IMPROVING TEXT CONVERSION UTILITES

Summer Research Intership - 2013
Mentor : Prof. Dr Sanjay Chaudhary

By Jaydeep Marvaniya
201001132

Background:

Unicode is accomplishing need of encoding standard needed for information interchange among various computing and rendering device. But prior before the Unicode there exists various encoding standard and fonts system especially for regional languages and due to lack of common standard among font developers there are tons of documents which are not possible to process computationally or to render in unsupported devices. This report depicts development of system for encoding and font conversion utilities developed to interchange from MS-DOS supported KRISHNA font to Unicode encoding scheme supported SHRUTI font. Unicode is commonly available scheme for encoding of electronic documents.

Keywords:

Information-interchange, encoding standards, Font Systems, Rendering, Software development,

Introduction:

There were many different fonts and encoding styles used in electronic publishing in regional languages. Also due to rise of web technologies many online systems are available for electronic publishing the primary requirement for any Publishing software to develop proficient publishing is a document established on highly reliable and largely supported platforms. A huge amount of typed content prepared in localized electronic publications of different regional languages is not being utilized due to technical obstacles like various independent versions of fonts, compatibility issues of 2-decade previously typed e-contents with latest and consistently advancing technologies. So in order to make this content available to process for current encoding and rendering standards we need to convert them to such system which is widely supported by major electronic publishing systems and other software platforms.[1] For example, Gujarati alone has a variety of randomly prepared MS-DOS based fonts like Krishna, Avantika, Gopika, Ghanshyam, Saral and many more. It is almost impossible to efficiently interchange the information generated using one type of font to other font of the same language due to lack of standardization. These languages have many literary documents including e-books, articles and other works typed in MS-DOS based fonts. Such useful documents are under-utilized and unavailable to a major part of the population due to incompatibility of the fonts with latest operating systems. To understand the problem clearly we have to understand the role of encoding and fonts in rendering, editing and in storage of electronic document.

Role of encoding:

Every font has a character set that it renders and for each of this character there is a byte code being used for digital manipulation and processing. For computer It does not make sense to have a string without knowing what encoding it uses. Different computer may use different encoding. But for the electronic publishing purpose we have to have a standard encoding in order to make document portable over system. The role of encoding comes in plays when dealing with different types of fonts.

Role of fonts:

Here it is necessary to understand role of font in publishing and its connection with typeface. The set of characters containing letters, numbers, punctuation marks, glyphs and special symbols of the same design is called typeface while specific style, pitch and size of typeface is known as fonts.

Development of a font (or font style) require minimum a typeface and an encoding to represent it in digital format. Each publication used to prepare their own fonts for particular language and typed their content in those fonts and so there were no common encoding. So if your device have one kind of font for a language then it is not necessary that you can read e-contents of other fonts of the same language due to the different character sets are encoded differently for each of those fonts. Many articles typed in MS-DOS based fonts become inappropriate in latest operating systems of electronics devices like Windows 7, XP, and Windows-8 etc. So standardization of font's encoding became essential for these languages as features like information interchange and searchable e-content are not possible without conversion of fonts. From here by 'conversion of font' we mean the conversion of unsupported font’s underlying encoding (font-dependent) to device supported encoding-font pair. Our target encoding is Unicode due to reason separately outlined below.

Need of common character set and encoding standard- 'Unicode'

Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.[3] The Unicode® Consortium developed an industry standard character set encoding named 'Unicode' with an aim to have a Universal Character Set (UCS) that supports all characters from worldwide scripts and symbols; that are in common use worldwide today or in the past and scope to add characters in future. The UCS has the capacity can support over 1,000,000 characters and presently around 96,000 characters are being supported to represent a large number of scripts. The benefits of a single UCS and implementation of Unicode have made it a dominant and omnipresent standard.

