the directory.

- Now run the following commands to tell the XServer the location of your font files.

  /usr/X11R6/bin/xset fp+ /usr/X11R6/lib/X11/fonts/ttfonts.

  This will inform the XServer about the location of your fonts. Then run

  /usr/X11R6/bin/xset fp rehash

  This will make the XServer rebuild its tables so that the new fonts are included in them.
  Above step needs to be done everytime server get started. The other way of doing it is to add the fontpath to the file /etc/XF86Config by adding the line FontPath = /usr/X11R6/lib/X11/fonts/ttfonts.

- The fonts are installed. You can now start netscape and check whether the fonts are visible in the list when you click edit-> preferences->fonts.
  U might not get installed fonts in netscape font list.This is because of encoding used. Just change edit->preferences-> fonts and define the encoding to be 'user-Defined'. Then select View->Character Set->User-Defined.

If the permissions are of user-level only some of the steps need to be changed. The following are the steps to install

- Make a directory for storing fonts.
  mkdir HOME/fonts.

- Put all the ttf fonts you want to install, in that directory.

- Change current directory to $HOME/fonts.

- Run ttmkfdir > fonts.dir and

  ttmkfdir > fonts.scale.

  This will create 'fonts.scale' and 'fonts.dir' files in the directory.

- Now run the following commands to tell the XServer the location of your font files.

  /usr/X11R6/bin/xset fp+ $HOME/fonts

  This will inform the XServer about the location of your fonts. Then run

  /usr/X11R6/bin/xset fp rehash

  This will make the XServer rebuild its tables so that the new fonts are included in them. Above step needs to be done everytime server get started. The other way of doing it is to add the above commands in a startup file such as .xinitrc or .bashrc or .bash_profile.

- The fonts are installed. You can now start netscape and check whether the fonts are visible in the list when you click edit-> preferences->fonts.
  You might not get installed fonts in netscape font list.This is because of encoding used. Just change edit->preferences-> fonts and define the encoding to be 'user-Defined'.

Bibliography

[1] Mark Andrews. Java plugin. World Wide Web,http://java.sun.com/products/jfc/tsc/articles/plugin/.

[2] Mary Campione. *The Java Tutorial*. Addison Wesley, Longman, 2nd edition, 1999.

[3] Modular-infotech. World Wide Web,http://www.modular-infotech.com.

[4] Bureau of Indian Standards. *Indian Script Code for Information Interchange - ISCII Standard*. Manak Bhawan, 9, Bahadur Shah Zafar Marg, New Delhi, December, 1991.

[5] ronald@innovation.ch. Ronald Tschal. Displaying html. World Wide Web,http://www.innovation.ch/java/HTTPClient/disp_html.html.

[6] tanilweb Singapore. Tamil on the web. World Wide Web,http://irdu.nus.edu.sg/tamilweb/.

[7] Tdil. World Wide Web,http://www.tdil.gov.in.

[8] Ibaraki Japan ULIS, Tsukuba. Multilingual html. World Wide Web,http://mhtml.ulis.ac.jp/people.html.

[9] Webgain. Webgain products: Javacc. World Wide Web,http://www.webgain.com/products/java_cc/.