Rendering multilingual documents with CSS2 and XSL

Chris Lilley, W3C
www.w3.org/people/chris

Abstract

The rendering of multilingual XML and HTML Web documents is discussed with reference to the internationalization features in the Cascading Style Sheets, level 2 (CSS2) Recommendation and the new XSL Working Draft.

1 What is a document?

1.1 The Web is not a typewriter

There is a fundamental difference between publishing on paper and publishing on the Web. On paper, it does not matter which keystrokes were used to produce a particular glyph. Indeed it is common for designers to re-arrange fonts to get a keyboard layout that suits them. The reader never sees anything except the final rendered (printed) version of the document, and there is no way to tell, looking at the printed result, what keystrokes produced it. On the Web, in contrast, it matters a great deal. Thus, the first task is to consider the question ‘what is a document’.

1.2 Unicode, HTML and XML

The Web uses HTML as its primary media type. HTML is an application of SGML, and the SGML declaration for each document type specifies a particular Document Character Set in which all computations and manipulations are performed. For HTML version 2.0 and above, this single Document Character Set has been ISO 10646 (although version 2 of HTML was restricted to the first 256 characters, in other words Latin-1). With HTML 4.0 the document character set was stated to not only be ISO 10646, but Unicode - which has the same code points but also implies additional functionality such as the Unicode bidirectional algorithm.

XML is a language for writing document types; a Web-enabled successor to SGML. Whilst individual XML applications will invent their own element names, certain things are common to all XML applications. One of these is the use of Unicode as a document character set (XML does not need or use SGML declarations).

Thus, both HTML and XML documents share a common repertoire of characters - Unicode - which may be used for document processing. In this paper I will use the term ‘document’ to refer to either an HTML or XML document.

1.3 Structured documents

Another thing that HTML and XML share in common is that they are markup languages - the textual content is structured by being grouped into elements, which may contain child elements. This structuring may be exploited by programs which are aware of the semantics associated with the element names. For example:

```
<presentation>
<title>presentation at 13<sup>th</sup> Unicode Conference</title>
<slide>
<heading>My first slide</heading>
<item>a key point</item>
<item>another one</item>
<item> a third, with a picture <image/></item>
</slide>
<slide>
<!-- another slide here -->
</slide>
</presentation>
```

The element names are suggestive to an (English-speaking) reader, but have no semantics for the computer. However, a human who knows the semantics can write a program to number the items in each slide, or to generate a table of contents containing the heading for each slide.

The nested hierarchy of elements can be considered as a tree. In this example the `presentation` element is the root node of the tree, and contains a `title` element and several `slide` elements as children. Each `slide` element contains a `heading` element and several `item` elements. This tree structure is important for document processing, and in particular for using style sheets.

1.4 Document transmission

The document can be transmitted from server to client in the document character set, but it need not be. Particularly if the document contains text in only one or two different languages, it is often more efficient to use a different c*har-*

acter encoding for transmission. The particular encoding in use is declared in the HTTP charset parameter of the MIME type. Examples include ISO-8859-1 for French documents, ISO-8859-8-I for Hebrew documents, and EUC-JP for Japanese documents.

Regardless of the encoding used in transmission, all numeric character references (NCRs) refer to the Document Character set not the character encoding. For example, Κ represents the character at code point 922 decimal, U+039A, GREEK CAPITAL LETTER KAPPA. This NCR can be inserted in any document, regardless of the character encoding that happens to be in use. For example, the document might be in Japanese and sent using shift-jis; it can still contain any Unicode character.

1.5 Bytes, characters, glyphs

Conceptually, the incoming document is converted from a stream of bytes into a stream of Unicode characters, using the character encoding information. It is then converted into a sequence of glyphs, using the font encoding vector information. Both of these mappings can be 1:1 but they *might not* be. Assuming that these mappings are always 1:1 has been a prime cause of non-interoperability on the Web [Harm].

Both HTML and XML rely on this distinction, which is well described in the ISO Technical Report on the Character Glyph Model, TR 15285. W3C style sheet specifications also rely on this distinction and in some cases provide facilities for enforcing it.

2 Style Sheets

2.1 Introducing style sheets

Style sheets are a way of gathering together all the presentational information in a document, leaving the actual document to concentrate on describing the structure.

