सत्यमेव जयते

Best Practices For Localization of E-Governance applications In Indian Languages

सत्यमेव जयते

Revision history

Version	Changes	Date
1.0 (Draft)	Based on feedback	7th June 2012
2.0	Based on feedback	3rd August 2012
3.0	Based on feedback	3rd October 2012
4.1	Based on feedback	6th March 2013
5.2 (CDAC L10N guidelines merged with DeitY guidelines)	Based on feedback	9th May 2013
5.3	Based on feedback	21th July 2013
5.4	Disclaimer added	20th July 2013
5.5	Based on feedback	17th September 2013
5.6	Based on Feedback (DeitY Logo added)	25th November 2013

Important information:

It is noticed that many legacy and existing e-Governance applications are based on proprietary tools, technologies and database formats. Quite a few of these also form part of some of the Mission Mode Projects envisaged under the NeGP. The scope of this document therefore has been kept broad to also include specific sections on proprietary software to cater to the requirements of such applications. The National e-Governance Plan (NeGP) of the Government of India however strongly recommends adherence to Open Standards while taking decisions on the use of tools, technologies and database formats for all software applications being currently planned and developed for delivering e-Governance services and solutions. All stakeholders are thus exhorted to contribute to the cause of open standards to help ensure seamless integration of services and sharing of data across different e-Governance applications and services."

सत्यमेव जयते

Contents

Summary (for Project Managers) 7

1 Introduction 8

1.1 Outline 8

1.2 Purpose of this Document 8

1.3 Scope 8

2 Background 9

3 Standards 11

3.1 Text Localization standards 11

3.1.1 Inputting multilingual data 11

3.1.2 Displaying multilingual data 13

3.1.3 Sakal Bharati font 17

3.1.4 Storing multilingual data 19

3.1.5 Unicode 20

3.2 Localization standards 22

1. XLIFF 22

2. SRX 22

3. TMX 22

4. TBX 22

5. GMX-V 22

6. xml:tm 22

4 Challenges in Software Localisation 27

4.1 Text Input / Output 27

4.2 Text editing and fonts: 28

4.3 Locale specific knowledge: 28

4.3.1 Number formatting: 28

4.3.2 Currency formatting: currency symbols 28

4.3.3 Date formatting: 29

4.3.4 Non-Gregorian calendars: 29
4.3.5 Measurement units: 29
4.3.6 Address formatting: 29
4.3.7 Personal titles: 29
4.3.8 Searching and sorting: 30
4.3.9 UI related issues: 30
5 Locale Support 30
5.1 What is Locale Data 31
6 Common Language Data Repository (CLDR) 32
6.1 APPROACH 33
6.1.1 ANALYSIS 33
7 Frequently Used Entries for Localization (FUEL) 37
8 Generic Guidelines 38
8.1 Categorical Classification of Guidelines: 44
9 Migration to Unicode 50
9.1.1 Compiling Unicode Applications in Visual C++ 51
10 Code Elements 52
10.1 Language identification and Negotiation BCP 47/RFC 5646 52
11 Data Encoding and Byte Order Markers 54
11.1 Tips and Tricks for C/C++/VC++ 56
11.2 Encodings in Web Pages 57
11.2.1 Setting and Manipulating Encodings 58
11.2.2 HTML pages 59
11.2.3 ASP.Net 59
11.2.4 XML pages 60
11.2.5 Java 60
11.2.6 PHP 60
11.2.7 PYTHON 64

सत्यमेव जयते

11.2.8 XML 65
11.2.9 XHTML 68
11.2.10 X-Forms 70
11.2.11 HTML 5.0 72
12 Cascading Style Sheets (CSS) 74
12.1 How to Use @font-face 74
12.2 CSS rendering Issues in Indic Script 75
13 Using Unicode In Databases For Multilingual Data 80
13.1 MS SQL: 80
13.2 Other Databases: 81
13.2.1 MySQL Data Types - NVARCHAR 81
13.2.2 MySQL NVARCHAR - Equivalents in Other Databases 81
14 Need of W3C Validation services for Indian websites 83
14.1 CSS Validator 84
14.2 XHTML Validator 85
14.3 Mobile OK Validator 86
15 Important W3C Standards to be adopted 87
16 ISO 639 Language Codes 88
17 References 90
18 Acronyms & Abbreviations 92
Annexure 93

Summary (for Project Managers)

This document discussed various guidelines to help developers to localize their software products / services. This document will benefit new software developers in using the existing technologies together with various initiatives and standards, which will help them, understand the complications involved with various scripts and platforms. A further benefit is better integration and interoperability of products. The document starts with the definitions of Localization (L10N), Internationalization (I18N), and Globalization. It also brings to limelight some national as well as international standards currently used in the Localization industry. Various important topics have been touched in this document like Inputting, storage and rendering Indian language data, Unicode migration from legacy data, Usage of Common Locale Data Repository (CLDR), Characters encoding for proper representation on various platforms, Unicode, GETTEXT package, Culture and locale specific information for Indian languages, guidelines to develop localized software application / service, encoding and byte order markers to differentiate and manipulate different files, directionality issues with right to left scripts such as Urdu and the problem of using Cascading Style Sheets (CSS) in context Indian languages.

The topics such as Authoring, Internationalization and Localization have been discussed in details with XLIFF as an example. Frequently Used Entries for Localization (FUEL) for term consistency as an open source initiative, in context of Indian language has been cited. Various ISO 639 Language Codes have been provided at the end of the document. However, some of the guidelines are generic in nature and may or may not be applicable to mobile platforms.

सत्यमेव जयते

1 Introduction

1.1 Outline

This document defines general guidelines to help developers to localize their software products / services. The document starts with the definitions of Localization (L10N), Internationalization (I18N), and Globalization. The advantage behind defining Localization is to create a common understanding of the term “Localization” and to learn about the dos and don’ts in order to create a localized application. The document will also discuss some of the frequently used national as well as international standards used in the Localization industry. The document has been divided into various topics viz., Localization / internationalization standards, User Interface (UI), directionality issues, Frequently Used Entries for Localization (FUEL), resource identifiers, Common Locale Data Repository (CLDR), Characters encoding, Unicode, Culture , Locales etc.

1.2 Purpose of this Document

This document is intended for:

Software Designers / Engineers: To understand and evaluate Localization areas related to product design.

Testing and QA Engineers: To define test plans and test cases keeping Localization and Internationalization in mind.

1.3 Scope

The scope of this document is to create awareness among the software developers and the designers to create applications/ products keeping the Localization aspect in mind. These guidelines do not describe how to internationalize a product. However, there are some coding snippets / examples which are used to illustrate the concept and not intended as a guide to implementation. Some of the guidelines are generic in nature and may or may not be applicable to mobile platforms.

2 Background

India is a multilingual country with 22 official languages and 12 different scripts. The official languages of India are Hindi, Marathi, Sanskrit, Bangla, Assamese, Punjabi, Gujarati, Malayalam, Maithili, Santali, Bodo, Dogri, Tamil, Oriya, Telugu, Kannada, Urdu, Sindhi, Kashmiri, Manipuri, Konkani and Nepali. The various scripts are: Devanagari, Gurmukhi, Bengali, Assamese, Gujarati, Kannada, Oriya, Telugu, Malayalam, Tamil, Perso-Arabic, Meitei Mayek, Ol chiki and Roman [16].

This document discusses various Localization standards along with the basic needs required to localize a piece of text i.e. input, storage and display. Standards formulating organizations are ISO (International Organization for Standardization), W3C (World Wide Web Consortium) and OASIS (Organization for the Advancement of Structured Information Standards); here we will focus on the translation and Localization standards developed by the last two organizations.

The definition of a standard according to ISO/IEC Guide 2:2004 [17] provides the following general definition of a standard:

> "a document, established by consensus and approved by a recognized body, that provides, for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context".

Various standards and specifications used in Localization are: Unicode, XML Localization Interchange File Format (XLIFF), Translation Memory eXchange (TMX), Term Base eXchange (TBX), Segmentation Rules eXchange (SRX), Indian Script Code for Information Interchange (ISCII)[18], Perso-Arabic Script Code for Information Interchange (PASCII) [19] etc.

There are mainly four terms widely used in Localization domain:

GILT stands for Globalization, Internationalization, Localization and Translation [20].

There are many definitions of GILT, but for the purposes of this document the following definitions will be used.

Translation is not just about text conversion but also deals with conversion of spoken words from source to target language, without losing its meaning.

Anastasiou & Schäler (2010) defines Translation [21] as *"Translation is the text transfer from a source to a target language; text is everywhere, in laws, news, academic dissertations, user manuals, advertisements and so on; the list is endless. Besides, we often see that text is accompanied by icons, diagrams, and other visual effects. For example, in newspapers as well as in user manuals and advertisements we have many pictures, animations, logos and so on. We often see that these icons change, when they are transferred to a target language."*

LISA defined Localization as follows[22]:

> “Localization refers to the actual adaptation of the product for a specific market. It includes translation, adaptation of graphics, adoption of local currencies, use of proper forms for dates, addresses, and phone numbers, and many other details, including physical structures of products in some cases”.

Schäler (2007) with emphasis on digital content defines Localization as:

> "Localization is the linguistic and cultural adaptation of digital content to the requirements and locale of a foreign market, and the provision of services and technologies for the management of multilingualism across the digital global information flow."

Globalisation deals with the strategy making for selling the product or services to the foreign markets.

> Sikes (2009) defines Globalisation [23] as
>
> *“Expansion of marketing strategies to address regional requirements of all kinds” while Localization is “Engineering a product to enable efficient adaptation to local requirements”*

Internationalisation [24]:

> *“Internationalisation is the design and development of a product, application or document that enables* ***easy Localization*** *for target audiences that differ in region, culture, or language”*.

In short, internationalisation reduces the engineering effort spent for various languages and cultures. It also makes languages locale independent. A locale is a specific geographical, political, or cultural region. It is usually identified by a combination of language and country, for example, en_UK represents the locale UK English.

3 Standards

The standards have been divided into two categories:

a) Text Localization standards

b) Localization standards

3.1 Text Localization standards

In order to localise a piece of text the following are the three basic steps:

1. Inputting multilingual data (Keyboards),
2. Displaying multilingual data (fonts),
3. Storing multilingual data (Unicode).

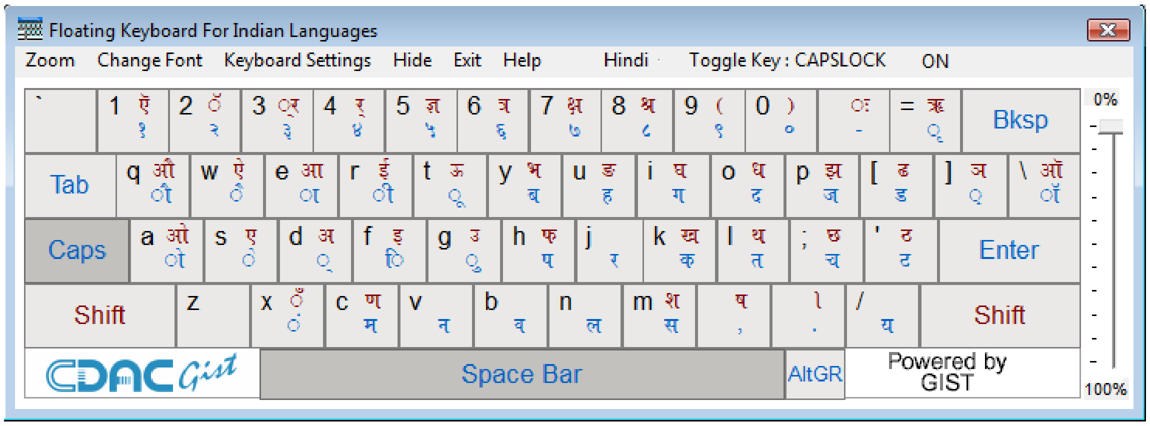
3.1.1 Inputting multilingual data

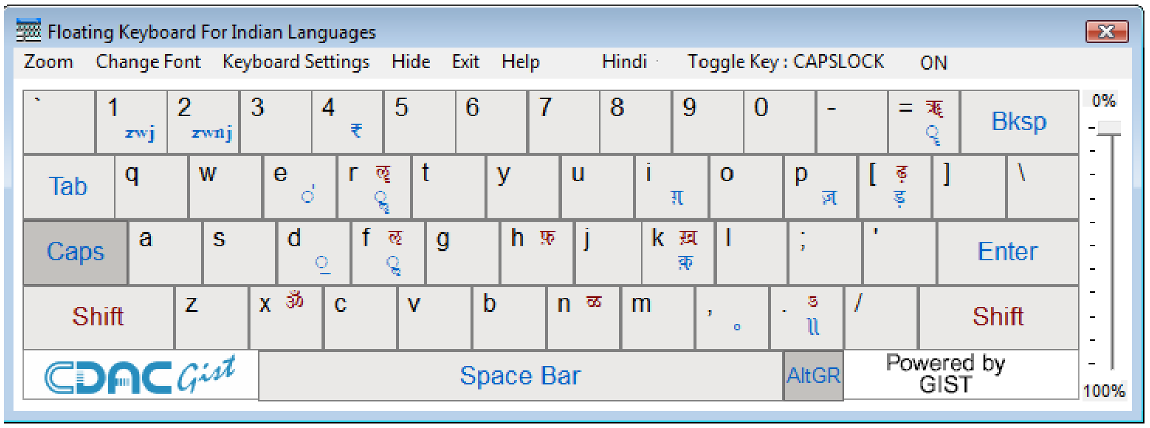
There are three types of inputting mechanism for Indian languages, viz.:

3.1.1.1 INSCRIPT/Enhanced INSCRIPT

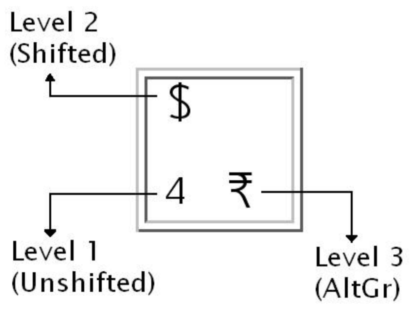
INSCRIPT: The INSCRIPT layout uses the standard QWERTY 101 keyboard. The Indian language characters are divided into vowels and consonants. The vowels are placed on the left hand side of the keyboard and the consonants are placed on the right hand side of the keyboard layout. The mapping of characters in Indian languages is such that it remains common for all the left to right languages. Due to this the basic character set of the Indian languages is common. Enhanced INSCRIPT has got four layers of Keyboard to accommodate more characters. INSCRIPT layout can be used by urban as well as non-urban community.

The following figure represents the INSCRIPT Layout.





3.1.1.2 Rupee Symbol



With the introduction of the new Rupee symbol, there are several hack/non-standard implementations existing on the web. It is mandatory that Rupee symbol placement on the keyboard should be as indicated in the figure above while the storage value should be U+20B9. The utility to input Rupee Symbol can be downloaded from the url: http://www.tdil-dc.in .

3.1.2 Displaying multilingual data

Displaying multilingual data would require fonts. Font is a set of well-defined shapes to display symbols (letters, punctuation marks, special characters of the language.) An 8-bit font can represent 256 glyphs by giving unique index (called glyph index) and name to each glyph/shape.

Since Indian scripts are complex in nature and because of this complexity rendering of these scripts is different than its display. Hence, many legacy standards have been developed to display ISCII text eg. Indian Script Font Code (ISFOC) [25] which will require special software to render stored text in ISCII. The right thing would be to use language specific fonts and converters in order to render text as per the language rules. With the evolution of computing environment from 8-bit to 16-bit ISCII was surpassed by Unicode and True type font by “Open Font Format” Font.

3.1.2.1 “Open Font Format” Font

Open Type is a cross-platform font file format developed jointly by Adobe and Microsoft[16].

“Open Font Format” Fonts convert the Unicode code numbers to their glyphs on the display interface. They are directly based on Unicode. Open Type provides a series of enhancements to the TrueType format, the most significant of which allows PostScript font data to nest inside a TrueType software “wrapper.

The main capabilities of Open Type are:

- Broadest multiplatform support
- Enabling fonts to address large character sets.
- Improved publishing for the internet.
- Better protection for fonts against piracy and duplication

3.1.2.1.1 Multiplatform Support:

Open Type is a superset of the existing TrueType and Type 1 formats, and provides support for both type in print and on-screen. “Open Font Format” Fonts works right “out of the box.” Both PostScript and TrueType versions of Open Type are supported across all platforms, making fonts easier to use and more versatile.

Your application needs to use Unicode and the UTF-8 encoding. The user’s platform needs to have an “Open Font Format” Font available on the Operating System for it to be used by the browser.

3.1.2.1.2 Large Character Sets:

Open Type allows type designers and font foundries to create larger character sets within fonts. Within the parameters of the TrueType and Type 1 formats, fonts are limited to 256 characters. If a typeface designer wanted to create an extended ligature set, small caps, swash and alternate characters, or characters to support multiple languages, these had to be put into another font. The large character set capabilities of Open Type allows type designers much more latitude in typeface design, resulting in better graphic communication.
This enables Indian Languages to be addressed "perfectly" by "Open Font Format" Fonts and all characters can be shown as they are supposed to.

3.1.2.1.3 Improved Internet and PDF Publishing:

Open Type makes it possible for Web page creators to include high quality on–screen fonts with their online documents. This means that designers are able to produce typographically richer documents and reduce the time required to download and display these documents on screen.

3.1.2.1.4 Better Font Protection:

"Open Font Format" Fonts also contain a "digital signature" that allows operating systems and browsing applications to identify the source and integrity of fonts – including embedded font files obtained in Web documents. Font developers are able to encode embedding restrictions in "Open Font Format" Fonts to maintain better control over how their fonts are used.