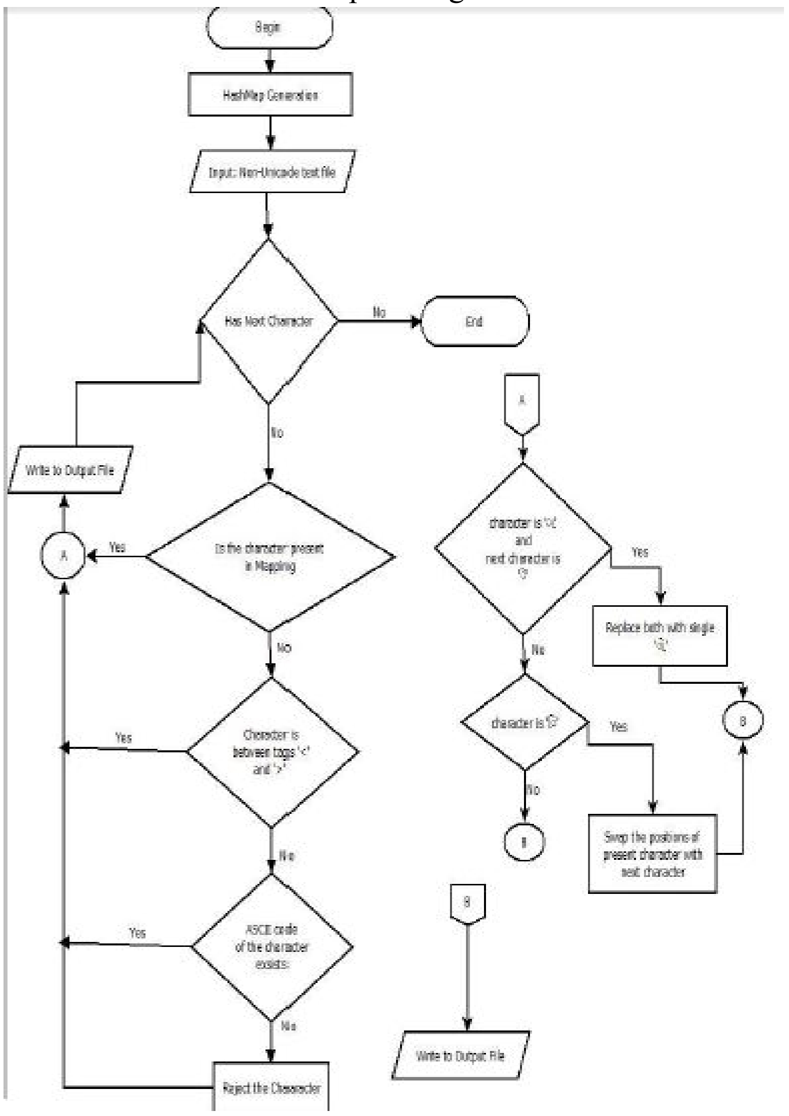
Overview of A text conversion utility

The main aim of the utility is to change different encoding of input document to a common encoding, Unicode and thus making it available for rendering in Unicode supported font of that particular regional language. For test cases and example documents in Gujarati language typed with Krishna font is used.

Logic :

- The input file was in ANSI encoding so it has to be decoded with the same format using "ISO8859-1" encoding format supported by java.
- There are two types of character mapping that are stored in two different HashMaps: One of them stores the Krishna(in ANSI format) to ASCII mapping and the other one stores ASCII to Shruti(in UTF-8 format) mapping.
- Now the input file is read character by character using iteration, the ASCII code of the read character is obtained through HashMap where the mapping has been stored as (key, value) pairs.
- Our next step is to find the character (in Shruti) in the second HashMap corresponding to the ASCII value obtained from the above.

- This character is then written to the output file generated in UTF-8 format.


Begin
HashMap Generation
Input: Non-Unicode text file
Has Next Character
No
End
No
Write to Output File
A
Yes
Is the character present in Mapping
No
Yes
Character is between tags '<' and '>'
No
Yes
ASCII code of the character exists
No
Reject the Character
A
character is and next character is
Yes
Replace both with single
No
character is
Yes
B
No
B
Swap the positions of present character with next character
B
Write to Output File
Figure 1: Flowchart of a convesion logic.

Challenges:

Types of error and reasons for those errors:

The aim of this report is to show efforts done to improve the quality of above conversion. The main reason for less quality of such conversion is variety of Independent Vowel Letters, Dependent Vowel Signs, Live Consonants, Dead Consonants, and Half-Consonant Forms, which need to be concatenated properly with logic so as to render the text tidily on the screen.

