Towards a multilingual content infrastructure

Andrew Cunningham
Public Libraries Unit, Vicnet,
State Library of Victoria

Introduction

Since the late nineties, public libraries’ web sites have become more sophisticated, with many libraries looking at how their web sites can be used more effectively. Public libraries have come to terms with web accessibility and an increasing number of public libraries are beginning to use web content management systems. A similar understanding of web internationalization and the techniques for developing multilingual content has not developed, despite the number of library based multilingual web projects that have occurred within Australia since the late 1990s.

The State Library of Victoria has been involved in a number of multilingual web projects over the last eight years. Within the Victorian context, the evolution of the multilingual web presence of public libraries and the projects coordinated by the State Library of Victoria had its genesis in the multilingual web site developments that were being undertaken by the Port Phillip Library Service and the Maribyrnong Library Service during 1997 and 1998¹, combined with Vicnet’s community web publishing programs, which had a non-English language component. There has been a progression from static web pages using legacy character encodings, moving towards static Unicode web sites through to database driven Unicode web sites on towards XML based projects.

During 2005, Vicnet is developing MyLanguage, a multilingual portal in 30-60 languages. This is a joint project involving the State Libraries of Queensland, New South Wales, Victoria, South Australia and Western Australia and the Northern Territory Library and Information Service. MyLanguage will provide public access to resource location tools in multiple languages.

The diversity of Australia’s internet presence can be characterised by the following statements:

♦ Australia’s language diversity is not reflected in the non-English language content available on Australian government and library websites.

♦ We cater for a small fraction of our community languages.

♦ Within the web development and translation industries, practical knowledge of web internationalization techniques is limited and fractured.

The development of sites with multilingual content takes a relatively unique form in Australian government and library websites. There are currently two approaches used:

♦ **Embedded languages**: Some Australian government websites tend to wrap non-English content in English language templates, where the headers, footers, and site navigation are in English.

♦ **Mediated content**: The other prevalent model is to provide all navigation and description of documents in English. When you locate the resource you require you are presented with all the translations of that one document.

The prevalence of the mediated access model seems to be based on:

♦ The perception, in some circles, that immigrants do not use the internet. Although census data suggests, the older, more established and ageing communities have very low internet usage rates (similar statements could be said about the wider community within the same age bracket). Data would suggest that many other ethnic communities range from participation rates somewhat lower than the state and national averages through to participation rates well above the state and national averages.

♦ The additional cost of web development (in other languages), including translation costs.

♦ A general lack of awareness of web internationalization techniques.

♦ A lack of awareness of the current capabilities of operating systems and web browsers. In the early stages of the development of the web, many languages were not well supported, making it relatively difficult to create content in those languages. Currently, most languages are well supported and easy to work with.

It is important to note, though, that some languages remain areas of concern. One aspect of the digital divide that is often ignored is the language-based digital divide, i.e. languages that are unsupported or poorly supported by current operating systems and applications.

Through the State Library of Victoria’s experience with multilingual web site development, a number of strategies have developed to facilitate the development and operation of multilingual projects. Current W3C internationalization architecture (as expressed in the current versions of the W3C character model), combined with W3C web internationalization techniques and current State Library of Victoria development strategies foreshadow the need for a common multilingual content infrastructure to facilitate the development of multilingual projects within the Australian library sector. Such a need also underlies the Council of Australian State Libraries’ discussion paper on electronic multicultural library services\(^2\).

---

There are five building blocks that we use to develop a content infrastructure for multilingual portals:

1. W3C internationalization architecture\textsuperscript{3}. The W3C character model documents key internationalization principles that are necessary for web technologies.

2. W3C internationalization authoring techniques for XHTML and HTML document key internationalization techniques that allow documents to be successfully created and accessed.

3. An audience approach to web accessibility within the context of multilingual web sites. It is important to develop web sites that adhere to web accessibility guidelines.

4. Internal strategies and techniques for developing multilingual content.

5. A Multilingual Public Access Services (MPAS) model utilizing current generation multilingual public access internet workstations. Any non-English web development at the State Library of Victoria is developed as a component in broader MPAS. The inclusion and development of material in each language is in one sense contingent on the capabilities of public access workstations.