3.1.2.1.5 Font Embedding:

The "Open Font Format" allows for font embedding. This means that a font can be included with a file and sent to someone else. There are four approaches to "Open Font Format" Font embedding:

- Font embedding that allows the document to be viewed on screen and printed
- Font embedding that allows viewing, printing, and document editing
- Font embedding that allows viewing, printing, editing, and installing onto a hard drive
- No embedding allowed.

3.1.2.2 Dynamic Fonts:

In order to view the web pages in Indian Languages, mechanism of dynamic delivery of font came in picture which is commonly known as “Font for Internet”. These fonts are loaded dynamically on the fly and get downloaded along with the web page that uses it and shows characters in readable format. Using this concept the problem of manual download of font got eliminated. It is not always possible that the Indian language fonts used to develop these web pages will be installed on the client's machine.

Following are the technologies available to support Dynamic Fonts

Portable Font Resource (PFR) - This is based on the web font technology from Bit Stream. PFR is closely coupled with Netscape browsers (4.03 and above) while it also performs well with Internet Explorer (Ver 4.0 and above). In Internet Explorer however, only a one-time download of a control on the client's machine is allowed. The dynamic font generated will have the extension as .pfr

PFRs also have the URL security and can be locked to particular URLs. PFRs need Encoding changes in Internet Explorer 5.0. Occasionally in IE 5.0 the user need to make the encoding as 'User Defined' to view the page properly.

Embedded Open Type (EOT) - Microsoft's Web Embedding Font Technology (WEFT) is used to generate EOT information. EOTs are best suited for Internet Explorer (IE 4.0 and above). Netscape browsers do not support EOTs.

The dynamic font generated has the extension as .eot. If there is a link to an EOT on a page, then the browser uses these EOTs to display the page with specified fonts. EOTs have specific URL roots so that only specified Web sites can use specific EOTs suitable to them. Once an EOT is prepared it is locked to that URL. The same EOT cannot be reused for some other site.

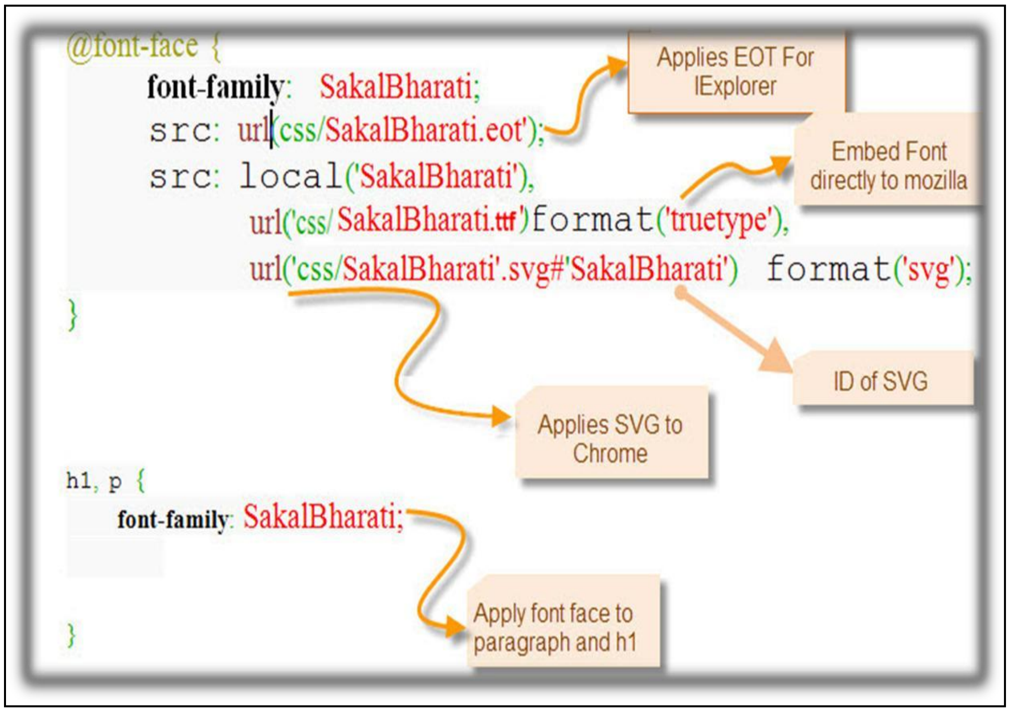
PFR's and EOT can co-exist on a single site. The Browser will sense the dynamic font and accordingly display.

सत्यमेव जयते

Embedding Font in different Browser using Font Face

A Font can be directly embedded into web page by keeping the Font on the server or the place where the webpage is hosted.

CSS code for @font-face embedding



```
@font-face {
        font-family:   SakalBharati;
        src:  url(css/SakalBharati.eot');
        src:  local('SakalBharati'),
                url('css/ SakalBharati.ttf')format('truetype'),
                url('css/SakalBharati'.svg#'SakalBharati')  format('svg');
}

h1, p {
      font-family: SakalBharati;


}
```

Applies EOT For IExplorer
Embed Font directly to mozilla
ID of SVG
Applies SVG to Chrome
Apply font face to paragraph and h1

Fig 3 : Embedding fonts

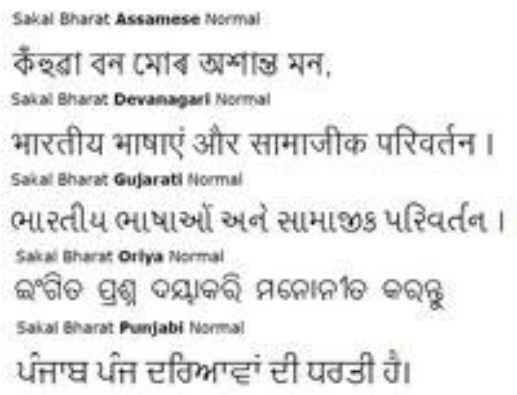
Web Font formats supported by various web browsers:

	Internet Explorer	Mozilla Family	Safari	Chrome	Opera
.ttf	No	3.5	3.1	2.0*	10
.otf	No	3.5	3.1	No	10
.eot	4	No	No	No	No
.svg	No	No	3.1	0.3	9.0

Table 4 : Fonts supported by Web Browser

3.1.3 Sakal Bharati font

A single font which contains all the Indic scripts has been developed by CDAC Pune. This font has got consistent look and feel across various Indic Scripts including English language. The following picture shows how different scripts are rendered on the screen. This font can be downloaded from the url: http://www.tdil-dc.in


Sakal Bharat **Assamese** Normal

কঁহুৱা বন মোৰ অশান্ত মন,

Sakal Bharat **Devanagari** Normal

भारतीय भाषाएं और सामाजीक परिवर्तन ।

Sakal Bharat **Gujarati** Normal

ભારતીય ભાષાઓં અને સામાજીક પરિવર્તન ।

Sakal Bharat **Oriya** Normal

ଇଂଗିତ ପ୍ରଶ୍ନ ଦୟାକରି ମନୋନୀତ କରନ୍ତୁ

Sakal Bharat **Punjabi** Normal

ਪੰਜਾਬ ਪੰਜ ਦਰਿਆਵਾਂ ਦੀ ਧਰਤੀ ਹੈ।

Sakal Bharati is a Unicode based “Open Font Format” Font which includes 11 scripts in one font i.e. Assamese, Bengali, Devanagari, Gujarati, Kannada Malayalam, Oriya, Punjabi, Telugu, Tamil & Urdu. It has used Mono thick (Equal thickness of horizontal stems and vertical stem) glyphs for all scripts. Same X height for all 11 scripts. There are more than 3698 glyphs in the Font.

In this font a special care of code-point wise different but visually similar characters is taken to help them being distinguished visually in the address bar of a browser. This is especially useful for the systems which do not support Unicode. With the help of these Fonts & Keyboard driver, Keyboard Engine of Sakal Bharti supports Phonetic, INSCRIPT, and Typewriter.

Why to use Sakal Bharti Fonts?

- A large number of scripts are supported (all 22 Official languages)) in one single font i.e. Single Font Multiple Script.
- Equal thickness of horizontal stems and vertical stem) glyphs for all scripts
- Sakal Bharati fonts have a single height.
- The font is pleasing to the eye and can be deployed in a wide range of applications where readability is a prime issue.
- Supports Indian Rupee Symbol.

3.1.1 WOFF (Web Open Font Format)

This format was designed to provide lightweight, easy-to-implement compression of the font data, suitable for use in conjunction with the @font-face CSS declaration. Any TrueType/Open Type/Open Font Format file can be loss lessly converted to WOFF for Web use (subject to licensing of the font data); once decoded by a user agent, the WOFF font will display identically to the original desktop font from which it was created.

The WOFF format also allows additional metadata to be attached to the file; this can be used by font designers to include licensing or other information, beyond that present in the original font. Such metadata does not affect the rendering of the font in any way, but may be displayed to the user on request.

The WOFF format is not expected to replace other formats such as TrueType/Open Type/Open Font Format or SVG fonts, but provides an alternative solution for use cases where these formats may be less performant, or where licensing considerations make their use less acceptable.

Why Use WOFF

WOFF provides a lot of advantages over other font choices. The standard fonts you can use end up making a web page. WOFF fonts have the following benefits:

- They are more accessible than fonts as images. WOFF styles plain HTML text with CSS.
- WOFF files include typographical information like contextual forms and old style figures. This gives your web pages better typography because you're using the correct characters, not just approximations.
- WOFF fonts can help with internationalization because you can embed fonts with characters from other languages.
- Like all CSS styled text, fonts styled with WOFF are more search engine friendly.
- WOFF fonts are compressed, so they are smaller than other types of font files

3.1.3.1 Rendering / Rasteriser Engines

Displaying multilingual data would also require, rendering / rasterisers engines such as Uniscribe, Pango, ICU, Harfbuzz.

Uniscribe: Uniscribe is a Microsoft shaping and rendering engine to ensure proper representation of multilingual content on windows platform.

Pango: Pango is a library for laying out and rendering of text, with an emphasis on internationalization. Pango can be used anywhere that text layout is needed, though most of the work on Pango so far has been done in the context of the GTK+ widget toolkit. Pango forms the core of text and font handling for GTK+-2.x.

ICU: ICU is a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.

Harfbuzz: HarfBuzz is an OpenType text shaping engine.

For further details please refer the following urls:

ICU: http://site.icu-project.org/

PANGO: http://www.pango.org/

Harfbuzz: http://www.freedesktop.org/wiki/Software/HarfBuzz

3.1.4 Storing multilingual data

Multilingual data can be stored in Unicode.

Unicode consortium defines Unicode as

"Unicode is the universal character encoding, maintained by the Unicode consortium. This encoding standard provides the basis for processing, storage and interchange of text data in any language in all modern software and information technology protocols."[4]

It is the superset of all the languages in the world which also includes punctuation, special characters (shapes), currency symbols, mathematical symbols etc [5]. Using Unicode, more than 65000 different characters can be represented. Unicode comprises of many code pages.

3.1.5 Unicode

Unicode is like ASCII. It is the only basis of internationalization in the world. All world standards that enable applications to be designed for localization (Localization) depend upon it.

3.1.5.1 What is Unicode?

Unicode is the standard for information interchange worldwide as all IT Companies support it. It is the Universal character encoding standard, used for representing text for information processing. Unicode encodes all of the individual characters used for all the written languages of the world. The standards provide information about the character and their use. Unicode uses a 16 bit encoding that provides code point for more than 65000 characters (65536). It assigns each character a unique hexadecimal numeric value and name. Unicode and ISO10646 Standard provide an extension mechanism called UTF-16 that allows for encoding as many as a million. Presently Unicode Standard provides codes for 49194 characters.

3.1.5.1.1 Principles of Unicode Standard

i. **Universality**: It applies to all the scripts of the world. This means that all Indian scripts are addressed.

ii. **Efficiency:** All characters are equally accessible and there are no modified states in the encoding. This means all characters are pure and have no enhancements.

iii. **Characters, not glyphs:** The visual characters are not represented (except mistakes) but only the component text elements.

iv. **Semantics:** These are used to enhance item “efficiency” above to allow components to be used correctly in the input/output/processing context so that the user experience is perfect.

v. **Plain text:** Unicode characters when used in isolation represent only the characters themselves. These can then be enhanced with tags and other markers in the context of their use to get the intended results.

vi. **Logical order:** This is how Unicode characters are stored in memory devices. It is usually left to right. Logical order needs to be converted to the required physical/processing order to get the required result. For example, Unicode characters for Urdu could be stored from left to right (in logical order) but would be required to be converted to their physical order (right to left) to be read correctly.

vii. **Unification:** There is no duplication in how Unicode characters are encoded. This is made possible by the efficiency rule.

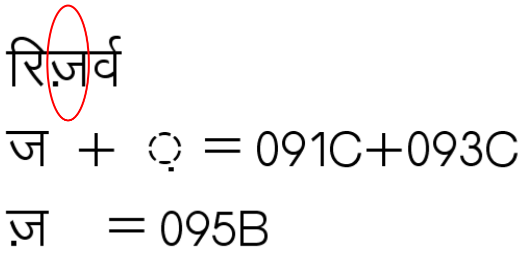
viii. **Dynamic Composition:** Unicode allows for component characters to be combined with others to form other characters. This is specially applicable to Indian Languages where most “matras” are formed using this.

ix. **Stability:** Most importantly, this means that once Unicode characters are assigned, their code point assignments cannot be changed, nor can characters be removed. Characters are retained in the standard, so that previously conforming data stay conformant in future versions of the standard. Sometimes characters are deprecated.

x. **Accurate Convertibility:** Because of principle IV above, Unicode data is fully convertible with any unique mapping required for other standards. This means that there is guaranteed one-to-one mapping between Unicode and ISCII. This makes all kinds of migration possible.

3.1.5.2 Normalization in Unicode

The Unicode data requires normalization. There are many cases where a character can be entered in more than one ways using the Unicode code chart.



रिज़र्व

ज + ◌़ = 091C+093C

ज़ = 095B

eg.

One must take utmost care while developing applications in Unicode like internationalized domain names (IDN).

3.2 Localization standards

The various standards in today's Localization world are: XLIFF, Segmentation Rules eXchange (SRX), Translation Memory eXchange (TMX), Term-Base eXchange (TBX), Global Information Management Metrics eXchange-Volume (GMX-V), XML Text Memory (xml:tm).

1. **XLIFF:** is an XML based intermediate format which is used to store, carry and interchange localizable data. According to XLIFF specification [13]: “XLIFF is the XML Localization Interchange File Format designed by a group of software providers, Localization service providers, and Localization tools providers. It is intended to give any software provider a single interchange file format that can be understood by any Localization provider.”

2. **SRX:** SRX rules based on XML vocabulary was developed for breaking the text into translatable segments/ smaller fragments. SRX is defined in two parts: <languagerules>: specification about rules applicable for each language. <maprules>: specification about how rules are applied for each language.

3. **TMX:** TMX is the translation memory data exchange standard between applications. It is divided into two parts: Translation Unit <tu> and Segment of translation memory text <seg>.

4. **TBX:** TBX-Basic is a TBX compliant terminology markup language for translation and Localization processes that permit a limited set of data categories. The purpose of TBX-Basic is to enhance the ability to exchange terminology resources between users.

5. **GMX-V:** It measures the work-load for a given Localization job, not just by word and character count, but also by counting exact and fuzzy matches, punctuation symbols etc. It can also be used to count the number of pages, screenshots etc.

6. **xml:tm:** It is the vendor-neutral open XML standard, which allows text memory including translations to be embedded within XML documents using XML namespace syntax.

The standards we are going to describe now follow the order of a Localization workflow, i.e. authoring, Internationalization and Localization as shown in the following diagram:

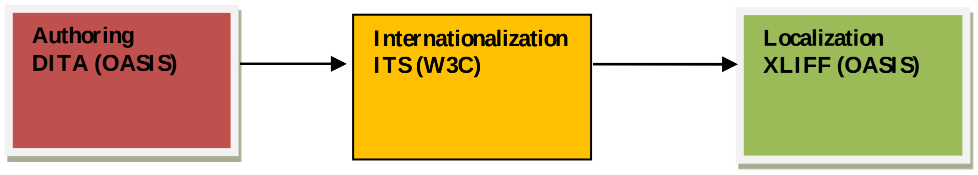


Fig. Authoring, internationalization, and Localization standards

At authoring stage, the standard which leaps out is DITA (Darwin Information Typing Architecture) managed by OASIS. ITS (Internationalization Tag Set) is the internationalization standard by W3C, while XLIFF is a standard which carries Localization data and metadata under the umbrella of OASIS. DITA is an XML-based specification which processes modular and extensible topic-based information types as specializations of existing types. It allows for the introduction of specific semantics for specific purposes.

An example of DITA is shown below:

(1) DITA example

```
<task id="installstorage">
<title>Installing a hard drive</title>
<shortdesc>You open the box and insert the drive.
</shortdesc>
<prolog><metadata>
<audience type="administrator"/>
<keywords>
<indexterm>hard drive</indexterm>
<indexterm>disk drive</indexterm>
</keywords>
</metadata></prolog>
<taskbody>
<steps>
```

```
<step><cmd>Unscrew the cover.</cmd>
<stepresult>The drive bay is exposed.</stepresult>
</step>
<step><cmd>Insert the drive into the drive bay.</cmd>
<info>If you feel resistance,try another angle.</info>
</step>
</steps>
</taskbody>
<related-links>
<link href="formatstorage.dita"/>
<link href="installmemory.dita"/>
</related-links>
</task>
```

The example is retrieved from DITA OASIS online community portal: http://dita.xml.org/what-do-info-typed-dita-topic-examples-look, 10/06/12.