Due to inconsistency in unicode codepoint and given font-table's chractors:

There may be inconsistency in unicode codepoint system and given font-table's chracters. Some chracterters may br wrongly mapped in table. This type of errors are left for spell checking and grammer correction.

Due to incorrect position of some dependent character:

With conversion of encoding we can able to render the correct dependent characters such as Independent Vowel Letters, Dependent Vowel Signs, Live Consonants, Dead Consonants, and Half-Consonant Forms but due to Indian regional language's wide variety of glyphs there is need of correct position of dependent Vowel signs and half consonant forms as the final word is made of joining this dependent character with corresponding independent character in any rendering system. The only solution is reconstruction of such words that contains dependent characters. There two types of reconstruction.Implicit Reconstruction of the output takes place abiding the Unicode Rendering Rules. And explicit reconstruction that is achieved manually.

Need of Explicit Reconstruction of word

To solve above error we have to reconstruct the above error containing words. There may be many such words that contains above characters. There are two types of rearranging needed; there are some dependent characters which need to be place after their successive character or characters depending upon input font style and that character. The other rearranging is one in which some dependent characters on detection while parsing need to put ahead of the previously written character or characters again depending upon font and the dependent character detected. . Former being easy to hold and write needed, later is challenging task when parsed chracter by chracter and written simentensiouly as we dont know in advance which upcoming depdent chractor we have to write before writing currently detected chracter.. So instead of detecting and writing one by one the following reconstruction algorithm is suggested and implemented for words to increase overall quality of conversion.

New Algorithms for reconstruction of words

To reconstruct words we have to hold dependent chractor of interest and put it before their previously detected independent chractor or after the successive independent chractor. The First case is relativly easy and one can hold the detected dependent chractor untill other successive chractor are written. This is achievably by detecting chractors one by one and writing it simeltenisiously. But the counter case where need to write detected dependent chractor before its previously detected chractor can't be done when being processed one by one. So this new algorithm for reconstruction of words is made by processing four chractor togather. Four chractor are detected and on comparing with font table necessory rules are applied to rearrage them if incorrect postion of dependent charactors are detected.

Logic for explicit reconstruction :

First case:

(i) if the present character is "ि" swap the position of the present character with the next character;

(ii) if the present character is "□" and the next character is "□" replace both characters with "□". It

Second case:

- Four characters are processed together to check if set of them satisfy any reconstruction criterion.
- If any condition is matched than it is reconstructed.
- And reconstructed word is written in output file.
- If any condition is not matched than the first character is written in output file .

- Then again next four characters are processed.

While parsing more than one chractor and putting dependent chratar after their successive independent chractor one need to keep this in mind that once rearranging is done, while detecting others one have to move ahead that much number of chractors which has been written already, else the written depedent word would detected another time and condition would be applyed more than necessory time resulting in completly incorrect word.

Packaging and distribution of conversion utility

Prior to this development work the programme was a java executable file. For graphical user interface Swing Graphics library in java is used. User could be able to choose input file and that file is being converted and output file is being written in same place as input file. In order to achieve this requirement JButton and JFilechooser utility of Swing Graphics library is used.[2]