Web accessibility and internationalization techniques are core elements in Web based technologies. I prefer a minimalist approach to web design. By a minimalist approach, by this I mean I prefer projects to be straightforward, well planned and easy to implement. The key is the initial conceptualization, planning, and specification development of the site. It is important that web accessibility and internationalization be addressed at the very beginning. Retro-fitting internationalization and accessibility features onto a website can be an expensive and time consuming process, while creating an well internationalized site from the very beginning is relatively straightforward.

Multilingual web development should be no more difficult than web development of English only web sites.

It is important that institutions and organizations utilize “institutional knowledge”. It is never necessary to start from scratch with web internationalization. Use the skills and knowledge gained on earlier web development projects to refine and improve web accessibility and internationalization techniques.

With the projects VICNET works on, collaboration is also an important factor in developing a well designed multilingual website.

\textsuperscript{3}http://www.w3.org/TR/charmod/, http://www.w3.org/TR/charmod-norm/, http://www.w3.org/TR/charmod-resid/
Web internationalization techniques

The W3C Internationalization working group\(^4\) has released drafts of three authoring techniques documents for XHTML and HTML. The authoring techniques cover the three most basic and essential elements of web internationalization:

- Characters and encoding;
- Specifying the language of content, and
- Handling bidirectional text.

These documents are supplemented by a series of tutorials, FAQs and articles covering various topics related to web internationalization.

**Character encodings**

The W3C Authoring techniques for XHTML and XML stress the need to choose and identify an appropriate character encoding for the web documents or web site you intend to develop.

Choose a character encoding for the web document.

- Choose UTF-8 or another Unicode encoding.

- If you do not use Unicode, select an encoding that contains the greatest number of characters that are present in your web document. This is especially important if your site is using forms to submit data.

  For some languages, it is not possible to use Unicode, and a legacy character encoding must be chosen. Languages in this category would be those languages whose writing scripts are not currently supported by the Unicode standard, or languages that are encoded in Unicode, but are not widely supported by available font rendering technologies\(^5\).

- It is important that the character encoding you have selected is supported by user agents used to access your web site.

Specify the web page encoding.

It is possible to specify the encoding of the document via the web server, or within the web page, or in both places. A web browser will look for information on the character encoding value in the `charset` parameter in the `Content-Type` specified in the HTTP response header. This value will be used in preference to another method of identifying the character encoding.

If no `charset` value is sent in the HTTP header, the browser will next look for information on the character encoding within the web page.

---

\(^4\) [http://www.w3.org/International/geo/](http://www.w3.org/International/geo/)

\(^5\) Information on writing scripts awaiting inclusion into Unicode is available on the Unicode website at [http://www.unicode.org/pending/pending.html](http://www.unicode.org/pending/pending.html). Many of the pending scripts are writing scripts minority languages in South East Asian. Refer to summary *Proposals to encode Tai and other southeast Asian scripts in the UCS* at [http://www.evertype.com/standards/tai/tai-scripts.html](http://www.evertype.com/standards/tai/tai-scripts.html).
Use the preferred names from IANA’s registry for the value of the `charset` parameter.

- **Using the HTTP header.**

  Since web browsers will take the `charset` value identified in the HTTP header in preference to `charset` value identified by other mechanisms, it is important to use this mechanism correctly.

  This is only practical when the web page author is able to change the information that is sent in the HTTP header. Changes to the web server’s configuration, or upgrades to the web server may change the `charset` being identified, resulting in an incorrect `charset` value being transmitted in the HTTP header.

  Setting the `charset` value via the HTTP header is necessary when the server is converting (transcoding) a document from one character encoding to another before transmitting the document.

- **Declaring the encoding within the document**

  **For HTML documents and XHTML documents served as `text/html`:**

  Use a `meta` element, as early as possible within a web page, to identify the encoding of a web page. The declaration will take the form:

  ```html
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  ```

  **For XHTML served as `application/xhtml+xml`:**

  Specify the document’s character encoding in the XML declaration is mandatory, except when the document is served as UTF-8 or UTF-16 or the character encoding is specified in the HTTP header. Although, it is recommended that the encoding is still identified within the XML declaration. The XML declaration would take the following form:

  ```xml
  <?xml version="1.0" encoding="utf-8"?>
  ```

  If you are serving an XHTML document as `text/html`, it is recommended that you declare the encoding in both the XML declaration and in a `meta` element.

  The XHTML 1.0 specification requires that any XHTML document that does not identify the character encoding of the document, either via the HTTP header or in the XML declaration, must use either UTF-8 or UTF-16 as the document encoding.

---

6 If some one viewing your web page saves the document to a local storage medium, the information identifying the character encoding is "lost". For this reason the authoring techniques recommend that if you identify the `charset` using the HTTP header you should also identify the `charset` within the document.
Specifying language

W3C’s Internationalization working group and the Web Accessibility Imitative stress the importance of specifying the languages used within a HTML or XHTML document. The authoring techniques make a distinction between primary language and text processing language. The text processing language is the language in which the text of the document is written, and is processed to be displayed or read by a screen reader. The lang and xml:lang attribute are used to indicates the text processing language.

It is also possible to indicate the primary language or languages of a document using the HTTP header’s Content-Language or the equivalent meta element. In this context we are not discussion instructions for a web browser regarding the language of content; rather we are describing metadata identifying the language or intended audience of the document.

It is necessary to declare the default text processing language for the whole document. Declaring a text processing language in the html element will specify the default language for the whole document.

♦ For HTML documents you use the lang attribute.
   
   E.g. <html lang="mi"> to identify the default text processing language as Maori.

♦ For XHTML 1.0 served as text/html, both the lang and the xml:lang attributes should be used.

   E.g.  <html lang="so" xml:lang="so"> to identify the default text processing language as Somali.

♦ For XHTML documents severed as XML, i.e. application/xhtml+xml the xml:lang element should be used.

   E.g. <html xml:lang="km" xmlns="http://www.w3.org/1999/xhtml"> to identify the default text processing language as Khmer.

If the document has multiple main languages, it will be necessary to decide wether you declare one of the languages as a text processing language in the html element or leave the default text processing language undefined. If the navigation or metadata is in one specific language, then that language should be identified as the primary text processing language.

It is also necessary to declare any language changes within a document. Use the lang and/or xml:lang attributes around any changes within a document. It there is no appropriate element to add the language declaration to use the div element for a block change and use a span element for an inline change, e.g.

<p>The Chinese title is <span lang="zh-Hans" xml:lang="zh-Hans">世界人权宣言</span></p>
If there are multiple main languages within the document, the web developer should divide the document up into blocks at the highest possible level. The appropriate text processing language should be declared for each of these blocks.

It is important to use a valid language code. W3C Internationalization working group recommend:

- Use the guidelines in RFC3066 for language attribute values;
- Use the two letter ISO-639 codes for language attribute values when a language has both a two and three letter code.
- Consider using the codes zh-Hans and zh-Hant to refer to Simplified and Traditional Chinese.

**Bidirectional text**

In HTML and XHTML Unicode documents, it is possible to add the `dir` attribute to a HTML entity to indicate the directionality of text within that element. For a web page written in a right-to-left script, the overall document direction should be indicated in the `html` element:

```html
<html lang="ar" dir="rtl">
```

Do not add `dir="rtl"` to the `body` element.

The authoring techniques for handling bidirectional text recommend that web developers:

- Do not use CSS to control directionality. Markup should be used instead.
- Only add bidi markup to a document when it is needed. The Unicode bidirectional algorithm should be sufficient in most cases.
- Avoid HTML attributes with values of right to left (align and clear attributes). Use CSS in a linked style sheet instead. This will make the document easier to localize into a language using a right-to-left script.

To change the direction of a block level element, add the `dir` attribute to that element. The content of all nested block elements will inherit directionality.

For controlling the direction of inline elements:

- Use the Unicode LEFT-TO-RIGHT MARK (U+200E) and RIGHT-TO-LEFT MARK (U+200F) to control the directionality of direction neutral characters in relation to directional text surrounding them.
- Use the `dir` attribute on an inline element to resolve nested direction runs.
- Unicode provides a series of bidirectional control characters:
  - LEFT-TO-RIGHT EMBEDDING (U+202A),
RIGHT-TO-LEFT EMBEDDING (U+202B),
LEFT-TO-RIGHT OVERRIDE (U+202D),
RIGHT-TO-LEFT OVERRIDE (U+202E), and
POP DIRECTIONAL FORMATTING (U+202C).

Do not use these control characters for bidirectional control if markup is available.

It may be necessary to use control characters for attribute text or element text that does not allow internal markup.

For non-Unicode Hebrew HTML and XHTML documents use logical rather than visual order. The preferred ISO-8859 character encoding is ISO-8859-8-i.

Intersections

It would seem that the application of web accessibility guidelines to multilingual web sites should be straightforward. The default language should be correctly identified, and any changes in language should be indicated.

Australian government websites (both state and federal) provide a unique opportunity to observe the intersection of web accessibility and web internationalization techniques.

Accessibility issues centre on the:

♦ Use of images to display non-English text.
♦ Use of PDF files for display of translated documents.
♦ Ambiguous identification of language.

Images

The need to use an alt attribute for images is probably the most well known aspect of web accessibility. Small images containing text are often used for language specific navigation on web sites containing resources in multiple languages. The img element often looks like the following HTML snippet:

&lt;img src="italian.jpg" alt="Italian" /&gt;

Although on the surface this is fine, there is one issue. The text within the image would be written in Italian, and may say “italiano” or a welcome message in Italian, yet the alt attribute value is in English. The image is designed for one audience, while the alt attribute value is relevant to a different audience.

This situation is very common in Australian government web sites that contain multilingual documents.
The text used for the `alt` attribute should be for the same audience as the image itself, and should reflect the content and meaning of the image. The following versions of the `img` element would be more relevant to the intended audience:

`<img src="italian.jpg" lang="it" xml:lang="it" alt="Versione italiana" />`  

`<img src="italian.jpg" lang="it" xml:lang="it" alt="Italiano" />`

A second problem with images concerns web sites that use images to display large amounts of non-English language text. This was a common practice in the early period of the World Wide Web since browsers were unable to render text in most languages. The current level of language support in modern web browsers makes such a crude “fix” unnecessary. Unfortunately, such extended text images are still used by a number of commonwealth government web sites, even for languages that are supported by the default character encodings used for English. Typically, these sites do not provide alternative text versions or use the `longdesc` attribute.

**PDF**

Translation agencies and government departments (both federal and state) find the use of PDFs with embedded fonts more convenient than working with HTML or other document formats for translated material.

Some screen readers can handle the PDF document format; while other tools exist that allow PDF files to be converted to other document types including HTML. Adobe themselves, the developers of the PDF specification provide an online conversion tool that allows for the conversion of PDF files to HTML documents.

For PDF documents to be accessible, the documents should be tagged PDF files. The languages used in the document should be identified, with all language changes tagged.

Additionally only well-formed TrueType font programs should be embedded in PDF files. It is important that the fonts contain information to identify the character encoding used, or identify the Unicode character code for each glyph. “This character identification can occur if either the font uses a standard named encoding or the characters in the font are identified by standard character names or CIDs in a well-known collection.⁷” If it is not possible to determine a Unicode codepoint for each character when the text is extracted from the PDF, then it is not possible to correctly identify the character.

**Languages**

It is important that language tagging be unambiguous. This is not always possible with language tags. Most language tags refer to both spoken and written forms of the language, although some language tags refer primarily to a written form of the language, while other tags refer to the spoken language. It is necessary to be as precise as practical.

---

Cues to the future

*MyLanguage* is a national multilingual resource location tool. The development of the *MyLanguage* portal builds on the knowledge and techniques acquired through earlier multilingual web development projects. The specification outlining the development of the portal draws on the W3C internationalization authoring techniques and on strategies and approaches developed in house to manage multilingual portals.

The central component of the *MyLanguage* portal is a custom built, modular and flexible content and document management system built using open source solutions. UTF-8 was the only possible character encoding to choose. The range of languages and writing scripts stored within the database, make any choice of a non-Unicode character encoding both impractical and undesirable. It also makes it feasible to interface with and utilise external web services.

It is essential to this project that at each step the character encoding is clearly identified and that the same character encoding is used in Web pages, scripts and databases.

One of the intentions is to develop a tool that will not only work with the large community languages in Australia, but that also accommodates minority and new community languages. Some of the new emerging community languages require complex script rendering and necessitates using very recent versions of Microsoft Windows. Such languages will require Unicode normalization and custom mechanisms to handle language appropriate collation sequences.