In the above mentioned example the metadata tag <metadata> stores the meta information for Keywords, index terms and audience type. After this the steps and the steps results are given. There is also a provision to link other DITA files too.

After this Authoring stage the next stage is Internationalization. Internationalization reduces the engineering effort spent for various languages and cultures. Directionality (RTL or LTR), which is very important for internationalization, is supported by the ITS specification.

The ITS specification consists of “data categories”, which is a set of elements and attributes. Perhaps the most important data category is <translate>, as it expresses information about whether the content of an element or attribute should be translated or not. The values of this data category are “yes” (translatable) or “no” (not translatable).

The selection of XML node(s) may be explicit (using local approach) or implicit (using global rules). An ITS example using local approach is given below:

(2) ITS example

```
<dbk:article
xmlns:its="http://www.w3.org/2005/11/its"
xmlns:dbk="http://docbook.org/ns/docbook"
its:version="1.0" version="5.0" xml:lang="en">
<dbk:info>
<dbk:title>An example article</dbk:title>
<dbk:author its:translate="no">
<dbk:personname>
<dbk:firstname>John</dbk:firstname>
<dbk:surname>Doe</dbk:surname>
</dbk:personname>
<dbk:affiliation>
<dbk:address>
<dbk:email>foo@example.com</dbk:email>
</dbk:address>
</dbk:affiliation>
</dbk:author>
</dbk:info>
<dbk:para>This is a short article.</dbk:para>
</dbk:article>
```

Here we see that we do not have an explicit metadata tag, but metadata is still available, such as title, author, person's name (first name and surname), affiliation, address, and contact e-mail. We also see that that all author's information should not be translated as indicated by the data category translate="no": <dbk:author its:translate="no">.
A basic minimal XLIFF file (3) follows:

(3) Minimal XLIFF file with one translation unit (TU)

```
<xliff version="1.2"
xmlns="urn:oasis:names:tc:xliff:document:1.2"
xmlns:xsi='http://www.w3.org/2001/XMLSchemainstance'
xsi:schemaLocation='urn:oasis:names:tc:xliff:documen
t:1.2 xliff-core-1.2.xsd'>
<file original="Greetings.txt" source-language="en-US"
```

```
datatype="plaintext">
<body>
<trans-unit id="#1">
<source xml:lang="en-US">Thankyou</source>
<target xml:lang="de-DE">Danke</target>
</trans-unit>
</body>
</file>
</xliff>
```

On the first line we have the XLIFF declaration and also the schemaLocation attribute of the XML schema-instance namespace. In the file element we have the name of the file (Greetings.txt), its source language (English (US)), and its data type (plain text). Then the translation unit element follows with its source and target text in the source language (SL) and target language (TL), respectively. XLIFF also allows the storage of important data for (software) localisation; an example is the restype attribute. Among its values, are checkbox, cursor, dialog, hscrollbar (horizontal scrollbar), icon, menubar (a toolbar containing one or more tope level menus), and window. An example15 of a dialog resource type follows:

(4) XLIFF TU of a dialog

```
<trans-unit id="34" resname="IDD_ABOUT_DLG"
restype="dialog" coord="0;0;235;100" font="MS Sans
Serif;8" style="0x0932239">
<source>About Dialog</source>
</trans-unit>
```

In example (4), we see metadata about font, style, and coordinates. This is specific to the dialog resource type. When metadata is about the file and the localisation process in general, then it is included in the header element. An example of the header element follows:

(5) Process metadata in header element

```
<header>
<phase-group>
<phase
phase-name="engineering"
process-name="sizing"
contact-name="John Doe"
contact-email=foo@example.com/>
</phase-group>
```

</header>

4 Challenges in Software Localisation

Challenges in Software Localisation There are several kinds of issues that arise in the process of releasing software into a new locale, and most of the times these issues requires the programmers to change the source code to handle the scenarios specific to different communities.

A completely internationalized product is never supposed to undergo a change in source code during the localization process.

Some common problems faced during software localization are discussed below.

4.1 Text Input / Output

Often software is unable to handle the characters of the target UI languages because during development it was designed to address a single human language. So it might fail to handle the characters of target UI languages and display junk characters instead of proper ones.

The following errors are generally observed in the area of Text I/O.

- Costs of Localization and Impact on Pricing.
- Language and Content.
- Currency.
- Time and Date Formatting.
- Measurements.
- Formats (Paper and Envelope Sizes, Address Formats).
- Collating Sequences.
- Numeric Formats.
- Color Scheme.
- Images and Sounds.

4.2 *Text editing and fonts:*

In text editing the general problems can be isolated to the following areas.

- Keyboard shortcuts might need to be changed according to the translated text that contains these shortcuts.
- The font name, size, and style in dialog boxes and documents can be changed to customize the final UI based on user preference.
- Incorrect cursor positioning.
- Inconsistent click, double-click and multi-click selections

4.3 *Locale specific knowledge:*

Locale-aware software means that the functional behaviour of software and display of UI elements and text varies as per the rules of a particular locale. In general this covers the following issues.

4.3.1 Number formatting:

Consider the examples below to understand the variations present in the number formatting in different locales.

- 12,34,56,789.00 in India (One grouping on thousand and then grouping on hundreds)

4.3.2 Currency formatting: currency symbols

- 4,567.89 (Indian Rupee)

4.3.3 Date formatting:

Elements like order, width of fields, separators, padding with zeros etc.

- DD-MM-YY (India Date)

Time formatting: It is also governed by locale.

- 10:00:15 PM (India Time)

4.3.4 Non-Gregorian calendars:

In different countries different calendars are used, even in a single country there can multiple calendars used. Some common calendars in use: Gregorian, Arabic Lunar, Buddhist, Japanese and so on.

4.3.5 Measurement units:

There will be problems if the software does not handle the unit measurement system of that locale properly or does not provide a mechanism for inter-conversion, common systems in use are foot, pound, second (FPS) and meter, kilogram, second (MKS).

4.3.6 Address formatting:

Software that deals with postal addresses will show error messages or will display the wrong output if it is not designed to handle the address formatting as per the selected locale of the software.

4.3.7 Personal titles:

One needs to be careful in deciding the length of fields for the names/family names/various titles .Ordering of title, first name, family name is also important and varies from locale to locale.

- Generational suffix (Jr., Sr., II, III)
- Honorific (Mr., Mrs., Ms., Miss, Dr., Master)
- Title (President, Director)
- Rank (Lt., Maj., Col.)
- First name
- Family name

Telephone number formatting: The problems in this section are generally caused by mishandling of the:

- Number of digits
- Digit groupings (country code, area code, local number)
- Digit separators (spaces, hyphens, periods, parentheses)
- Use of extensions

4.3.8 Searching and sorting:

The sorting order of data and the ability to search are among the most critical and important features of the software. The software may sort incorrectly and not find the searched text if the following issues are mishandled in a particular locale.

- Binary order A < C < Z < a < c < z < Ç
- Code Point Order (same as UTF-8 binary comparison)
- UTF-16 Order (Java String binary comparison)
- Refinements, usually only for matching, not sorting
- Case-insensitive
- Matching equivalent forms of text

4.3.9 UI related issues:

Translated strings can be of different height, width and length so the UI elements' size for the source strings may not be suitable for the target language. This can cause truncation of the UI elements.

Additionally we may face problems related to duplicate and non-functioning hotkeys and keyboard shortcuts.

5 Locale Support

The Unicode CLDR provides key building blocks for software to support the world's languages, with the largest and most extensive standard repository of locale data available. This data is used by a wide spectrum of companies for their software internationalization and localization, adapting software to the conventions of different languages for such common software tasks as:

- formatting of dates, times, and time zones,
- formatting numbers and currency values
- sorting text
- choosing languages or countries by name

Language matching is used to find the best supported locale ID given a requested list of languages. The requested list could come from different sources, such as such as the user's list of preferred languages in the OS Settings, or from a browser Accept-Language list. For example, if my native tongue is Hindi, I can understand Punjabi and Tamil, my Bangla is rusty but usable, ideally an implementation would allow me to select preferred list of languages, skipping Bangla because my comprehension is not good enough for arbitrary content.

Language Matching can also be used to get fallback data elements. In many cases, there may not be full data for a particular locale. When such fallback is used for resource item lookup, the normal order of inheritance is used for resource item lookup, except that before using any data from root, the data for the fallback locales would be used if available. Language matching does not interact with the fallback of resources within the locale-parent chain. For example, suppose that we are looking for the value for a particular path P in nb-NO. In the absence of aliases, normally the following lookup is used.

The language Matching data is interpreted as an ordered list. To find the match between any two languages, use the likely sub tags to maximize each language, and perform the following steps.

- Remove any trailing fields that are the same.
- Traverse the list until a match is found. (If the one way flag is false, then the match is symmetric.)
- Record the match value.
- Remove the final field from each, and if any fields are left, repeat these steps.

5.1 What is Locale Data

A locale represents information on the rules and preferences that is specific to the end user's particular country, language and territory. For example, the holidays of India are a part of its locale.

The data associated with locale provides support for: presenting, formatting, parsing of local data elements like dates, timestamps, numbers, currencies, measurement units, sort-order (collation), holidays, calendars, translated names for time zones, languages, countries and Scripts.

These elements can be used (and ultimately enhanced) to present a complete set of “user-environment variables” for the application to be presented to the user as completely native.

When used properly, every element of a user’s application-use environment is according to his expectations, practices and conventions. It allows users to work completely according to his real-world environment.

6 Common Language Data Repository (CLDR)

The CLDR provides key building blocks for software to support the world's languages. The data in the repository is used by companies for their software internationalization and localization: adapting software to the conventions of different languages for such tasks as formatting of dates, times, time zones, numbers, and currency values; sorting text; choosing languages or countries by name; and many others. C.L.D.R.’s provide useful information as to the locale and are therefore crucial from the perspective of localization. For the purpose of clarity, CLDR’s may be reduced to two sub-types: CLDR’s are of two types: Basic and Exhaustive.

The Basic CLDR comprises of

- Calendars
- Numeric formats,
- Date and Time formats
- Currencies

These are used not only by Operating Systems to show time date and conversion but also for more extensive functions such as inserting Date, Time, Currency etc in the locale of the country.

Advanced or Exhaustive CLDR’s cover in addition to the above, items such as

- Weight systems

- Distance systems
- Location
- Modes of Address
- Language Selection
- Oral Pronunciation (tied to the Voice Browser)

A majority of CLDR's use LDML for mark-up.

6.1 APPROACH

Existing CLDR templates (for the major part from the Unicode site) were analysed from the point of view of their compliance to Locale data. Experts were invited to give their comments on the tags within the CLDR and whether the existing tags were sufficient. The results of the analysis of CLDR are given below.

6.1.1 ANALYSIS

This analysis of CLDR's is divided into two parts:

Basic CLDR's and Advanced CLDR's

6.1.1.1 BASIC CLDR

The repository contains information as to Time, Date, Year, Currency and Weight. Insofar as the basic CLDR's are concerned, it was noted that a majority were geared towards the Western model. The lacuna in each locale data is analysed in what follows:

YEAR

The year format which is normally followed is the Gregorian calendar. In fact even the CLDR for Hindi and other Indian languages complies with the Western norm, with the months being transliterated into the local language.

It would be advisable for true representation (as in the case of Chinese) to adopt the Indian luni-solar calendar and have the year calculated as Vikram Samwat for Gujarati, Dogri, Marathi and Konkani as well as to a certain extent for Sindhi. In the case of Gujarati, the Parsi calendar where each day of the month has a separate name would involve a very specific CLDR. On the other hand, the calendar for Urdu and Kashmiri could conform to the Muslim calendar.

DATE

The same assumptions could be for Date where the date notation in Indian languages varies and can be

```
DD MM YYYY

DD MM      YY

MM DD YYYY

MM DD    YY
```

TIME

Time is calculated as per Hour Minute and Second. But traditional time calculation in India, especially in Astrology, is based on ghatikas and palas.

Even in the case of Hour Minute Second notation, apart from the Indian railways and Airlines where a 24 Hour notation is maintained, the normal notation is for a 12 hour cycle.

NUMBERS

The case of Numbers is unique, since Numbers in Indian languages have their own grammar and a NUM2Word routine necessarily implies a deep study of the various ways of representing numbers in Indian Languages. This has been one of the areas of intensive study and the results of the Number notation as represented in a Num2Word routine are provided at the end of the chapter for Marathi, Konkani, Gujarati, Urdu, Sindhi, Dogri and Kashmiri.

CURRENCY

The normal currency is Rs and Paise. However in common parlance there are further subdivisions into a quarter, half and three quarters. Should these be represented as is done in the US with the word dime being acceptable?

WEIGHTS MEASURES AND DISTANCE

Although India has adopted the Metric System to be at par with the world, in real time, old weights still continue to function. Should these be included in the CLDR where the Weight and Measures are analyzed.

A similar problem arises in the case of distance where the old Foot and Mile system is partly used. Similarly measurement of land is still in traditional measures. This is especially important for localizing land records where each state has its own traditional term for measurement such as Guntha, Bigha etc.

The above few items show the complexity in evolving a basic CLDR for Indian languages. In the case of the more advanced aspects of the CLDR which are normally not adopted in the standard CLDR, socio-cultural aspects are analyzed.

6.1.1.2 ADVANCED CLDR

This embraces socio-cultural aspects. such as Icons, Images, Symbols, Myths, Beliefs, Geographical entities, Custom and Tradition, Festivals. Another aspect is that of name representation where the traditional FIRST NAME FATHER'S NAME & SURNAME are not pertinent, especially in the South where such features are replaced by patronymics. Similarly notification of location, especially in the case of addresses is not easy to reduce to a single format. Different cultures tend to change the order from Ascending to Descending as is the case in Iran and to a certain extent in parts of Bhuj in Gujarat, where the country is placed first and the destinator is placed at the end. Honorifics and terms of respect and address also form an important part of CLDR which to be totally exhaustive has to embrace each and every aspect of the culture of the locale and render it transparent to the user.

Since exact formats in LDML for these different entities are still under development, these have not been studied in depth.

COMPLIANCY

The different CLDR’s that have been analyzed and studied / and or developed (as in the case of Dogri) are all compliant with the Western notational system. In that sense they are perfectly compliant. They have been implemented by both IBM and Microsoft in their operating systems and within these limits can be termed as perfectly compliant.

However the term “compliancy” implies a norm or standard and if the above discussion is to be gone by, the CLDR’s do not reflect in any manner the cultural and ethnic thought processes of the languages under study. A closer look at this problem which is a vital one especially in the area of e-governance localization is a must.

In what follows two sample CLDR’s one for Dogri and the other for Urdu are presented. Both are in the traditional framework although the Urdu CLDR proposes like the Arabic one a dual system: Gregorian and Muslim. Simultaneously the results of the study undertaken on Num2Word conversion in the shape of a tabular representation of the basic numerals from one to the highest acceptable number are also provided for all the languages under study.

Further details can be found at : http://cldr.unicode.org/

7 Frequently Used Entries for Localization (FUEL)

FUEL is an open source initiative to standardize terms for open source software programs. It aims at resolving the problem of term inconsistency and lack of standardization in Computer software translation, across various platforms. It also works to provide a standard and consistent terminology for a language. Following Indian language support has been added in this initiative.

Languages with FUEL [31]:

Assamese (as)

Bengali (India) (bn_IN)

Bhojpuri (bho)

Chhattisgarhi (hne)

Gujarati (gu)

Hindi (hi)

Kazakh (kaz)

Magahi (mag)

Maithili (mai)

Malayalam (ml)

Marathi (mr)

Punjabi (pa)

Oriya (or)

Tamil (ta)

Telugu (te)

Urdu (ur)

Kannada (kn)

8 Generic Guidelines

i. Use Unicode as your character encoding to represent text.

ii. It is recommended to use Latest version of Unicode and Unicode Compliant Open Font Type during design and deployment of Indian language application (Refer : http://unicode.org)

iii. It is recommended to use of Enhanced INSCRIPT keyboard for inputting

    i. Drawback of Phonetic / transliteration based layouts is that they are useful only for English knowing users.
    ii. Drawback of Typewriter layouts is that they are preferred only by operators migrating from physical typewriters.
    iii. INSCRIPT is easy to learn and use especially for non English speaker.
    iv. Major OS such as MS-Windows and Linux support INSCRIPT by default
    v. It also caters to latest additions such as the Rupee symbol
    vi. Refer : http://cdac.in/downloads

iv. Isolate all user interface elements from the program source code. Move all localisable resources to separate resource-only DLLs. Localisable resources include user interface elements, such as strings, error messages, dialog boxes, menus, and embedded object resources. Resources which are not localisable should not put into the resource-only DLLs.
Refer: http://msdn.microsoft.com/en-us/library/w7x1y988.aspx

v. Use the same resource identifiers throughout the life of the project. Changing identifiers makes it difficult to update localised resources from one build to another. Refer: http://www.lingobit.com/solutions/bestpractices.html

vi. Allow plenty of room for the length of strings to expand in the user interface. In some languages, phrases can require 50-75 percent more space than they need in other languages. For example, dialog boxes may expand due to Localization, a large dialog box that occupies the entire screen in low-resolution mode may require resizing to an unusable size when localised. Refer: http://msdn.microsoft.com/en-us/library/w7x1y988.aspx

vii. UI controls such as buttons or drop-down lists should not be placed on top of other controls. Sizing and hotkey issues with hidden controls usually are found through testing, which might not be done during Localization . In this case, the UI is not localisable because the button size cannot be extended to the length required for the translation without rearranging the button positions. Rearranging button positions can be costly and makes the UI inconsistent among languages. Refer: http://msdn.microsoft.com/en-us/library/aa163857(v=office.10).aspx

viii. There should be proper use of decimal separator which varies from locale to locale.