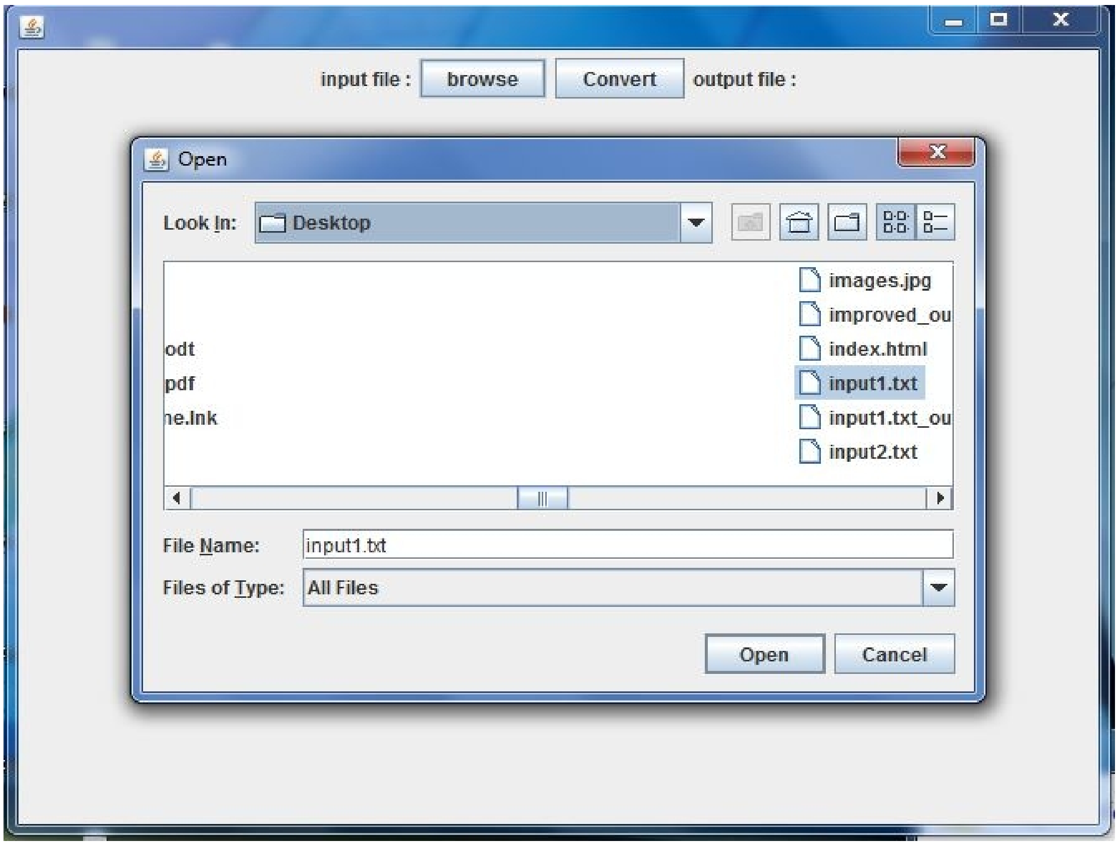
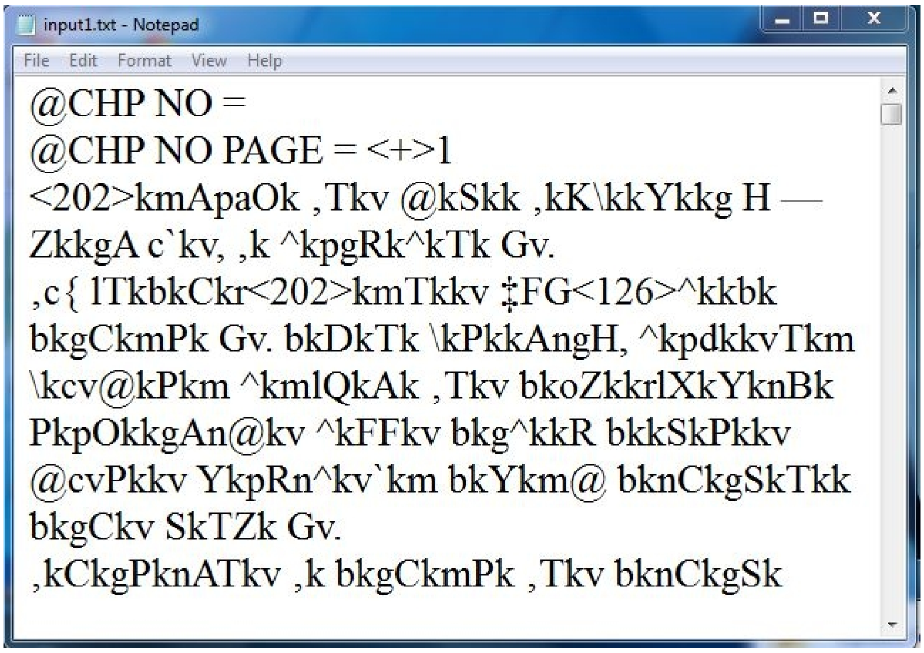