**Normalization**

One key issue with resource location tools, whether they are search engines, web directories or other tools, is the need for Unicode normalization. The Vietnamese phrase “Tiếng Việt” can be represented in Unicode as (1) a sequence of single discrete characters for each letter of the Vietnamese alphabet, (2) the vowels with diacritics could alternatively be represented as a base character followed by combining diacritics, or (3) the Microsoft approach, which uses precomposed characters for the vowels “a, ā, ě, ē, i, o, ō, ơ, u, ŭ and y” and combining diacritics for the tones.

<table>
<thead>
<tr>
<th></th>
<th>Tiếng Việt</th>
<th>NFC based input (composed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Tếông Viẽt</td>
<td>NFD based input (decomposed)</td>
</tr>
<tr>
<td>3</td>
<td>Tiếng Viẽt</td>
<td>Windows 2000/XP keyboard layout</td>
</tr>
</tbody>
</table>

---

8 The foundation for the *My Language* project includes knowledge and experience gained from stage one and two of the Open Road project (http://www.openroad.vic.gov.au/). Research for stage three of Open Road provided the starting point for the CASL discussion paper on electronic multicultural library services which formed the basis of the *My Language* proposal.
Unicode considers these three sets of character sequences to be canonically equivalent. Although they are distinct unicode codepoint sequences.

The Unicode standard includes a process, referred to as normalization, which allows Unicode text to be converted to one of four preferred forms: canonical decomposition, compatibility decomposition, canonical composition and compatibility composition. For most languages we deal with, data will invariably be fully precomposed (NFC). Vietnamese is an example of a problematic language. Most Vietnamese third party input software by default uses precomposed characters. Some software also offers the option of using decomposed character sequences. Microsoft added Vietnamese support to the International English edition of Windows 2000. The keyboard layout it uses in Windows 2000 and Windows XP does not produce normalized character sequences.

When using services such as Google, which do not normalize search strings, it is important to conduct a Unicode search using both a third party Vietnamese input software and Microsoft’s Vietnamese keyboard layout. Each search will produce a different set of results. This type of dilemma is also being observed in some African languages.

In order to minimize the impact of divergent keyboard layouts in specific languages on the MyLanguage portal, all data entered by the site’s editors and administrators should be normalized. Normalization Form C was chosen. Data sets that are imported into the portal’s database also be normalized when imported. The portal’s search mechanism should also include an option to normalize search strings.

**Collation**

The effective use of collation is one area of multilingual web development that can be problematic. In the context of the My language project, it will be necessary to develop an effective mechanism for handling collation issues.

The Unicode Consortium defines colation as “the general term for the process and function of determining the sorting order of strings of characters”. The ability to sort data is crucial for a resource location tool. Collation varies by language, culture, and geographical region. The Common Locale Data Repository (CLDR) Project provides access to a range of locale data including collation sequences for a wide range of languages. Existing tools use this data to enable language specific collation.

The key issue for the My language project is that not all the languages within the project have documented locales, and sorting routines would be unable to handle these languages. For some of the new and emerging community languages, it will be necessary to develop custom collation routines to sort data for these languages.

---


**Bidi support**

Handling bidirectional support in websites is relatively straight forward, with the W3C Internationalization working group providing useful guidelines, tutorials and FAQs on a wide range of issues relating to bidi support in XHTML and HTML.

Within the context of our projects, bidi support is straight forward for generated web pages. It is simple to specify the correct directionality for a web page based on the language and writing script used.

The component of the web site where care needs to be taken is in editorial pages, where content is created or edited for the website. The editorial components use XHTML forms, these forms need to support the directionality required for the language of the resources being edited.

One possible strategy is to assign the directionality of the page based on which editor is using the site. This approach assumes that the editor is only ever creating and editing content in one language, or languages sharing the same writing system. An alternative strategy would be for the editor to choose a language they wish to edit and generating an input page based on the language chosen.

**Stylesheets**

Stylesheets can be used to control language specific presentation of a website. Different languages may require different fonts, font sizes, line heights, quotation marks, justification, list styles, etc. CSS provides mechanisms for applying style based on language.

There are four ways to commonly apply styles based on the current text processing language:

- :lang() pseudo class selector.
- [lang('='|"...")] selector matching the beginning of a language attribute value.
- [lang='...'] selector matching a language attribute value.
- A generic class or id selector.

The first three methods work with more recent browsers like Mozilla, Mozilla Firefox and Opera. Unfortunately, they do not work with the increasingly dated Internet Explorer. As a work around it is possible to use the class or id selectors instead which are support by Internet Explorer.

In the *My language* portal, the approach taken is to apply a base style to the XHTML body element, and use language selectors for language specific style. Internet Explorer would use the style applied to the body tag, and would be unable to render style applied to a CSS selector based on the indicated language of the text. Obviously, for a multilingual portal supporting diverse community languages, applying

---

11 FAQ: Styling using the lang attribute. [http://www.w3.org/International/questions/qa-css-lang](http://www.w3.org/International/questions/qa-css-lang)
a specific font to the body element would only address some languages, and not all. Internet Explorer has its own algorithm to selecting appropriate fonts for different writing scripts, allowing it to correctly render a language even though the language is not supported by the specified font.