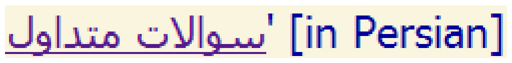
ix. Avoid images, banners with text, because for Indian language version they need to be translated as well and language switch will not be able to handle the text inside images.

x. For desktop applications the icon, icon-text and title should be in local language.

For applications requiring RTL support it recommended that the html tag 'direction' be specified with RTL as value e.g.

```
'<a dir="rtl" lang="fa" href="...">...</a>' [in Persian].
```

That would yield this result:


سوالات متداول' [in Persian]

xi. Application surface Localization are useful where end user is only consuming information and processing / computing with Indian language data is not critical.

i. example of surface Localization is printing of statement of accounts or and bill passbook printing

xii. Use clear, concise, and grammatically correct language. Ambiguous words, obtuse or highly technical sentences, and grammatical mistakes increase translation time and costs.

xiii. When using abbreviations and acronyms, ensure that the abbreviations and acronyms have meanings that are understood by most users. You should always define abbreviations and acronyms that might not be obvious in all languages.

Refer: http://msdn.microsoft.com/en-us/library/aa163854(v=office.10).aspx

xiv. Avoid using images and icons that contain text in your application. They are expensive to localise.

xv. Avoid strings that contain a preposition and a variable are difficult to localise because, in some languages, different prepositions are used in different contexts.

Refer: http://msdn.microsoft.com/en-us/library/aa163852(v=office.10).aspx

Example code	Better code
At %s	Time: %s
At %s	Date: %s
At %s	Location: %s

xvi. Automated translation tools can significantly cut down on Localization vendor's costs. But automatic translation tools only work if standard phrases are being used. Many Localization vendors are paid per word. Consider the amount of money that can be saved if one standard phrase can be easily, or automatically translated into multiple languages. For example, the following messages could be standardized into one consistent message:

Refer: http://msdn.microsoft.com/en-us/library/aa163854(v=office.10).aspx

Message	Standardized version of message
Not enough memory	There is not enough memory available.
There is not enough memory available	There is not enough memory available.
Insufficient Memory!	There is not enough memory available.

xvii. Different languages often have different punctuation and spacing rules. Consider these differences when writing strings in code. Thus, if this string is constructed at run time, the localiser cannot change the point to a comma. For similar reasons, apply these considerations to numbers, dates, or any other information that might have different formats in other languages. Refer: http://msdn.microsoft.com/en-us/library/aa163854(v=office.10).aspx

xviii. Use of numerals / digits

i. For applications involving number crunching such as banking applications, billing applications, statistics it is recommended that English numbers / Digits be used. Unless mandated otherwise by the agency commissioning the project.

- This is because most programming environments, spreadsheets and databases do not support computing with Indian digits which are treated as characters as opposed to numeral.

ii. For applications such as digital data preservation Indian digits may be considered.

xix. **Avoid using compounded variables**

In the following example, to translate the preposition "on" correctly, you might have to ask the developer what the variables stand for.

Refer: http://msdn.microsoft.com/en-us/library/aa163852(v=office.10).aspx

Example String	Explanation from developer
%I:%M%p on %A, %B %d, %Y	%A Full weekday name
	%B Full month name
	%d Day of month as decimal number (01 - 31)
	%I Hour in 12-hour format (01 - 12)
	%M Minute as decimal number (00 - 59)
	%p Current locale's A.M./P.M. indicator for 12-hour clock

	%Y Year with century, as decimal number

xx. Use Microsoft specified “resource file” naming convention.

xxi. **Use unique variable names**

If the same variable name is used for different variables, for example, if the sequence of the variables is hard coded, the word order in the translated sentence might be wrong because word order differs from language to language.

Example code	Better code
Set created on %s at %s	Set created on %1 at %2
Backup of %s on %s at %s	Backup of %1 on %2 at %3
Printing %s of %s on %s	Printing %1 of %2 on %3

xxii. Do not hardcode strings or user interface resources.

xxiii. Allocate text buffers dynamically since text size may expand when translated.

xxiv. **Avoid composite strings**

The strings shown in the following table cannot be localized unless the localizer knows what the type of object or item the variables stand for. Even then, Localization might be difficult, because the value of the variable might require a different syntax; the article "the" has variations in another language (in German: "der, die, das, dem, den, des," the same as in English where you have "a" or "an"); the adjectives might change according to the gender of the word; or other factors. Using composite strings increases the chances of mistranslation. These Localization problems can be eliminated by writing out each message as a separate string instead of using variables.

Refer: http://msdn.microsoft.com/en-us/library/aa163852(v=office.10).aspx

Examples of composite strings
Are you sure you want to delete the selected %s?
%s drive letter or drive path for %s.
Are you sure you want to delete %s's profile?
Cannot %s to Removable, CD-ROM or unknown types of drives.
A %s error has occurred %sing one of the %s sectors on this drive.

xxv. Unused strings and dialog boxes should be removed from the source, so localisers do not waste time localising them.

xxvi. Avoid using controls within a sentence. You might want to place a UI control within a sentence. For example, you might want to give users a drop-down menu to make a choice within a sentence. This practice is not recommended, because to localise a sentence that includes UI controls, the localiser often has to either change the position of the controls (if possible) or be content with an improper sentence structure. Also, the UI controls are often drop-down combo boxes that are comprised of multiple controls. Moving and aligning these can be error-prone.

xxvii. Test localised applications on all language variants of Windows XP (except Oriya). If your application uses Unicode, as recommended, it should run fine with no modifications. If it uses a Windows code page, you will need to set the culture/locale to the appropriate value for your localised application and reboot before testing. Refer: http://msdn.microsoft.com/en-us/library/aa291552(v=vs.71).aspx

xxviii. **Cultural and Policy Making Issues**

i. Avoid slang expressions, colloquialisms, and obscure phrasing in all text. At best, they are difficult to translate; at worst, they are offensive.

ii. Avoid maps that include controversial regional or national boundaries. They are a notorious source of political offense.

iii. Avoid images in bitmaps and icons that are ethnocentric or offensive in other cultures/locales. Refer: http://www.lingobit.com/solutions/bestpractices.html

8.1 Categorical Classification of Guidelines:

Sr. No.	Category	Sub Category	Guideline
1	**Translatable Components**		
		Textual Objects	
		Fixed - textual objects which should not get translated. e.g. - User Name, Group Name, Password, System or host names, keyboard accelerators	All fixed text should be separate from resource file and not translated.
		Message - translated information displayed to the user by the product.	All messages should be reviewed for slang terminology, message fragments, and other criteria that makes message difficult to translate.
			All Translatable strings should be separate from the source and placed in a locale-specific location.
			If translated items do not exist for a locale, correct default values should be used.
			Allow plenty of room for the length of strings to expand in the user interface. In some

		languages, phrases can require 50-75 percent more space than they need in other languages. When variables are used for the text strings, extra space (at least one line per variable) should be provided. There should not be any controls placed in between the sentences. When possible, it is best to put text labels above UI controls such as edit boxes. This positioning allows for the greatest extension of the text field. Avoid inline CSS with values. Never use absolute positions. UI controls such as buttons or drop-down lists should not be placed on top of other controls. Text on a button should never be dynamically linked to the button from a string variable. However, it can be placed on the button itself as a property of the button.
	Other - translated help files, documentation.	Help files / documentation should be available for different Locales.
	Non - Textual	
	Icons, Images, Colors, Sounds, etc	These items should be culturally neutral as much as possible. Avoid culture-specific examples, showing body parts, gender-specific roles, religious references, political symbols, text in graphics.

		If these items are not culturally neutral then all non-textual items should reside in a locale-specific directory and these items should be separate from the source product If these items are not culturally neutral and if an item has not been translated into a specific locale then correct default item should appear.
		If these items are not culturally neutral and if text messages appear in icons/ images, then the text from the icon or image should be easily separated from the image. And these messages must be translated separately from the icon. 5. If these items are not culturally neutral, the tools which were used create the items should be easily available in other places, where the localization of these items will take place.
2 **Cultural Data**		
	Time, Date and Calendar	
	- system or user log files or text windows - display of time and date information in files - calendar functions - display of chronologically-sorted data - display of user accesses to the system	Time, Date and Calendar format in a locale-specific format.
	Numbers, Currency, and Metrics	
		Numbers should display in the locale-specific format. The currency symbol should be displayed in

		the locale-specific format. Currency strings should be formatted according to locale conventions. If using the International three-letter currency code, correct one should be used according to the locale.
	Collation	
	Collation refers to the order in which characters are sorted.	If the sort order changes in different locales then the sorting should be according to the rules of the respective locales.
	Personal Names, Honorifics, etc	
		The formats for personal names, honorifics and salutations should be configurable for each locale. The formats for postal addresses and telephone numbers should be configurable for each locale.
	Characters and Strings	
		Application should handle Unicode and convert correctly between Unicode and the

			native encoding of the platform. i.e. non-ASCII filenames and pathnames should be handled correctly.
		Filesystem I/O	
			Can non-ASCII characters be saved and loaded correctly from the file system, without mention of proper byte order marker?
3	**Text (Writing System) Foundation**	**Transfer Encoding**	
		Wherever data is shared between applications, such as email, Internet browsing, etc. or wherever data is displayed.	If the application transfers data through protocols or networks that strip the 8th bit, The application should encode, decode, and display the data correctly. If the product includes a help browser or web browser: 1. Browser should load files or pages in different encodings. 2. Browser should be properly display files or pages with multibyte text. 3. Once a file is loaded, is it possible to display it in another encoding? 4. Bookmarks and menus should show filenames with multibyte text correctly.
		Input	
			Application should correctly accept, parse and display non-ASCII input in all places where it is

		appropriate to enter non-ASCII text.
		As non-ASCII text is being entered, the backspace key should correctly delete the non-ASCII text.
		Application/Product should have the mechanism to input characters used for various languages.
		Shortcut key combinations should be accessible on international keyboards
	Output	
	Labels, menu items, text areas, canvases, HTML pages, etc.	All areas of the application should display all characters within the current locale.
		Your application should use the default fonts, or if explicit fonts are named, the fonts should be stored in an external resource file that can be configured for each locale.
		Printing out characters from the users' native languages should be proper.
		Application should work correctly on localized editions of operations systems that the product supports.

9 Migration to Unicode

According to Microsoft creating a new program based on Unicode is fairly easy. Unicode has a few features that require special handling, but you can isolate these in your code. Converting an existing program that uses code-page encoding to one that uses Unicode or generic declarations is also straightforward. Here are the steps to follow:

1. **Modify your code to use generic data types.** Determine which variables declared as char or char* are text, and not pointers to buffers or binary byte arrays. Change these types to TCHAR and TCHAR*, as defined in the Win32 file WINDOWS.H, or to _TCHAR as defined in the Visual C++ file TCHAR.H. Replace instances of LPSTR and LPCH with LPTSTR and LPTCH. Make sure to check all local variables and return types. Using generic data types is a good transition strategy because you can compile both ANSI and Unicode versions of your program without sacrificing the readability of the code. Don't use generic data types, however, for data that will always be Unicode or always stays in a given code page. For example, one of the string parameters to MultiByteToWideChar and WideCharToMultiByte should always be a code page-based data type, and the other should always be a Unicode data type.
2. **Modify your code to use generic function prototypes.** For example, use the C run-time call _tcslen instead of strlen, and use the Win32 API SetWindowText instead of SetWindowTextA. This rule applies to all APIs and C functions that handle text arguments.
3. **Surround any character or string literal with the TEXT macro.** The TEXT macro conditionally places an "L" in front of a character literal or a string literal definition. Be careful with escape sequences. For example, the Win32 resource compiler interprets L/" as an escape sequence specifying a 16-bit Unicode double-quote character, not as the beginning of a Unicode string.
4. **Create generic versions of your data structures**. Type definitions for string or character fields in structures should resolve correctly based on the UNICODE compile-time flag. If you write your own string-handling and character-handling functions, or functions that take strings as parameters, create Unicode versions of them and define generic prototypes for them.
5. **Change your build process.** When you want to build a Unicode version of your application, both the Win32 compile-time flag -DUNICODE and the C run-time compile-time flag -D_UNICODE must be defined.
6. **Adjust pointer arithmetic.** Subtracting char* values yields an answer in terms of bytes; subtracting wchar_t* values yields an answer in terms of 16-bit chunks. When determining the number of bytes (for example, when allocating memory for a string), multiply the length of the string in symbols by sizeof(TCHAR). When determining the number of characters from the number of bytes, divide by sizeof(TCHAR). You can also create macros for these two operations, if you prefer. C makes sure that the ++ and -- operators increment and decrement by the size of the data type. Or even better, use Win32 APIs CharNext and CharPrev.
7. **Add code to support special Unicode characters.** These include Unicode characters in the compatibility zone, characters in the Private Use Area, combining characters, and characters with directionality. Other special characters include the Private Use Area noncharacter U+FFFF, which

can be used as a placeholder, and the byte-order marks U+FEFF and U+FFFE, which can serve as flags that indicate a file is stored in Unicode. The byte-order marks are used to indicate whether a text stream is little-endian or big-endian. In plaintext, the line separator U+2028 marks an unconditional end of line. Inserting a paragraph separator, U+2029, between paragraphs makes it easier to lay out text at different line widths.

8. **Debug your port by enabling your compiler's type-checking.** Do this with and without the UNICODE flag defined. Some warnings that you might be able to ignore in the code page-based world will cause problems with Unicode. If your original code compiles cleanly with type-checking turned on, it will be easier to port. The warnings will help you make sure that you are not passing the wrong data type to code that expects wide-character data types. Use the Win32 National Language Support API (NLS API) or equivalent C run-time calls to get character typing and sorting information. Don't try to write your own logic for handling locale-specific type checking-your application will end up carrying very large tables!

9.1.1 Compiling Unicode Applications in Visual C++

By using the generic data types and function prototypes, you have the liberty of creating a non-Unicode application or compiling your software as Unicode. To compile an application as Unicode in Visual C/C++, go to Project/Settings/C/C++ /General, and include UNICODE and _UNICODE in Preprocessor Definitions. The UNICODE flag is the preprocessor definition for all Win32 APIs and data types, and _UNICODE is the preprocessor definition for C run-time functions.

10 Code Elements

10.1 Language identification and Negotiation BCP 47/RFC 5646

DEFINITION:

The language used when presenting information. In some contexts, it is possible to have information available in more than one language, or it might be possible to provide tools (such as dictionaries) to assist in the understanding of a language. Also, many types of information processing require knowledge of the language in which information is expressed in order for that process to be performed on the information; for example spell-checking, computer-synthesized speech, Braille, or high-quality print renderings.

GOALS:

Language Tags can be used by the Applications to deliver to users the most appropriate information. It is useful for accessibility, authoring tools, translation tools, font selection, page rendering, search, and scripting and is used by screen readers and accessibility as these applications are interested in output produced by them in different language mode.

CONCEPT OF MACRO LANGUAGES AND COLLECTIVE LANGUAGES:

Macro Languages: Some Languages are defined as macro languages covering either significantly different dialects or a net of very closely related languages. Two new terms were introduced in ISO 639-2 called Macro languages and Collective languages. The term Collective language is not continued in ISO 639-3. However, Macro language is still working.

Some of examples of Macro languages are: doi: is the ISO 639-3 language code for Dogri. There are 2 individual language codes assigned: dgo — Dogri, an individual language and xnr — Kangri, a dialect.

Collective Languages: A collective language code element is an identifier that represents a group of individual languages that are not deemed to be one language in any usage context. Collective languages and their ISO 639-2 codes are: bih - Bihari. This Language also has an ISO 639-1 code and him – Himachali. However, this is bad example because these two are not kept separate for the revision.

सत्यमेव जयते

FORMAT:

The Format of Language Tags is described in BCP 47 published by Internet Engineering Task Force (IETF). It describes the structure, content, construction, and semantics of language tags for use in cases where it is desirable to indicate the language used in an information object, how to register values for use in language tags and the creation of user-defined extensions for private interchange.

The Language Subtag Registry, maintained by IANA, lists the current valid public subtags. Language tags consist of only language subtag, or a language subtag and a region subtag. Subtags are not case sensitive, but the specification recommends using the same case as in the Language Subtag Registry, where region subtags are uppercase, script subtags are title case and all other subtags are lowercase. Syntax will be:

Language-Tag = langtag

/ Private use ; private use tag

/ grandfathered ; grandfathered registrations

Langtag = (language ["-" script] ["-" region] *("-" variant) *("-" extension) ["-" private use])

For Example, bn language code is used for Bengali language; Beng Script code is used for Bengali Script and IN is used as Region Code for India. So, to represent the whole Language tag we can declare it as Bn-Beng-IN (Bengali language uses Bengali Script in India).