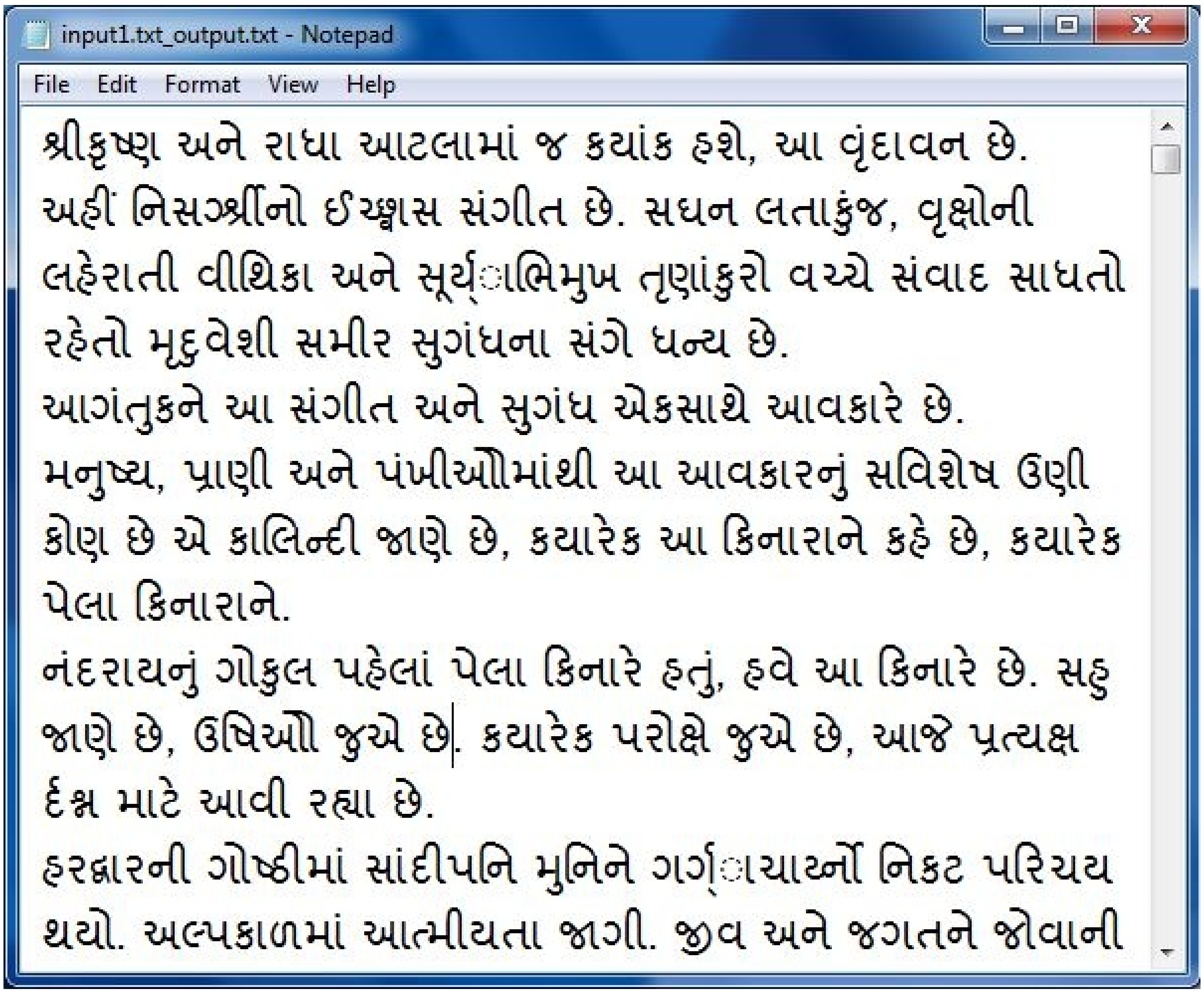