```html
body {font-family: "Times New Roman", serif;}
*{[lang|="ar"] {font-family: "Traditional Arabic", serif;
    font-size: 1.2em;}}
:lang(ar) {font-family: "Traditional Arabic", serif; font-size: 1.2em;}
*{[lang|="ig"] {font-family: "Doulos SIL", serif;}
:lang(ig) {font-family: "Doulos SIL", serif;}
*{[lang|="zh-Hans"] {font-family: SimSum-18030, SimSum, MS Hei, serif;}
:lang(zh-Hans) {font-family: SimSum-18030, SimSum, MS Hei, serif;}
```

In the above example, the body element is displayed with *Times New Roman*, while text identified as Arabic would display using the *Traditional Arabic*, Igbo (Ibo) would display using *Doulos SIL*, and Simplified Chinese text would use *SimSum-18030* if the font was available.

**Language control table**

Rather than building in language specific features into the scripts that generate the portal’s web pages, the scripts use variables that are controlled from a language table within the portal’s database.

The table contains the following information for each language that is enabled for the portal:

- A lexicon, containing translations of a controlled vocabulary of terms used in various parts of the portal.
- RFC 3066 compatible language codes that scripts will use to populate `lang` and `xml:lang` attribute.
- Language specific collation sequence for minority languages.
- Directionality. Scripts will use this value to populate the `dir` attribute.
- A flag to indicate if normalization should be used on data. Not all languages will require normalization.
- Markup and Javascript fragments to generate virtual keyboards, if required.
- Language specific CSS.
Towards an ideal Multilingual Public Access Workstation

There are two main components involved in developing multilingual public access services within public libraries. The first is access to appropriate resource location tools. The internationalization techniques are central in the development language specific resource location tools. The Open Road directory[^12] and the My language portal are examples of resource discovery tools. In addition, many public libraries have developed their own small directories to non-English resources that reflect the information needs of their local community.

The second component is the multilingual public access workstation (MPAW) itself. This is the aspect of service provision that is often neglected. The basic model that most public libraries follow is to base the requirements for their public access workstations on the needs of their English language users and then throw in a few extras. The installation of a few extra legacy fonts, enablement of a small set of Windows keyboard layouts, and the addition of one or two language specific applications is, at best, a very basic MPAW.

What would the ideal workstation be? The starting point is the operating system, or more precisely the language support offered by the operating system. If you only needed to support English, then realistically, you could get away with any available operating system. Each version of Windows supports a different range of languages. The general rule of thumb is that newer operating systems support more languages.

The starting point is your own community. What languages are used in your local community? Which languages do you need to support? This will differ from library service to library service. Some will support a small number of the largest community languages. Some will try to accommodate as many community languages as possible. The range of languages you need to support will determine the minimum operating system requirements.

If I wish to support Italian or Greek, I could choose any operating system from Windows 95 on. On the other hand, if I need to support Hindi, Tamil and Arabic, you would need Windows 2000 as a minimum. Gujarati or Panjabi (Punjabi) support would require Windows XP. Bengali would require Windows XP Service Pack 2.

It is necessary to differentiate between languages that Microsoft provides full support for, and those languages that are supported by the font rendering system but have no other support (i.e. no fonts or keyboard layouts). Amharic, Dinka, Lao, Khmer and others can be supported, if a version of Windows that supports their writing scripts is available.

Currently, the ideal operating system is Windows XP Service Pack 2. It provides support for the widest range of languages.

The next element to consider is the web browser. Despite Internet Explorer 6 being a relatively old web browser, Internet Explorer 6 and its predecessors remain the most common web browser in use. Patrons expect to find Internet Explorer on a public

access workstation. Additionally, it is advisable to also install the most recent version of Mozilla Firefox.

**Firefox**

Localized language packs \(^{13}\) are available for Mozilla Firefox. These language packs contain translated user interfaces (UIs) for the web browser. Mozilla Firefox also supports multiple user profiles. It is possible to combine user profiles and UI language packs to enhance non-English language access to the internet.