11 Data Encoding and Byte Order Markers

- Consider using locale-based routines and further internationalization.
- For Windows 95, 98 and ME, consider using the Microsoft MSLU (Microsoft Layer for Unicode)
- Consider string compares and sorting, Unicode Collation Algorithm
- Consider Unicode Normalization
- Consider Character Folding

Unicode BOM Encoding Values

Name	UTF-8	UTF-16	UTF-16BE	UTF-16LE	UTF-32	UTF-32BE	UTF-32LE
Smallest code point	0000	0000	0000	0000	0000	0000	0000
Largest code point	10FFFF	10FFFF	10FFFF	10FFFF	10FFFF	10FFFF	10FFFF
Code unit size	8 bits	16 bits	16 bits	16 bits	32 bits	32 bits	32 bits
Byte order	N/A	<BOM>	big-endian	little-endian	<BOM>	big-endian	little-endian
Fewest bytes per character	1	2	2	2	4	4	4
Most bytes per character	4	4	4	4	4	4	4

Constant and Global Variables

ANSI	**Wide**	**TCHAR**
EOF	WEOF	_TEOF
_environ	_wenviron	_tenviron
_pgmptr	_wpgmptr	_tpgmptr

Data Types

ANSI	Wide	TCHAR
char	wchar_t	_TCHAR
_finddata_t	_wfinddata_t	_tfinddata_t
__finddata64_t	__wfinddata64_t	_tfinddata64_t
_finddatai64_t	_wfinddatai64_t	_tfinddatai64_t
int	wint_t	_TINT
signed char	wchar_t	_TSCHAR
unsigned char	wchar_t	_TUCHAR
char	wchar_t	_TXCHAR
	L	_T or _TEXT
LPSTR (char *)	LPWSTR (wchar_t *)	LPTSTR (_TCHAR *)
LPCSTR (const char *)	LPCWSTR (const wchar_t *)	LPCTSTR (const _TCHAR *)
LPOLESTR (For OLE)	LPWSTR	LPTSTR

For further details regarding data types and functions please refer: http://www.i18nguy.com/unicode/c-unicode.html

Most string operations for Unicode can be coded with the same logic used for handling the Windows character set. The difference is that the basic unit of operation is a 16-bit quantity instead of an 8-bit one. The header files provide a number of type definitions that make it easy to create sources that can be compiled for Unicode or the Windows character set.

For 8-bit (ANSI) and double-byte characters:
typedef char CHAR; // 8-bit character
typedef char *LPSTR; // pointer to 8-bit string

For Unicode (wide) characters:
typedef unsigned short WCHAR; // 16-bit character
typedef WCHAR *LPWSTR; // pointer to 16-bit string

The figure below shows the method by which the Win32 header files define three sets of types:

- One set of generic type definitions (TCHAR, LPTSTR), which depend on the state of the _UNICODE manifest constant.
- Two sets of explicit type definitions (one set for those that are based on code pages or ANSI and one set for Unicode).

With generic declarations, it is possible to maintain a single set of source files and compile them for either Unicode or ANSI support.

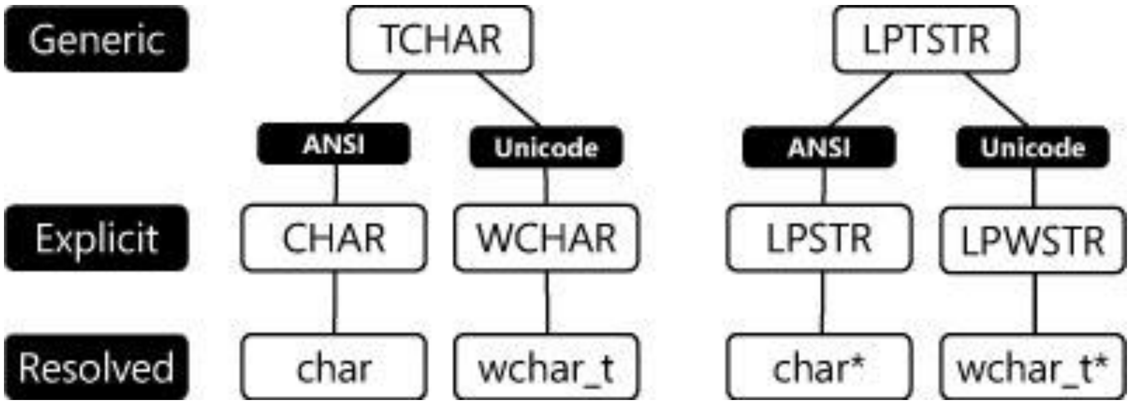


Figure 4: WCHAR, a new data type (source Microsoft/MSDN).

11.1 Tips and Tricks for C/C++/VC++

These are some small tips which we can use during coding :-

(1) Using CString in string copy function

```
wcscpy(wchar_t*,(const wchar_t*)(LPCTSTR)Cstring)
```

(2) For converting integer to string

```
char buffer[20];
int  i = 3445;
_itoa( i, buffer, 10 );
printf( "String of integer %d (radix 10): %s\n", i, buffer );
```

(3) To know the size of any file

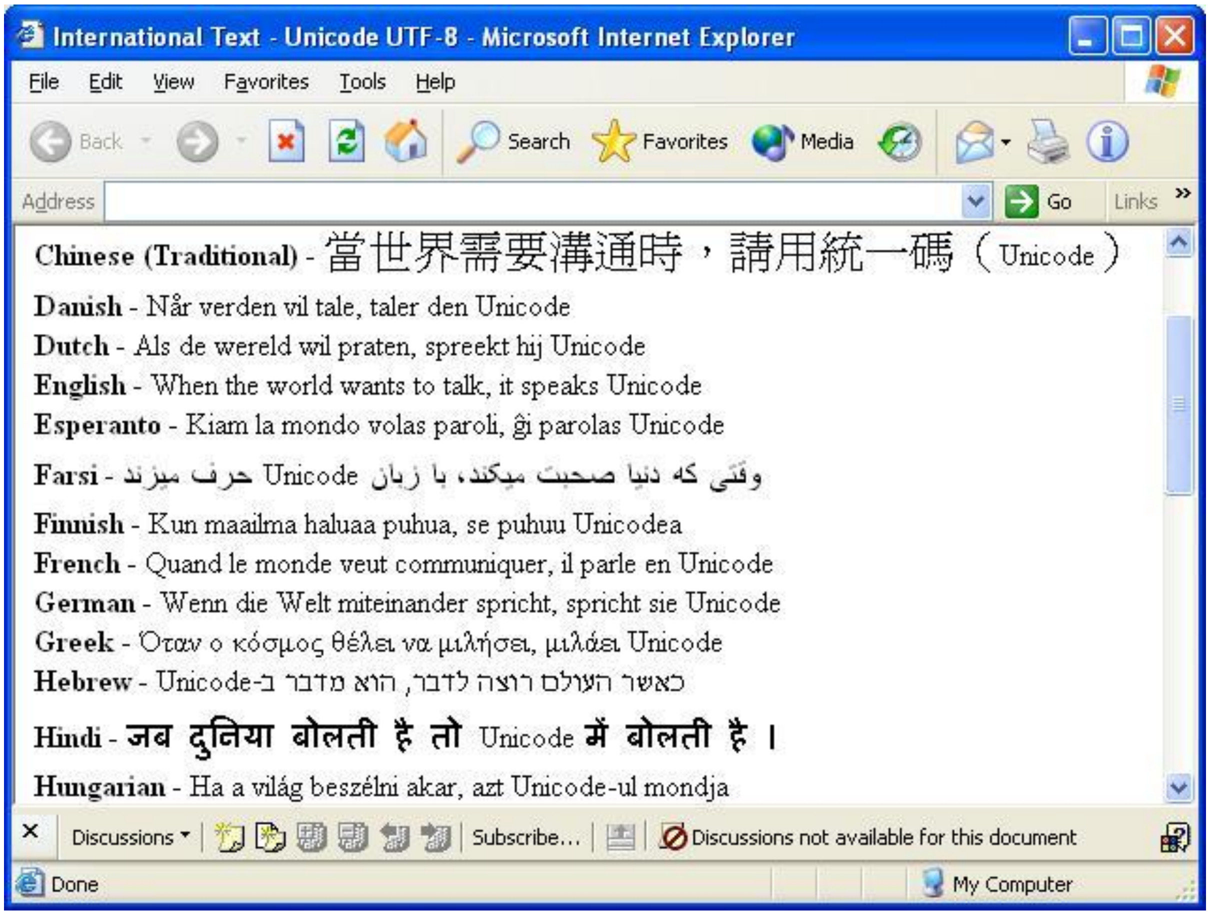
```
FILE *fpRead;
long  lFileSize;
if ((fpRead=_tfopen(_T(“c:\\test.txt”),_T("rb"))) == NULL)
        MessageBox (_T("Canot Open"),NULL,MB_OK);
else
{
        fseek(fpRead,0L,SEEK_SET);
        fseek(fpRead,0,SEEK_END);
        lFileSize =ftell(fpRead);
}
```

11.2 Encodings in Web Pages

Generally speaking, there are four different ways of setting the character set or the encoding of a Web page.

- With this approach, you can select from the list of supported code pages to create your Web content. The downside of this approach is that you are limited to languages that are included in the selected character set, making true multilingual Web content impossible. This limits you to a single-script Web page.
- Number entities can be used to represent a few symbols out of the currently selected code page or encoding. Let's say, for example, you have decided to create a Web page using the previous approach with the Latin ISO charset 8859-1. Now you also want to display some Greek characters in a mathematical equation; Greek characters, however, are not part of the Latin code page. Take, for instance, the Greek character Φ, which has the Unicode code-point U+03A6. By using the decimal number entity of this code point preceded by &# the character's output will be as follows: This is my text with a Greek Phi: Φ. and the output would be:This is my text with a Greek Phi: Φ. Unfortunately, this approach makes it impossible to compose large amounts of text and makes editing your Web content very hard.
- Unlike Win32 applications where UTF-16 is by far the best approach, for Web content UTF-16 can be used safely only on Windows NT networks that have full Unicode support. Therefore, this is not a suggested encoding for Internet sites where the capabilities of the client Web browser as well the network Unicode support are not known.

- This Unicode encoding is the best and safest approach for multilingual Web pages. It allows you to encode the whole repertoire of Unicode characters. Also, all versions of Internet Explorer 4 and later as well as Netscape 4 and later support this encoding, which is not restricted to network or wire capabilities. The UTF-8 encoding allows you to create multilingual Web content without having to change the encoding based on the target language.


International Text - Unicode UTF-8 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address Go Links
Discussions Subscribe... Discussions not available for this document
Done My Computer

Chinese (Traditional) - 當世界需要溝通時，請用統一碼（Unicode）
Danish - Når verden vil tale, taler den Unicode
Dutch - Als de wereld wil praten, spreekt hij Unicode
English - When the world wants to talk, it speaks Unicode
Esperanto - Kiam la mondo volas paroli, ĝi parolas Unicode
Farsi - وقتی که دنیا صحبت میکند، با زبان Unicode حرف میزند
Finnish - Kun maailma haluaa puhua, se puhuu Unicodea
French - Quand le monde veut communiquer, il parle en Unicode
German - Wenn die Welt miteinander spricht, spricht sie Unicode
Greek - Όταν ο κόσμος θέλει να μιλήσει, μιλάει Unicode
Hebrew - כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode
Hindi - जब दुनिया बोलती है तो Unicode में बोलती है ।
Hungarian - Ha a világ beszélni akar, azt Unicode-ul mondja

Figure: Example of a multilingual Web page encoded in UTF-8 (Source: Microsoft).

11.2.1 Setting and Manipulating Encodings

Since Web content is currently based on Windows or other encoding schemes, you'll need to know how to set and manipulate encodings. The following describes how to do this for HTML pages, Active Server Pages (ASP), and XML pages.

11.2.2 HTML pages

Internet Explorer uses the character set specified for a document to determine how to translate the bytes in the document into characters on the screen or on paper. By default, Internet Explorer uses the character set specified in the HTTP content type returned by the server to determine this translation. If this parameter is not given, Internet Explorer uses the character set specified by the meta element in the document, taking into account the user's preferences if no meta element is specified. To apply a character set to an entire document, you must insert the meta element before the body element. For clarity, it should appear as the first element after the head, so that all browsers can translate the meta element before the document is parsed. The meta element applies to the document containing it. This means, for example, that a compound document (a document consisting of two or more documents in a set of frames) can use different character sets in different frames. Here is how it works:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=<value>">
```

11.2.3 ASP.Net

1. Explicitly set the CurrentUICulture and CurrentCulture properties in your application. Do not rely on defaults.
2. Note that ASP.NET applications are managed applications and therefore can use the same classes as other managed applications for retrieving, displaying, and manipulating information based on culture.
3. Be aware that you can specify the following three types of encodings in ASP.NET:
    - requestEncoding specifies the encoding received from the client's browser.
    - responseEncoding specifies the encoding to send to the client browser. In most situations, this encoding should be the same as that specified for requestEncoding.
    - fileEncoding specifies the default encoding for .aspx, .asmx, and .asax file parsing.
4. Specify the values for the requestEncoding, responseEncoding, fileEncoding, culture, and uiCulture attributes in the following three places in an ASP.NET application:
    - In the globalization section of a Web.config file. This file is external to the ASP.NET application. For more information, see <globalization> Element.
    - In a page directive. Note that, when an application is in a page, the file has already been read. Therefore, it is too late to specify fileEncoding and requestEncoding. Only uiCulture, Culture, and responseEncoding can be specified in a page directive.
    - Programmatically in application code. This setting can vary per request. As with a page directive, by the time the application's code is reached it is too late to specify fileEncoding and requestEncoding. Only uiCulture, Culture, and responseEncoding can be specified in application code.
5. Note that the uiCulture value can be set to the browser accept language.

11.2.4 XML pages

All XML processors are required to understand two transformations of the Unicode character encoding: UTF-8 (the default encoding) and UTF-16. The Microsoft XML Parser (MSXML) supports more encodings, but all text in XML documents is treated internally as the Unicode UTF-16 character encoding.

The encoding declaration identifies which encoding is used to represent the characters in the document. Although XML parsers can determine automatically if a document uses the UTF-8 or UTF-16 Unicode encoding, this declaration should be used in documents that support other encodings.

For example, the following is the encoding declaration for a document that uses the ISO 8859-1 encoding (Latin 1):

```
< xml version="1.0" encoding="ISO-8859-1" >
```

11.2.5 Java

Application programming interfaces can be used to access localized messages, that is, translated versions of original text. These messages need to be accessed by applications at runtime to ensure that the correct locale-specific messages are used. Special APIs must be used to retrieve this text. For example, there are two distinct APIs for message handling on Solaris: the catgets() family, which is used with .msg files, and the gettext() family used with .po files. In Java, resource bundles can be used. These are basically Java classes; that is .java files. The API getString() is used to retrieve localized text from Messages the resource bundles.

11.2.6 PHP

Hypertext Preprocessor (PHP) is a scripting language. The gettext functions implement the NLS (Native Language Support) API which can be used to internationalize PHP based applications. [http://www.php.net/manual/en/intro.gettext.php] GNU gettext is an important step for the Translation Project, as it is an asset on which many other steps can be built. This bundle offers a lot of well integrated tools and documents to programmers, translators and end users. GNU gettext provides a framework in which various utilities and tools can be integrated which may produce multi-lingual messages.

11.2.6.1 Gettext

The Gettext PHP extension allows you to dynamically translate strings in your PHP code by using the gettext() function to get the appropriate translated string. If the string is not translated yet, the original one is used instead. A simple example follows:

```php
<?php
// I18N support information here
$language = 'en';
putenv("LANG=$language");
setlocale(LC_ALL, $language);

// Set the text domain as 'messages'
$domain = 'messages';
bindtextdomain($domain, "/www/htdocs/site.com/locale");
textdomain($domain);

echo gettext("A string to be translated would go here");
?>
```

11.2.6.1.1 Setting Up the Gettext Files

Gettext works by expecting a locale directory where all of the translated strings are kept, in the following structure:

```
/locale
    /en
        /LC_MESSAGES
            messages.po
            messages.mo
```

So in the case of a Web site, the locale directory would be inside the webroot. Or not; it's totally up to you, as thebindtextdomain() function showed above.

The directories you can create yourself, always remembering to create one language subdirectory for each language for which you wish to show translated strings. For instance, if you want to translate your site to Brazilian Portuguese (pt_BR), you would need to create a pt_BR subdirectory and assign the proper language code in your PHP code, like this:

```php
<?php
// The language code goes here
$language = 'pt_BR';
putenv("LANG=$language");
setlocale(LC_ALL, $language);

// ....
```

```
?>
```

So after creating the new pt_BR subdirectory, your directory structure would look somewhat like this:

```
/locale
    /en
        /LC_MESSAGES
            messages.po
            messages.mo
    /pt_BR
        /LC_MESSAGES
            messages.po
            messages.mo
```

After you have the directories all prepared, it's time to create the actual "pot" file, as it is usually referred to: themessages.po file. To do this, you will need to have PHP files that use the gettext() function to "mark" strings to be translated and use the xgettext command.

```
$ xgettext -n *.php
```

The line above will create a messages.po file, with some strings to be translated. It will look close to the following:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2002-04-06 21:44-0500\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: gettext_example.php:12
msgid "A string to be translated would go here"
msgstr ""
```

This file contains all of the strings found inside gettext() calls, and it is used by the translators of the respective languages to translate the application (or Web application, in our case).

Distributing the Pot File

Ok, so now that you have your pot file with the strings that need to be translated, you need to distribute it to your translators. In a successful open source project, you would have several different volunteers that take care of the translation of your user interface messages. In this case, you would send an email to the development mailing list and tell the volunteers that the next release would go out on the X date and that the following list of languages need to be updated.

That's, of course, in a successful project. The normal situation, though, would have you, the project leader, to do the most important translations yourself and wait for contributions from volunteers. In a Web site or a Web application, this is usually not the case. That is, you normally wouldn't have a team of volunteers working on your personal Web site, but the picture is different for a community Web site like themes.org, for instance.

In any case, either you or the volunteers will translate the pot file and then you will need to convert the file into a binary file that Gettext actually "understands." For that you would use the following command:

```
$ msgfmt messages.po
```

The line above will create a messages.mo file, which you should save in the appropriatelocale/<LANG_CODE>/LC_MESSAGES/ ng strings y.

1.1.1 Managing Evolving Pot Files

Now think about all of this for a moment -- you have a Web site that is constantly evolving, with new features being added or inconsistencies being removed, and your strings to be translated are also changing. New ones are added, old ones are modified, and so on.