Figure 2: Graphical User Interface of the conversion tool.



```
@CHP NO =
@CHP NO PAGE = <+>1
<202>kmApaOk ,Tkv @kSkk ,kK\kkYkkg H —
ZkkgA c`kv, ,k ^kpgRk^kTk Gv.
,c{ 1TkbkCkr<202>kmTkkv ‡FG<126>^kkbk
bkgCkmPk Gv. bkDkTk \kPkkAngH, ^kpdkkvTkm
\kcv@kPkm ^kmlQkAk ,Tkv bkoZkkrlXkYknBk
PkpOkkgAn@kv ^kFFkv bkg^kkR bkkSkPkkv
@cvPkkv YkpRn^kv`km bkYkm@ bknCkgSkTkk
bkgCkv SkTZk Gv.
,kCkgPknATkv ,k bkgCkmPk ,Tkv bknCkgSk
```

Figure 3: sample input file, input1.txt in krishna font(below)



```
શ્રીકૃષ્ણ અને રાધા આટલામાં જ કયાંક હશે, આ વૃંદાવન છે.
અહીં નિસર્ગનો ઈચ્છ્વાસ સંગીત છે. સઘન લતાકુંજ, વૃક્ષોની
લહેરાતી વીથિકા અને સૂર્ય્ાભિમુખ તૃણાંકુરો વચ્ચે સંવાદ સાધતો
રહેતો મૃદુવેશી સમીર સુગંધના સંગે ધન્ય છે.
આગંતુકને આ સંગીત અને સુગંધ એકસાથે આવકારે છે.
મનુષ્ય, પ્રાણી અને પંખીઓમાંથી આ આવકારનું સવિશેષ ઉણી
કોણ છે એ કાલિન્દી જાણે છે, કયારેક આ કિનારાને કહે છે, કયારેક
પેલા કિનારાને.
નંદરાયનું ગોકુલ પહેલાં પેલા કિનારે હતું, હવે આ કિનારે છે. સહુ
જાણે છે, ઉષિઓ જુએ છે. કયારેક પરોક્ષે જુએ છે, આજે પ્રત્યક્ષ
દૈશ્ન માટે આવી રહ્યા છે.
હરદ્વારની ગોષ્ઠીમાં સાંદીપનિ મુનિને ગર્ગ્ાચાર્યનો નિકટ પરિચય
થયો. અલ્પકાળમાં આત્મીયતા જાગી. જીવ અને જગતને જોવાની
```

Figure 4: Output file after conversion to unicode

Challenges:

The above developed system works satisfactorily except few characters that one have to remove manually. The programmatic conversion of such words is challenging task and new rules needed as with every such dependent character whose position in newly constructed word is not known.

The software has been prepared keeping in mind that Hash Maps are created and stored only once for a particular font mapping. Running the program for the first time will ask you to provide the mapping and from next time onwards the program works with the stored mapping. One can easily add font mapping just for once in the code and the output could be achieved in the desired font.

Future work:

From here one should go for expanding this project for multiple fonts. Graphical Interface could be developed to convert folders containing input file instead of individual files.

The future aspects of this project could be to extend this tool by adding the Gujarati lexical analyser for spell check and grammar to correct remaining dependent characters which are not being detected while parsing. Further, once these old literary works are translated to Unicode, they can be converted to .epub or any other e-book document format so that they can be read as an e-book. Thereafter searching for a word could also be made possible in the e-pub document by importing the Gujarati font files and the lexical analyser as well for even better reading and formatting experience.

Reference :

[1] Sean Pue, *E-Journal of the South Asia Language Resource Center*, The University of Chicago http://salpat.uchicago.edu/index.php/salpat/article/view/33/29
Article on *'Desktop Publishing in the 21st Century'*

[2] The API specification for version 6 of the *Java™ Platform, Standard Edition*
http://docs.oracle.com/javase/6/docs/api/

[3] E-book ' *Understanding Unicode™ - I' & E-Book on 'Guidelines for Writing System Support*: Technical Details: Encodings and Unicode: Part 2' by Sean Pue

[4] http://scripts.sil.org/IWS-Chapter04a
http://scripts.sil.org/WSI_Guidelines_Sec_6_2

[5] Standards for *language technology industry, Technology Development for Indian Languages*
http://tdil.mit.gov.in/FAQs.aspx