It is possible to customize each profile, optimizing the profile for each target language. Assume I am creating a profile for Russian (Русский). Possible customizations include:

- Setting the order for preferred languages. Some web sites use language negotiation. When a web browser requests a web page from a server, it automatically identifies the preferred language of the user. If the web server supports language negotiation, it will send the appropriate version of the web page if it has a page that matches the language identified by the web browser. For the Russian profile, I’d set up Russian (ru) as the preferred language.

- If appropriate, change the default character encoding. Some older web pages fail to identify the character encoding used. For Russian, one possible character encoding would be Windows-1251.

- If the language in question uses an unsupported character encoding, select fonts supporting this encoding for the *User defined* slot. This should be unnecessary for most languages. All main legacy encodings are supported internally by Internet Explorer and Mozilla Firefox.

- Install the applicable language pack. There is a *ru-RU* language pack available for Mozilla Firefox.

- Bookmark key online media sites and search engines for the required language, adding them to the *Bookmarks Toolbar*. It would also be possible to subscribe to appropriate syndicated news services. Mozilla Firefox 1.0 has a built in RSS viewer. For instance, in the Russian profile, it would be possible to bookmark the BBC World Services Russian RSS file, adding it to Firefox’s *Bookmarks Toolbar*.

- Add useful Firefox extensions. Examples of possible extensions within relevant languages:
  
  - **同文堂 (Tong Wen Tang)**: provides a toolbar button that allows you to convert a web page between Simplified and Traditional Chinese Characters.
  
  - **Right Encoding**: provides a context menu for selecting (or over-riding) document’s character encoding.

---

\(^{13}\) Mozilla Firefox 1.0 language packs:
ftp://ftp.mozilla.org/pub/mozilla.org/firefox/releases/1.0/win32/xpi/
BiDi UI: Provides a menu and context menu interface for controlling direction when viewing some right to left languages. In the long term this extension should become unnecessary as browser support for languages requiring bidirectional support is extended.

Link Toolbar: provides a navigation toolbar that utilises the information in the link elements of a web page.

Once the browser is configured correctly, it is possible to make a shortcut on the desktop and name it. The format for the command-line options (or shortcut target) would be:

"C:\Program Files\Mozilla Firefox\firefox.exe" -p profile -UILocale locale

Examples:

<table>
<thead>
<tr>
<th>Shortcut title</th>
<th>Shortcut target</th>
</tr>
</thead>
<tbody>
<tr>
<td>中文简体字</td>
<td>&quot;C:\Program Files\Mozilla Firefox\firefox.exe&quot; -p ChineseS -UILocale zh-CN</td>
</tr>
<tr>
<td>Ελληνικά</td>
<td>&quot;C:\Program Files\Mozilla Firefox\firefox.exe&quot; -p Greek -UILocale el-GR</td>
</tr>
<tr>
<td>Italiano</td>
<td>&quot;C:\Program Files\Mozilla Firefox\firefox.exe&quot; -p Italian -UILocale it-IT</td>
</tr>
<tr>
<td>Русский</td>
<td>&quot;C:\Program Files\Mozilla Firefox\firefox.exe&quot; -p Russian -UILocale ru-RU</td>
</tr>
<tr>
<td>Türkçe</td>
<td>&quot;C:\Program Files\Mozilla Firefox\firefox.exe&quot; -p Turkish -UILocale tr-TR</td>
</tr>
</tbody>
</table>

Input

The next step would be to identify which community languages have input support available within the version of MS Windows the public library/service is using. Many of the larger community languages are supported by Windows XP Service Pack 2. It is straightforward to enable these languages.

An additional measure that should be taken is setting up a desktop shortcut for the Windows 2000/XP virtual keyboard. The shortcut's target would be:

%SystemRoot%\system32\osk.exe

Microsoft has included a Vietnamese Unicode keyboard layout in Windows 2000 and Windows XP. As indicated earlier Microsoft’s Vietnamese keyboard layout does not produce normalized output. It would be advisable to install third party input software to provide an alternative input method for those instances where it is required. There are a number of alternatives.

It will then be necessary to then add software to handle the languages not supported by Microsoft. Within our projects, we have been using Tavultesoft Keyman. The keyman utility allows you to use keyboard layouts that have been written for Keyman. A wide range of Keyman keyboard layouts have been developed and are available

14 The open source program “Unikey” is widely used. In addition to Unicode support, it also allows you to type in any of the major legacy 8-bit encodings used for Vietnamese. http://unikey.sourceforge.net/
15 http://www.tavultesoft.com/keyman/
for downloading and installation, including some of the new and emerging community languages from Sudan and the Horn of Africa.