So how do you manage multiple versions of the pot files? In a usual situation, you will have a messages.po that is completely translated for a specific language, and a new file with the new strings to be translated. The problem lies here: since Gettext doesn't work in any other way, this new file will look just like the example above -- it will be empty.

The question is how to merge the files in a way that keeps the already- translated strings while adding the new untranslated ones. The answer is provided by the msgmerge Gettext utility. An example of its use would be the following sequence of commands:

```
$ ls
example.php
$ xgettext -n *.php
$ ls
example.php   messages.po
// ...
// Translates the messages.po file now
// ...
$ msgfmt messages.po
$ ls
example.php   messages.po   messages.mo
// ...
// Changes the example.php file
// ...
$ mv messages.po old.po
$ xgettext -n *.php
$ ls
example.php   messages.po   messages.mo   old.po
$ msgmerge old.po messages.po --output-file=new.po
$ ls
example.php   messages.po   messages.mo   new.po    old.po
// ...
// Translates the new.po file
// ...
$ msgfmt new.po
```

11.2.7 PYTHON

In order to provide multilingual messages for your Python programs, you need to take the following steps:

1. prepare your program or module by specially marking translatable strings
2. run a suite of tools over your marked files to generate raw messages catalogs
3. create language specific translations of the message catalogs
4. use the gettext module so that message strings are properly translated

In order to prepare your code for I18N, you need to look at all the strings in your files. Any string that needs to be translated should be marked by wrapping it in _('...') — that is, a call to the function _(). For example:

```
filename = 'mylog.txt'
message = _('writing a log message')
fp = open(filename, 'w')
fp.write(message)
fp.close()
```

In this example, the string 'writing a log message' is marked as a candidate for translation, while the strings 'mylog.txt' and 'w' are not.

For further details please refer http://docs.python.org/2/library/gettext.html

11.2.8 XML

All XML processors are required to understand two transformations of the Unicode character encoding: UTF-8 (the default encoding) and UTF-16. The Microsoft XML Parser (MSXML) supports more encodings, but all text in XML documents is treated internally as the Unicode UTF-16 character encoding.

The encoding declaration identifies which encoding is used to represent the characters in the document. Although XML parsers can determine automatically if a document uses the UTF-8 or UTF-16 Unicode encoding, this declaration should be used in documents that support other encodings.

For example, the following is the encoding declaration for a document that uses the ISO 8859-1 encoding (Latin 1):

The example is enclosed as **Annexure 5.**

Requirement of XML for Indian Language:

Internationalization Tag Set (ITS)

- ITS is a technology to easily create XML which is internationalized and can be localized effectively.

ITS for Schema developers

- User will find proposals for attribute and element names to be included in their new schema (also called "host vocabulary").
- It leads to easier recognition of the concepts represented by both schema users and processors.

Main Attributes

- Defining mark-up for natural language labelling.
- Defining mark-up to specify text direction.
- Indicating which elements and attributes should be translated.
- Providing information related to text segmentation.
- Defining mark-up for unique identifiers.
- Defining mark-up for notes to localizers.
- Working with multilingual documents.

सत्यमेव जयते

Comparison of Database Models

S.No	**Relational Database**	**XML Database**
1	A relational database contains tables.	An XML database contains collections.
2	A relational table contains records with the same schema.	A collection contains XML documents with the same schema.
3	A relational record is an unordered list of named values.	An XML document is a tree of nodes.
4	A SQL query returns an unordered set of records.	An XQuery returns an ordered sequence of nodes.

Table : Comparison of databases

Pros of Relational Databases

1. SQL has four fundamental operations -- SELECT, INSERT, UPDATE, and DELETE -- as well as some lesser commands for creating and dropping tables and users
2. Manipulation of Database is easily implemented using SQL in Relational Databases
3. **Multiple views of the same data**- A related advantage of storing data in a database is that doing so enables multiple views of the same content
4. **Indexing** - It can jump right to the element with a certain ID rather than having to walk the tree looking for it,
5. The database has essentially *pre-parsed* each document when storing it. Therefore, it doesn't need to check each document that the query has accessed for well-formedness, or build an object model representing that document. All these details are already inside the database in a form the query engine can use.

Cons of Relational Databases

1. **Performance**- The database does more work when it adds or modifies a document, stores the result of that work, and then uses the results to run lightning-fast queries. If queries are significantly more frequent than insertions and updates -- as they tend to be in many applications -- then the extra cost paid to put documents into the database is more than earned back when retrieving them.

Pros of XML Databases

1. In most XML databases, the fundamental unit is the XML document, which roughly corresponds to a record in a traditional database.
2. One big advantage of a native XML database is that it can run queries that combine (or *join,* in SQL parlance) information contained in multiple XML documents.
3. What SQL is to relational databases, XQuery is to native XML databases.
4. **Everything is in one place**- The most important advantage of a native XML database is simply that it keeps all your content in one easily searched, easily managed place.
5. Queries over a well-designed, well-implemented native XML database are simply faster than queries over documents stored in a file system
6. XML databases use a lot of other tricks to optimize performance. A few of these (smart query rewriting, for example) are available to tools that aren't backed by a database
7. Can retrieve the original, unparsed document, character-per-character or even byte-per-byte

Cons of XML Databases

1. XQuery does not do as much as SQL does.
2. XQuery lets you retrieve information from an XML database, but that's it.
3. No Indexing – One needs to walks down the tree to get the specific element
4. It can't add documents to the database, delete documents from the database, modify existing documents, or do anything else, which is a pretty gaping hole in its capabilities
5. Multiple views of the same data is hardly possible

Building XQuery-powered applications with PHP and Zorba

Add XQuery support to PHP with Zorba

Zorba is an open-source, robust, and standards-compliant XQuery processor. The Zorba extension in PHP provides an API to Zorba functions from within PHP, and thereby allows developers to add sophisticated XQuery processing to their PHP/XML applications.
Zorba is currently available for Mac OS X, Linux®, and Microsoft® Windows® platforms as an open-source project under the terms of the Apache License, and it is supported by a number of organizations.

SimpleXML does include some basic XQuery support, it's not (nor is it supposed to be) a full-fledged XQuery processor. For that, you have to look further afield, to the PHP Extension Community Library (PECL), which contains an extension for the Zorba XQuery processor. This extension can be used to add full-featured XQuery processing support to PHP.
Processes are as follows:

1) **Installing Zorba**
2) **Understanding basic usage**
   How to use Zorba to process XQuery queries from within a PHP script:

a) **Assigning and displaying variables**

Code Snippet: XML Database

The example is enclosed as **Annexure 6.**

For more details :
http://www.zorba-xquery.com/html/demo
http://www.w3.org/TR/xquery
http://www.zorba-xquery.com/html/index
http://www.w3schools.com/xquery/default.asp

11.2.9 XHTML

XHTML (Extensible Hypertext Mark-up Language) is a family of XML mark-up languages that mirror or extend versions of the widely used Hypertext Mark-up Language (HTML), the language in which web pages are written.

While HTML (prior to HTML5) was defined as an application of Standard Generalized Mark-up Language (SGML), a very flexible mark-up language framework, XHTML is an application of XML, a more restrictive subset of SGML. Because XHTML documents need to be well-formed, they can be parsed using standard XML parsers—unlike HTML, which requires a lenient HTML-specific parser.

XHTML 1.0 became a World Wide Web Consortium (W3C) Recommendation on January 26, 2000. XHTML 1.1 became a W3C Recommendation on May 31, 2001. XHTML5 is undergoing development as of September 2009, as part of the HTML5 specification.

Current Versions of XHTML:-

XHTML 1.0

This was the first XHTML standard to be released. It was created to redesign HTML 4.0 to act more like an XML file. To make the change from HTML 4.0 to XHTML 1.0, there were three standards: **strict**, **transitional** and **frameset**.

- Strict was the standard that accepted the full XHTML standard.
- Transitional was used by developers to either make it easier to migrate over from HTML 4.0 or to allow certain standards from HTML version 4.01 Transitional in the XHTML file.
- Frameset was very simply the standard that allowed frames in the XHTML file.

XHTML has the capability to be used with XML files and applications that work with XML files.

XHTML 1.1

XHTML 1.0 was made with the intention of allowing developers to move from HTML standards to XHTML standards more easily.

XHTML 2.0

XHTML 2.0 was drafted between August 2002 and July 2006. It was designed to supersede HTML 4.0 and XHTML 1.1. In July 2009, World Wide Web Consortium announced plans to discontinue work on XHTML 2.0 by the end of the year in order to focus its effort on HTML 5. HTML 5 is expected to include a XML scheme in place of XHTML 2.0 currently called XHTML 5.

Future version can be tracked from W3C.

Converting a document from HTML 4.0 to XHTML 1.0:-

Converting a document from HTML 4.0 to XHTML 1.0 will not be a totally painless affair - some changes WILL need to be made.

- **An XHTML document MUST be well-formed XML**

  It must conform to basic XML syntax. If it does not, the XML parser does not have an obligation to continue processing the document. Unlike today's HTML parsers, an XML parser will not try to recover and "guess" what you meant if the syntax is incorrect.

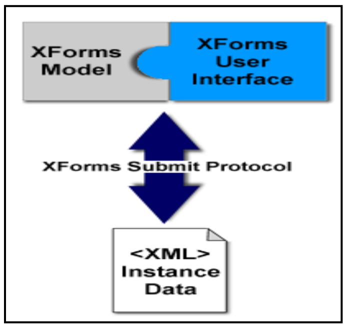
- **<Html> MUST be the top-level element.**
  Not changes from HTML, but there are quite a few documents out there that neglect this important point.
- **Element and attribute names MUST be in lower case**
  HTML is not case-sensitive; but XML is.
- **Attribute values MUST be quoted**
- **End tags are required for non-empty elements. .**
  They are no longer optional. .
  Affected Elements: base font, body, colgroup, dd, dt, head, html, li, p, rt, spacer, tbody/thead/tfoot, th/td, tr
- **All empty elements must use the XML "empty tag" syntax**
  XML empty elements are explicitly closed with a trailing forward slash ("/") before the end bracket (eg: <br> becomes <br />)
  Affected Elements: area, base, bgsound, br, col, frame, hr, img, input, isindex, keygen, link, meta, option, param, wbr
- **XML does not allow attribute minimization.**
  Stand-alone attributes must be expanded (eg: <td nowrap>cell</td> becomes <td nowrap="nowrap">cell</td>)
- **Whitespace handling in attribute values is different in XML.**
  Leading/trailing spaces are truncated, and multiple spacing characters within the attribute value are collapsed to single spaces.
- **Script sections should be wrapped in XML CDATA sections**
- **SGML DTD exclusions are not possible in XML, but they should still be observed as "good practice".**
  Not allowed to nest within themselves: a, button, form, label
  Pre exclusions: big, img, object, small, sub, sup
  Button exclusions: fieldset, form, iframe, input, label, select, text area

11.2.10 X-Forms

- X-forms is XML based Open Standard for e-forms domain
- Separates Data, Logic and Presentation
- Uses Declarative language by avoiding scripting
- Design started from scratch Based on totally new model
- Not compatible with HTML Forms

X-forms: MVC

- "Model" describes form data, constraints and submission.
- "View" describes what visual controls appear in the form, how they are grouped together and what data they are bound to. CSS can be used to describe a form's appearance
- "Controller" that handles event input and the mappings between the two



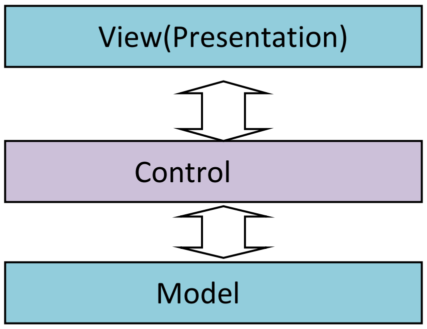


Fig : X-forms framework

X-forms Framework:

- Based on MVC paradigm
- X-Forms model defines
  -What the form is
  -What data it contains
  -What it should do
- X-Forms user interface defines the form controls and how they should be displayed
- Instance contains the data and initial form values
- Submit protocol defines how X-Forms sends and receives instance data
- Bindings bind properties to instance data

Comparison between X-forms 2.0 and Web forms 2.0

X-Form 2.0	Web Form 2.0
1. Support for structured form data and incompatible to legacy browsers.	1. Backwards compatibility to legacy browsers.
2. Advanced forms logic without server round-tripping	2. Ease of authoring for authors who are familiar with commonly used languages such as HTML and ECMAScript but have limited knowledge about XML, data models, etc.
3. Dynamic access to server data sources during form execution	3. Basic data typing, providing new controls for commonly used types so that authors do not

	need to repeatedly design complicated widgets such as calendars.
4. Seamless integration with other XML tag sets	4. Simpler validation on the client side
5. Multiple forms per page, and pages per form	5. Dynamic addition of controls (repeating structures) on the client side without scripting.
6. Richer user interface to meet the needs of business, consumer and device control applications	6. XML submission
7. Suspend and Resume capabilities	7. WHATWG specification
8. No In-Line Exposure	8. Automatic In-Line Exposure

Table: Comparison

Unicode with X-forms:

1) X-Forms use XML as its underlying serialization form.
2) Any browser uses XML while receiving or sending X-forms from server to client and vice-versa
3) XML documents are Unicode.
4) They can be stored in other encodings, but the translation between those encodings and Unicode is simple and deterministic.
5) This makes it possible to handle data in various international as well as national languages.

Internationalisation and Localisation in X-forms

1) True internationalization and localization go beyond merely letting people type their names and addresses using non-ASCII letters.
2) It's also necessary to allow forms in languages such as Hebrew and Arabic to flow right to left rather than left to right.
3) To handle this again, the separation of presentation from content played an important role.
4) Since presentation and contents are totally separate in X-Forms so fields can be laid in the direction, based on the requirement of local renderer.

11.2.11 HTML 5.0

HTML4 became a W3C Recommendation in 1997. While it continues to serve as a rough guide to many of the core features of HTML, it does not provide enough information to build implementations that interoperate with each other and, more importantly, with a critical mass of deployed content. The same goes for XHTML1, which defines an XML serialization for HTML4, and DOM Level 2 HTML, which defines JavaScript APIs for both HTML and XHTML. HTML5 will replace these documents.

HTML5 defines an HTML syntax that is compatible with HTML4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML4.W3C HTML5 defines:

1. A single language called HTML5 which can be written in HTML syntax and in XML syntax.
2. A detailed processing model to foster interoperable implementations.
3. How to Improves mark-up for documents.
4. APIs for emerging idioms, such as Web applications.
5. Detailed parsing rules (including "error handling")

For more details : http://www.w3.org/TR/2011/WD-html5-diff-20110525/

12 Cascading Style Sheets (CSS)

12.1 How to Use @font-face

With the increasing browser capabilities beautiful typography is not a challenge any more. @font-face can be used to create beautiful typography. Using @font-face is also very simple and easy. It requires few lines of CSS and the declaration of font family like in any other font on the web.

The following code snippet is taken from: http://boldperspective.com/2011/how-to-use-css-font-face/

```
body { font-family: web-font, fallback-fonts; }
strong { font-family: web-font-bold; }
em { font-family: web-font-italic; }

@font-face {
    font-family: 'web-font';
    src: url('web-font.eot?') format('eot'),
         url('web-font.woff') format('woff'),
         url('web-font.ttf') format('truetype'),
         url('web-font.svg') format('svg');
    font-weight: normal;
    font-style: normal;
}
@font-face {
    font-family: 'web-font-bold';
    src: url('web-font-italic.eot?') format('eot'),
         url('web-font-italic.woff') format('woff'),
         url('web-font-italic.ttf') format('truetype'),
         url('web-font-italic.svg') format('svg');
    font-weight: bold;
    font-style: normal;
}
@font-face {
    font-family: 'web-font-italic';
    src: url('web-font-bold.eot?') format('eot'),
         url('web-font-bold.woff') format('woff'),
         url('web-font-bold.ttf') format('truetype'),
         url('web-font-bold.svg') format('svg');
    font-weight: normal;
    font-style: italic;
}
```

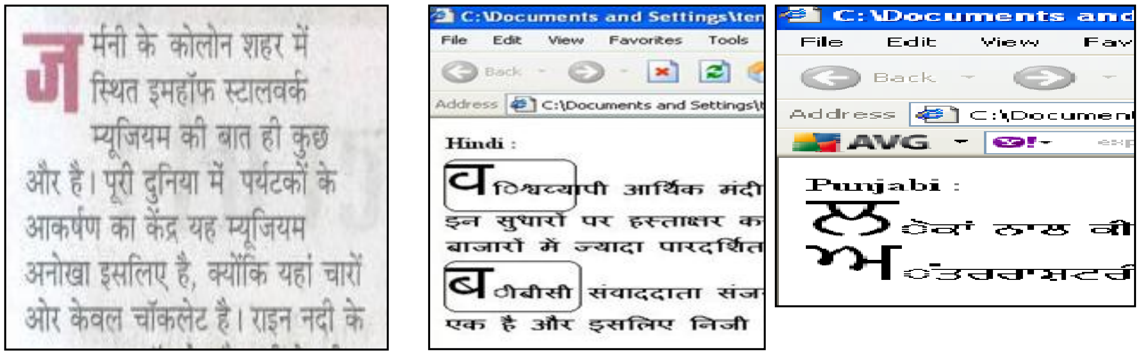
12.2 CSS rendering Issues in Indic Script

There are many rendering issues associated with the application of CSS on an Indic script. The following is the list of few of the issues:

(i) Styling of first letter pseudo-element Issues in Indian script – Hindi

The first-letter pseudo-element represents the first letter of The first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line. It allows that first letter to be styled individually, without markup. It may be used for "initial caps" and "drop caps“

For more details: www.w3.org/TR/CCS2/Selector.html


C:\Documents and Settings\ten
File Edit View Favorites Tools
Back
Address C:\Documents and Settings\
C:\Documents and
File Edit View Fav
Back
Address C:\Documen
AVG

ज र्मनी के कोलोन शहर में स्थित इमहॉफ स्टालवर्क म्यूजियम की बात ही कुछ और है। पूरी दुनिया में पर्यटकों के आकर्षण का केंद्र यह म्यूजियम अनोखा इसलिए है, क्योंकि यहां चारों ओर केवल चॉकलेट है। राइन नदी के

Hindi :
व िश्वव्यापी आर्थिक मंदी
इन सुधारों पर हस्ताक्षर क
बाजारों में ज्यादा पारदर्शित
ब ीबीसी संवाददाता संज
एक है और इसलिए निजी

Punjabi :
ਲ ੋਕਾਂ ਨਾਲ ਕੀ
ਅ ੰਤਰਰਾਸ਼ਟਰੀ

Fig 5: Example of first cap in Indian languages

This example is showing a possible rendering of an initial cap. In this example the computed height or width of the first letter is different, that is why it is not showing first character properly. (IE, Opera)

सत्यमेव जयते

(ii) **Bullets and Numbers**

- Number schemes/ bulleting needs to be supported in Indian languages as well. Some standards however need to be provided to those developing CSS so that by default user could have the facility to use bulleting in his own Indic languages.
- The word processors are sometimes used by the user to develop pages for the web. Therefore standards are must. In most application Devanagari order is followed for languages sharing the script, which unfortunately is not the correct thing while deciding on sorting/collation for Indic languages. Number schemes to be supported in Indian languages also.
- In case of Bangla and Assamese language the bulleting should be like as in below In case of numeric bulleting the digit of the Bangla scripts like ১,২,৩,৪,৫,৬,৭,৮,৯,০ is to be used.
- For bulleting by alpha numeric the consonant of Bangla scripts are used. The Bangla vowels are not used in bulleting purpose.

he proposed bulleting for Hindi language is shown in the table given below.

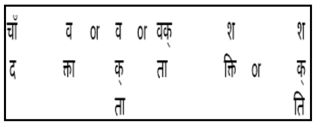
Character	Code Points
क	0915
ख	0916
....	0938
ह	0939
कक	0915+0915
कख	0915+0916
कग	0915+0917
....	....
कह	0915+0939
खक	0916+0915
खख	0916+0916
....	....
खह	0916+0939
....	....
हह	0939+0939
ककक	0915+0915+0915
ककख	0915+0915+0916
....	....

ककह	0915+0915+0939
खखक	0916+0916+0915
खखख	0916+0916+0916
…………	………………
खखह	0916+0916+0939
………….	……………
हहह	0939+0939+0939

Table 6: Proposed Hindi bulleting

(iii) Vertical & Horizontal Alignment arrangements of characters

Presentation / Styling issues: Vertical arrangement of characters If some string is written in vertical mode, then writing each character on a new line may not be suitable.



ধা রা বা হি ক উ প ন্যা স
পা ল্টা হা ও য়া

In case of horizontal alignment of characters, the space is given between the every character in case of English. But in case of Indian language like Bangla, Assamese etc the space may give not in every character but after some portion of the character sequence as in above figure.

Horizontal justification in HTML page in Indic languages, does not works on few browsers. It works well on IE6.0 but not on Firefox 2.

(iv) Underlining of the characters

There is some examples of Indian languages in which Matra's are not readable.

Underlining of characters:

Hindi - अन्य भाषाओं में भी अनुवाद

Punjabi ਗੁਰੂ

Bengali: তাই পুরোনো আর্কাইভ একটু ওলট পালট।

Guajarati - સરદાર ગુર્જરી

सत्यमेव जयते

Marathi- मराठी मुला मुलींची नावे

Tamil- நீரிற்குமிழி யிளமை நிறைசெல்வம்
நீரிற் சுருட்டு நெடுந்திரைகள் - நீரில்
எழுத்தாகும் யாக்கை நமரங்கா ளென்னே
வழுத்தாத தெம்பிரான் மன்று

Telugu - శ్రీశైలం ప్రాజెక్ట్‌పై TV9 ప్రోగ్రాం " డ్యామ్ ఇన్ డేంజర్ " పార్ట్ - 2

Proposed Solution for CSS issues

ABNF valid segmentation (Aksara): Proposed Solution for CSS issues such as First drop letter, Vertical & Horizontal arrangements, text segmentation, Line breaking etc is shown below:

Definition of the Indic Aksara

V[m]|{C[N]H}C[N](H|[v][m])

where V (upper case) is any independent vowel, m is any vowel modifier (Devanāgari Anusvāra, Visarga, Candrabindu), C is any consonant (with inherent vowel), N is Nukta, H is halant or Virāma and v (lower case) is any dependent vowel sign (mātrā). Following Conventions are used in the syntax: {} is enclosing items which may be repeated one or more times, [ ] is enclosing items which may not be present and | is separates items, out of which only one can be present.

The definition given above in the form of a regular expression may be paraphrased as follows:

• any independent vowel is an akṣara, e.g.
अ, ई, उ

any independent vowel followed by anusvāra, candrabindu or visarga is an akṣara, e.g.
अं ,उँ ,आः

zero or more consonant(+Nukta)+virāma sequences followed by a consonant (+Nukta) is an akṣara, e.g.
र्त ,त्र्स ,त्स्र्न ,त्स्न्र्य ,फ़्क़   वफ़्क़

• zero or more consonant(+Nukta)+virāma sequences followed by a consonant (+Nukta) followed by a virāma is an akṣara, e.g.
र्त् ,स्त् ,फ़्क़्

• zero or more consonant+(Nukta)+virāma sequences followed by a consonant (+Nukta) followed by a vowel sign is an akṣara, e.g.

तर्ा ,त्स्न्यर्ा ,फ़्जी लफ़्ज़ी

• zero or more consonant+(Nukta)+virāma sequences followed by a consonant (+Nukta) followed by an anusvāra or candrabindu or visarga is an akṣara, e.g.

स्तं ,स्त्रँ ,स्तः ,फ़्ज़ँ

• zero or more consonant+(Nukta)+virāma sequences followed by a consonant (+Nukta) followed by a vowel sign and an anusvāra or candrabindu or visarga is an akṣara, e.g.

त्स्न्यर्ाः त्स्न्युं ,त्स्न्युँ ,फ़्ज़ें

• nothing else is an akṣara.

13 Using Unicode In Databases For Multilingual Data

13.1 MS SQL:

1. Use nchar, nvarchar and ntext data types to store Indic/Unicode data
2. Precede all Unicode string with a prefix N (capital N – case sensitive) when dealing Unicode string constants

   e.g. SELECT * FROM TeluguDictionary WHERE (Telugu like N'%అక్క%')

INSERT INTO TeluguDictionary VALUES ('akkadi', N'అక్కడి')References

3. Note that in Indian Languages several words have multiple correct spellings and alternate representation forms. The Unicode data requires normalization (for details please refer section: 3.1.3.1).

e.g. Alternate representation forms: हिंदी हिन्दी विट्ठल विठ्‌ठल

4. Also note that IL numerals are not mapped to English numerals. So a MS-SQL query will give different result than the query using English numerals

e.g. select * from trains_table where train_no ='5312‘ ;

select * from trains_table where train_no ='५३१२‘ ;

So for correct results, you must map it to the English numerals.

Note:

- Compared to English, Indian language data takes more space in text fields. Fields of Unicode data therefore should have adequate length in case length limitation has been set in a text field.
- The design of the application will influence the choice between storing Indian language data in separate columns or in separate table.

13.2 Other Databases:

13.2.1 MySQL Data Types - NVARCHAR

Versions: MySQL 5.x, 4.x and 3.23

MySQL - NVARCHAR	
Syntax	**NVARCHAR**(n)
Data	Variable-length character data in the predefined character set - UTF8
Parameters	n is the maximum number of characters, optional
Range	1 ⇐ n ⇐ 21845 (65535 bytes is the maximum row size shared among all columns)
Default	n is mandatory
Trailing Spaces	Not removed when the value is stored and retrieved. But trailing spaces are insignificant in comparisons, primary and unique keys
Error Handling	Exceeding data truncated and a warning is generated if strict SQL mode is not enabled, otherwise in case of non-space characters truncation, an error is raised
Storage Size	Actual entry length
Synonyms	NATIONAL VARCHAR, NATIONAL CHAR VARYING, NATIONAL CHARACTER VARYING
Standards	ANSI SQL

13.2.2 MySQL NVARCHAR - Equivalents in Other Databases

Database	**Data Type and Conversion**
Oracle	**NVARCHAR2(n)**, 1 ⇐ n ⇐ 4000/charsize, Unicode - UTF-8 or UTF-16
SQL Server	**NVARCHAR(n | max)**, 1 ⇐ n ⇐ 8000, 2G if *max* is specified, Unicode - UCS-2
PostgreSQL	**VARCHAR(n)**, 1 ⇐ n ⇐ 1G
Sybase ASE	**NVARCHAR(n)**, 1 ⇐ n ⇐ pagesize/@@ncharsize (page size is 2K, 4K, 8K or 16K); **UNIVARCHAR(n)**, 1 ⇐ n ⇐ pagesize/@@unicharsize, Unicode
Informix	**NVARCHAR(n,r)**, 1 ⇐ n ⇐ 255, **LVARCHAR(n)**, 1 ⇐ n ⇐ 32739
HP Neoview	**NVARCHAR(n)**, 1 ⇐ n ⇐ (32708 - size of other columns)
Ingres	**NVARCHAR(n)**, 1 ⇐ n ⇐ 16000, Unicode - UTF-8

Related Data Types in MySQL

Data Types				
Fixed-length character data	**CHAR(n)**	**NCHAR(n)**		
Variable-length character data	**VARCHAR(n)**	**TINYTEXT**	**TEXT**	**MEDIUMTEXT**
Character large objects	**LONGTEXT**			

14 Need of W3C Validation services for Indian websites

Validating Web documents is an important step which can dramatically help improving and ensuring their quality and it can save a lot of time and money (read more on why validating matters). Validation is, however, neither a full quality check, nor is it strictly equivalent to checking for conformance to the specification.

Website Errors

The Web page errors that are generated using Web program are considered to identify the measures for quality of Website design. These errors are further divided into major and minor errors using statistical techniques.

1. **Major error:**

   The major errors directly affect the quality of Web site design and developers must concentrate on this category of errors and these should be eliminated. The major errors include: broken links, document type declaration errors, applet usage errors, server connectivity errors, image load errors, frames tag usage errors and title tag with no keyword errors. The major errors are proportional to the down load time of the Web pages. If major errors are minimized then down load time will be automatically reduced and hence it leads to the better quality.

2. **Minor errors:**

   The minor errors are HTML tag errors and these may cause incorrect display of some components of Web pages. The minor errors include: table tag errors, body tag errors, image tag errors, head tag errors, font tag errors, script tag errors, style tag errors, form tag errors, link tag errors and other tag errors. The developers must be attentive so that Web pages can be properly designed with appropriate HTML tags.

Evaluating Qualitative Measures for improved Website Design:

The errors that are found in Websites' of various lead to the necessity of qualitative measures for effective Website design [8]. The head tag errors (HTE), font tag errors (FoTE) and body tag errors (BTE) identify the problems in the text elements of web page. Thus Text formatting measures are to be evaluated. The image tag error (ITE), body tag errors (BTE) and image load errors related to image identifies the errors in display of images and hence Graphic element measures to be evaluated. The table tag errors (TTE), frame tag errors (FTE), style tag errors (StTE), font tag errors (FoTE), frame tag usage errors and document type declaration errors cause the invention of page formatting measures. Link Tag Errors (LTE) and broken links identify the need of link formatting measures. The form tag errors (FmTE), script tag errors (STE) and title tag with no keyword errors identify the need of page performance measure. The script tag errors (STE) applet usage errors, server connectivity errors and broken link errors contribute the need of Website architecture measure.

14.1 CSS Validator

What is CSS Validator and why we need it?

The W3C CSS Validation Service is free software created by the W3C to help Web designers and Web developers to check Cascading Style Sheets (CSS). It can be used on this free service on the web, or downloaded and used either as a java program, or as a java servlet on a Web server.

If you are a Web developer or a Web designer, this tool will be an invaluable ally. Not only will it compare your style sheets to the CSS specifications, helping you find errors, typos, or incorrect uses of CSS, it will also tell you when your CSS poses some risks in terms of usability.

What does “Valid CSS” mean? Which version of CSS does this Validator use?

According to the CSS 2.1 Specification: The validity of a style sheet depends on the level of CSS used for the style sheet. […] valid CSS 2.1 style sheet must be written according to the grammar of CSS 2.1. Furthermore, it must contain only at-rules, property names, and property values defined in this specification

By default, this Validator checks style sheets against the grammar, properties and values defined in the CSS 2.1 specification, but other CSS profiles can be checked against by using the options.CSS is an evolving language, and it is considered by many that “CSS” is a single grammar (the one defined in the latest specification) with a number of properties and acceptable values defined in various profiles. In a future version of this Validator, the default behaviour may be to check style sheets against that latest “CSS grammar” and the cloud of all standardized CSS properties and values.

14.2 *XHTML Validator*

The Mark up Validator is a free service by W3C that helps check the validity of Web documents.Most Web documents are written using mark up languages, such as HTML or XHTML. These languages are defined by technical specifications, which usually include a machine-readable formal grammar (and vocabulary). The act of checking a document against these constraints is called validation, and this is what the Mark up Validator does.

Validating Web documents is an important step which can dramatically help improving and ensuring their quality and it can save a lot of time and money (read more on why validating matters). Validation is, however, neither a full quality check, nor is it strictly equivalent to checking for conformance to the specification.

This Validator can process documents written in most mark up languages. Supported document types include the HTML (through HTML 4.01) and XHTML (1.0and1.1) family, MathML. SMIL and SVG (1.0 and 1.1, including the mobile profiles). The Mark up Validator can also validate Web documents written with an SGML or XML DTD, provided they use a proper document type declaration.

This Validator is also An HTML validating system conforming to International Standard ISO/IEC 15445—HyperText Mark up Language, and International Standard ISO 8879—Standard Generalized Mark up Language (SGML) – which basically means that in addition to W3C recommendations, it can validate according to these ISO standards.

Before an XHTML file can be validated, a correct DTD must be added as the first line of the file.

The Strict DTD includes elements and attributes that have not been deprecated or do not appear in framesets:

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//HI"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
```

The Transitional DTD includes everything in the strict DTD plus deprecated elements and attributes:

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//HI"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
```

The Frameset DTD includes everything in the transitional DTD plus frames as well:

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//HI"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"
```

14.3 Mobile OK Validator

The W3C MobileOK Checker is a free service by W3C that helps check the level of mobile-friendliness of Web documents, and in particular assert whether a Web document is MobileOK.

To understand why checking a Web document for mobile-friendliness really matters, it is probably worth emphasizing a few points about the so-called mobile world. Compared to a regular desktop computer, a mobile device may be regarded as limited at first glance: smaller screen size, smaller processing powers, smaller amount of memory, no mouse, and so on. Compared to fixed data connections, mobile networks can be slow and often have a higher latency. Compared to a user sitting in front of his computer, the user on the go has limited time and is easily distracted. On top of these constraints, the mobile world is highly fragmented: many different devices, each of them defining a unique set of supported features.

For these reasons, although most mobile devices may render Web documents, the user experience when browsing the Web on a mobile device is often poor when a Web document hasn't been designed with mobility in mind.

To help ensure that an appropriate user experience is possible on as many mobile devices as possible, the Mobile Web Best Practices Working Group, part of the W3C Mobile Web Initiative, defined a set of recommended guidelines to follow when creating Web documents: the Mobile Web Best Practices 1.0 specification.

Out of these best practices, the working group extracted a consistent set of best practices that may be automatically checked. This set of best practices defines the notion of mobile-friendliness mentioned above, and this is what the W3C MobileOK Checker actually tests. When a Web document passes all the tests, it is MobileOK. More precisely, the tests run by the W3C MobileOK Checker are formally defined in the MobileOK Basic Tests 1.0 specification.

Being MobileOK is neither a guarantee that the Web document may be rendered correctly by all mobile devices, nor insurance that the user experience was correctly addressed. Further quality improvements based on the full set of the Mobile Web Best Practices may be in order. The Mobile Web Best Practices Flip cards are a useful and handy companion for creators of Web content to keep the Mobile Web Best Practices in mind.

15 Important W3C Standards to be adopted

W3C develops technical specifications and guidelines through a process designed to maximize consensus about the content of a technical report, to ensure high technical and editorial quality.

- Gather information about the areas where best practice guidelines are needed: best practices will be drawn from the successes (and failures) of efforts at opening, sharing, and re-using knowledge about the use of standards and specifications by government applications that could be collected into a set of best practices with the intent of identifying productive technical paths toward better public services.

- Provide input on how to ease standards compliance: use previous successful experiences in terms of broad government use (such as the Web Accessibility Initiative work) to identify ways for standard bodies to better speak in terms of government needs; for example, additional effort to package, promote, and train on best practices and existing material and tools.

16 ISO 639 Language Codes

ISO 639 is the set of international standards that lists short codes for language names. It provides two set of three-letter alphabetic codes for the representation of names of languages. The ISO 639 code is divided into two parts i.e. Two-character code (639-1) Example: hi for Hindi, kn for Kannada and Three characters code (639-2, 639-3,639-5) Example: mar for Marathi, san for Sanskrit Used in ISO 639-1.

S.NO	639-1	639-2	639-3	Language Name	Native Name	Language Status
1.	as	Asm	asm	Assamese	অসমীয়া	Scheduled (Assam)
2.	-	Sit	brx	Bodo		Scheduled (Assam)
3.	bn	Ben	ben	Bengali	বাংলা	Scheduled (West Bengal, Bangladesh)
4.	-	-	dgo	Dogri	کنے /	Scheduled (J&K)
5.	gu	Guj	guj	Gujarati	ગુજરા તી	Scheduled (Gujarat)
6.	kn	Kan	kan	Kannada	ಕನ್ನಡ	Scheduled (Karnataka)
7.	ks	Kas	kas	Kashmiri	كشميري	Scheduled (POK)
8.	mr	Mar	mar	Marathi	मराठी	Scheduled Maharashtra)
9.	ne	Nep	nep	Nepali	नेपाली	Scheduled (Nepal, Sikkim and Bengal)
10.	or	Ori	ori	Oriya	ଓଡ଼ିଆ	Scheduled (Orissa)
11.	pa	Pan	pan	Panjabi	ਪੰਜਾਬੀ	Scheduled (Punjab)
12.	sd	Snd	snd	Sindhi	سنڌي، سندھی	Scheduled (Maharashtra, Gujrat)
13.	ta	Tam	tam	Tamil	தமிழ்	Scheduled (Tamil Nadu)
14.	te	Tel	tel	Telugu	తెలుగు	Scheduled (A.P)

सत्यमेव जयते

15.	ur	Urd	urd	Urdu	اردو	Scheduled (A.P, Delhi, J&K)
16.	sa	San	san	Sanskrit	संस्कृतम्	Scheduled
17.	-	Kok	Gom (kok)	Konkani	कोंकणी	Scheduled (Goa)
18.	-	Mai	mai	Maithili	मैथिली	Scheduled (Bihar)
19.	-	Mni	mni	Manipuri/ Meetei	মেইতেই লোন্	Scheduled (Manipur)
20.	-	Sat	sat	Santhali		Scheduled (Jharkhand)
21.	hi	Hin	hin	Hindi	हिन्दी	Scheduled (UP, UK)
22.	ml	Mal	mal	Malayalam	മലയാളം	Scheduled

Table: PRESENT TABLE OF LANGUAGE TAGS IN ISO 6391, 2 AND 3

For more details: http://www.w3.org/International/articles/language-tags/
http://www.sil.org/iso639-3/default.asp
For further details please refer: http://sil.org/iso639-3/codes.asp

Four letters Script code is available at the following url: http://unicode.org/iso15924/iso15924-codes.html

17 References

1. Microsoft: http://msdn.microsoft.com [Accessed: 12 April 2012]

2. Localisation : http://www.wordesign.com/Localization /index.htm#_Toc469910877 [Accessed: 12 April 2012]

3. Unicode: http://unicode.org [Accessed: 20 April 2012]

4. CDAC: http://cdac.in/downloads [Accessed: 20 April 2012]

5. Lingobit Best Practices: http://www.lingobit.com/solutions/bestpractices.html [Accessed: 30 April 2012]

6. Localisation: http://www.wordesign.com/Localization/index.htm#_Toc469910877 [Accessed: 30 April 2012]

7. I18Guy Localisation: http://www.i18nguy.com/unicode/c-unicode.html [Accessed: 25 April 2012]

8. ISO 639 code for languages: http://sil.org/iso639-3/codes.asp[Accessed: 30 June 2012]

9. 4 Letter Script Code: http://unicode.org/iso15924/iso15924-codes.html[Accessed: 30 June 2012]

10. CLDR: http://cldr.unicode.org/[Accessed: 30 June 2012]

11. Language Tag & Test Direction: http://www.w3.org/TR/html4/struct/dirlang.html[Accessed: 30 June 2012]

12. IBM – Globalisation Guidelines: http://www-01.ibm.com/software/globalization/[Accessed: 30 June 2012]

13. OASIS: https://www.oasis-open.org/[Accessed: 20 June 2012]

14. SUN Globalisation: http://developers.sun.com/dev/gadc/i18ntesting/checklists/detcheck/detcheck.html [Accessed: 15 April 2012]

15. ITS: http://www.w3.org/TR/2007/REC-its-20070403/[Accessed: 28 June 2012]

16. Indic Scripts and Unicode http://www.Unicode.org/standard/WhatIsUnicode.html [Accessed on 21 May 2012]

17. ISO/IEC Guide 2:2004, Standardization and related activities – General vocabulary.

18. TDIL: url: tdil.mit.gov.in [Accessed on 10-April-2012]

19. PASCII: url: http://parc.cdac.in/PASCii.htm [Accessed on: 29 May 2012]

20. Esselink, B. (2000). A Practical Guide to Localisation, Amsterdam: Benjamins.
21. Anastasiou & Schäler, (2010) ‘Localisation, Internationalisation, and Globalisation Against Digital Divide and Information Poverty’.
22. LISA Internationalisation definition, url: http://www.lisa.org/Internationalization.58.0.html, [accessed 17 Oct 2010]
23. Sikes, R. (2009) ‘localisation: the global pyramid capstone’, Multilingual, April, pp.3-5.
24. Internationalisation: http://www.w3.org/International/questions/qa-i18n [Accessed on: 25 May 2012]
25. N.S. Nikolov, R. Gupta and M. O'Haodha, Crowdsourcing and pedagogy: some reflections on the museum as collaborative learning space, 5th QQML conference [URL: http://www.isast.org/images/Book_of_ABSTRACTS_2013.pdf]
26. ISFOC: url: http://pune.cdac.in/html/gist/standard/isfoc.aspx [Accessed on 28 May 2012]
27. ICU: http://site.icu-project.org/ [Accessed on 29 May 2012]
28. PANGO: http://www.pango.org/ [Accessed on 29 May 2012]
29. ICU: http://site.icu-project.org/ [Accessed on 29 May 2012]
30. R. Gupta and S. Unde 'Towards evolution of localisation standards in Indian scenario', Translation, Technology And Globalization in Multilingual Context 3rd International Conference, June 23-26, 2012, New Delhi, [URL: India http://www.itaindia.org/ITAINDIA_conference_2012.pdf]
31. Multilingual Context 3rd International Conference, June 23-26, 2012, New Delhi, [URL: India http://www.itaindia.org/ITAINDIA_conference_2012.pdf]
32. D. Anastasiou and R. Gupta, 'Comparison of crowdsourcing translation with Machine Translation', Journal of Information Science, vol. 37, no.
33. 6, pp. 637-659, 2011.
34. Harfbuzz: http://www.freedesktop.org/wiki/Software/HarfBuzz [Accessed on 29 May 2012]
35. FUEL: https://fedorahosted.org/fuel/wiki/fuel-hindi [Accessed on 20 July 2012]
36. Font face: http://boldperspective.com/2011/how-to-use-css-font-face/ [Accessed on 25 July 2012]
37. Datatypes : http://wiki.ispirer.org/sqlways/informix/data-types/nvarchar_n [Accessed on 02-January-2013]
38. http://docs.python.org/2/library/gettext.html
39. GETTEXT: url: http://www.gnu.org/software/gettext/manual/gettext.html#Introduction [Accessed on 07-May-2013]

18 Acronyms & Abbreviations

L10N	Localization
I18N	Internationalization
CLDR	Common Locale Data Repository
FUEL	Frequently Used Entries for Localization
ISO	International Organization for Standardization
W3C	World Wide Web Consortium
OASIS	Organization for the Advancement of Structured Information Standards
XLIFF	XML Localization Interchange File Format
TMX	Translation Memory eXchange
TBX	Term Base eXchange
SRX	Segmentation Rules eXchange
ISCII	Indian Script Code for Information Interchange
PASCII	Perso-Arabic Script Code for Information Interchange
PFR	Portable Font Resource
EOT	Embedded Open Type
WOFF	Web Open Font Format

Annexure

Annexure 1

C/C++/VC++

These are some small tips which we can use during coding: -

```
A clear way to compare two strings to see if they're equal.
Code:
#include <string.h>
if (! strcmp (s1, s2)) // This is a BAD way to do it, it's counterintuitive
because! means not.
 #define Strequ (s1, s2) (strcmp ((s1), (s2)) == 0)
if(Strequ(s1, s2)) // This make a lot more sense as to what you're testing

Extensions:
#define StrRel (s1, op, s2) (strcmp((s1), (s2)) op 0)
Then call it with:
StrRel(s1, ==, s2);
StrRel(s1,!=, s2);
StrRel(s1, <=, s2);
Using CString in string copy function
    wcscpy(wchar_t*,(const wchar_t*)(LPCTSTR)Cstring)

For converting integer to string
char buffer[20];
int i = 3445;
_itoa( i, buffer, 10 );
Printf (“String of integer %d (radix 10): %s\n", i, buffer);

To know the size of any file
FILE *fpRead;
long lFileSize;
if ((fpRead=_tfopen(_T(“c:\\test.txt”),_T("rb"))) == NULL)
MessageBox (_T("Canot Open"),NULL, MB_OK);
else
{
fseek (fpRead, 0L,SEEK_SET);
fseek(fpRead,0,SEEK_END);
lFileSize =ftell(fpRead);
```

सत्यमेव जयते

Annexure 2

HTML

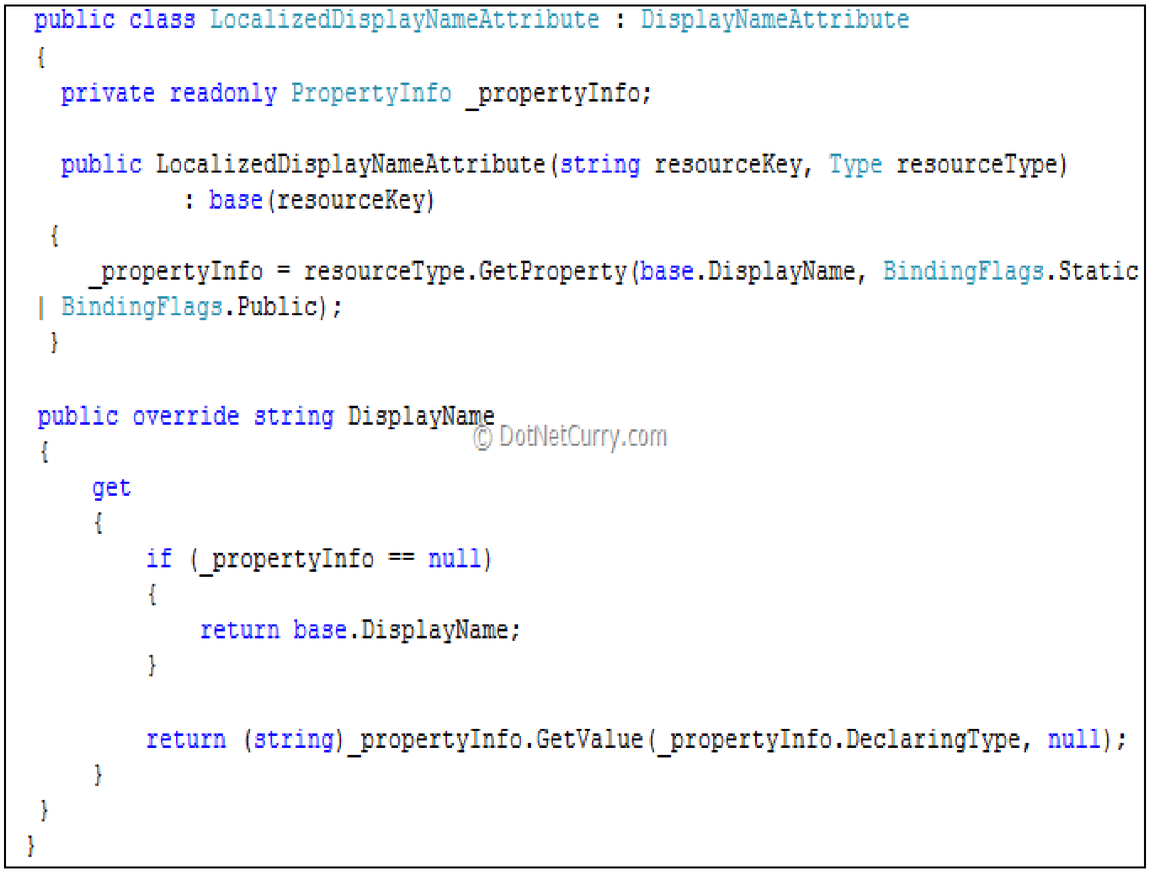
```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=<value>">

<html>

<head>
<meta name="description" content="Free Web tutorials by Rajat Gupta" />
<meta name="keywords" content="HTML,CSS,XML,JavaScript" />
<meta name="author " content="Hege Refsnes" />
<meta http-equiv="content-type" content="text/html;charset=UTF-8" />
</head>
<body>
</body>
</html>
```

Annexure 3

Asp.net



```
public class LocalizedDisplayNameAttribute : DisplayNameAttribute
{
  private readonly PropertyInfo _propertyInfo;

  public LocalizedDisplayNameAttribute(string resourceKey, Type resourceType)
          : base(resourceKey)
 {
    _propertyInfo = resourceType.GetProperty(base.DisplayName, BindingFlags.Static
| BindingFlags.Public);
 }

public override string DisplayName
{
    get
    {
        if (_propertyInfo == null)
        {
            return base.DisplayName;
        }

        return (string)_propertyInfo.GetValue(_propertyInfo.DeclaringType, null);
    }
}
}
```

Annexure 4

Java

```
JavaCodeDom jcd = new JavaCodeDom(new File(pathToJavaSource),
CompilerEnum.Java16)
List <ClassSrc> classes = jcd.getClasses();
ClassSrc cls = classes.get(0);
Map<MethodSignatureSrc,MethodSrc> methods = cls.getMethodsMap();
MethodSrc main = mothds.get(new MethodSignatureSrc(Modifiers.Public,
Modifiers.Static, ReturnTypes.Void, "main", new MethodParams(String[].class))
List<StatementSrc> statements = main.getStatements();
for(StatementSrc statement : statements){
   if(statement.getType()==StatementTypes.Assignment()){
        AssignmentStatementSrc assignment =
(AssignmentStatementSrc)statement;
        Identifier src = assignment.getAssigneeVariable();
        ExpressinoSrc = assignment.getAssignmentValue();
   }
}
List<AnnotationsSrc> annotations = cls.getAnnotations();
```

सत्यमेव जयते

Annexure 5

XML

```
< xml version="1.0" encoding="ISO-8859-1" >

<Configuration>
    <system.serviceModel>
        <bindings>
            <customBinding>
                <binding name="MyBinding">
                    <textMessageEncoding>
                        <readerQuotas maxDepth="2147483647"
maxStringContentLength="2147483647"
                            maxArrayLength="2147483647"
maxBytesPerRead="2147483647" maxNameTableCharCount="2147483647" />
                    </textMessageEncoding>
                    <httpTransport maxReceivedMessageSize="2147483647"
maxBufferSize="2147483647" />
                </binding>
            </customBinding>
        </bindings>
        <client>
            <endpoint binding="customBinding"
bindingConfiguration="MyBinding"
                contract="IMetadataExchange"
                name="http" />
        </client>
    </system.serviceModel>
</configuration>
```

सत्यमेव जयते

Annexure 6

Code Snippet : XML Database

```
<?php
// include Zorba API
require_once 'zorba_api.php';

// create Zorba instance in memory
$ms = InMemoryStore::getInstance();
$zorba = Zorba::getInstance($ms);

try {
  // create and compile query string
  $queryStr = <<<END
  let \$message := 'She sells sea shells by the sea shore'
  return
  <result>{\$message}</result>
END;
  $query = $zorba->compileQuery($queryStr);

  // execute query and display result
  $result = $query->execute();
  echo $result;

  // clean up
  $query->destroy();
  $zorba->shutdown();
  InMemoryStore::shutdown($ms);
} catch (Exception $e) {
  die('ERROR:' . $e->getMessage());
}
?>

Output :
                 <?xml version="1.0" encoding="UTF-8"?>
      <result>She sells sea shells by the sea shore</result>

For more details: http://www.ibm.com/developerworks/opensource/library/x-
zorba/index.html

   b)  Some Raw Zorba X-Query Examples :
(i) X-Query Insert
variable $stores := doc("stores.xml")/stores;

insert node element store {
  element store-number { 2 },
  element state { "Uttar Pradesh" }
} into $stores;
$stores
```

```
Output :
              <?xml version="1.0" encoding="UTF-8"?>
            <stores>
            <store>
/* existing node */
            <store-number>1</store-number>
            <state>Delhi</state>
          </store>
        <store>                                                /* output of
(A) new node is added with previously existing node
       <store-number>2</store-number>
       <state>Uttar Pradesh</state>
      </store>
       </stores>

(ii) X-Query Replace
     variable $store := doc("stores.xml")/stores/store[store-number = "2"];

    replace value of node $store/state/text()

    with "MH";
    $store

Output :
<?xml version="1.0" encoding="UTF-8"?>
<store>
    <store-number>2</store-number>
    <state>MH</state>
  </store>
(iii) X-Query Delete
      variable $stores := doc("stores.xml")/stores;

     delete node $stores/store[store-number = "2"];

     $stores

(iv) X-Query Full Text
doc("books.xml")/bib/book[author contains text "Singh"]

Output :
             <?xml version="1.0" encoding="UTF-8"?>
           <book>
           <author>R.M Singh </author>
           <author>T.M Verma</author>
            <title>SQL:1999</title>
            </book>
            <book>
             <author>J.K Singh</author>
             <title>Advanced SQL:1999</title>
             </book>
```