For example, if a document has ten subheadings, the stylesheet could simply indicate, with one or more style sheet *properties*, that they are all to be in 16/20 point Helvetica Oblique rather than, as with the HTML `FONT` element, having this information embedded in the HTML and thus repeated ten times. To do this, the style sheet must have some way to indicate the elements to be styled (a selection mechanism) and a means to indicate the desired styling (a set of formatting properties, and values for each property).

Style sheets can be external (in another file) or internal (inside the document). If external, they are associated with the document by a link. A document may

be associated with more than one style sheet. This allows alternative presentations of the same information.

2.2 Cascading Style Sheets (CSS)

The term CSS refers to a style sheet language first developed at CERN, around 1994, and later at W3C [CSS]. Originally aimed particularly at online use with HTML documents, it can also be used with XML documents and can format for other output media such as speech and print. The *cascade* refers to the way multiple style sheets - those linked to the document by the document author, the reader's personal stylesheets, and the browser internal default stylesheet - are combined in a precise order to contribute towards the end result. Level 2 of the CSS specification, finalized in May 1998, is a W3C Recommendation. Work is in progress on the next level of CSS.

In CSS, the formatting object tree is nearly identical to the document tree (the differences being generated text, and optional special treatment of the first line or first letter of an element). Inheritance of formatting property values is on the document tree.

CSS uses selectors to determine which elements to apply particular style rules to. The particular properties to change are then given inside curly brackets. For example, to make all `item` elements which are children of `slide` elements green and italic:

```
slide > item { color: green; font-style: italic }
```

In this example two properties are modified, one for the text color and one which controls whether the desired font is upright, oblique or italic.

2.3 Extensible Style Language (XSL)

The eXtensible Stylesheet Language, XSL, is the other style sheet format from W3C; it aims to include the functionality of both DSSSL, the ISO style sheet language, and CSS2 [XSL]. A W3C Working Group was formed in January 1998 to develop XSL, which is specifically targetted at complex, data-rich XML documens which require extensive reordering and computation for display. A requirements document was issued by this group in May 1998 and the first Working Draft of XSL was recently released in August 1998.

In XSL, the formatting object tree can be radically different from the document tree and there is a special tree construction step to generate it. The content of elements from the document tree can appear in multiple places in the formatting object tree, for example the text of headings can reappear in an index, table of contents, and page headers or footers. In consequence, inheritance of formatting property values is on the formatting object tree.

XSL uses patterns to select which elements to apply styles to, and then uses a template to describe the part of the formatting object tree to construct. XSL is itself written in XML, and the style properties to change are given as XML attributes.

For example, to make all `item` elements which are children of `slide` elements green and italic:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/TR/WD-format"
  result-ns="fo">

  <xsl:template match="slide/item">
    <fo:block font-style="italic" color="green">
      <xsl:process-children/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

There are two different ways to use XSL on a data-rich XML document. One way is to use the formatting objects to transform and style the content directly. The other way is to first transform into a simpler XML document, then style this using CSS. It is possible to transform on the server, and then style on the client (the browser).

3 Language-dependent styling

HTML 4.0 includes the `lang` attribute, introduced in RFC 1766 and more tightly defined in RFC 2070, to specify the human language used by that element and its children.[1] In XML, the `xml:lang` attribute exactly mirrors the semantics and properties of the HTML 4.0 `lang` attribute.

Both CSS2 and XSL make this information available to the style sheet author to perform language specific formatting.

1. This attribute contains a primary code and a (possibly empty) series of subcodes, separated by hyphens. For example, `de`, `ja-jp`, `en-nz`, `no-nynorsk`, `fr-ca`, `x-cherokee`. If the primary code is two letters, it is an ISO 639 language abbreviation [639]; if a subcode has two letters, it is an ISO 3166 country code [3166].

3.1 Language-dependent font selection

A Web page containing passages both in Chinese and in Japanese may well use the same CJK Unified Ideographs in both sections; the different language codes `zh-tw` and `ja-jp` may be used by the style sheet implementation to select the appropriate Traditional Chinese and Japanese fonts, respectively.

```
*:lang(ja-jp) { font: 900 14pt/16pt "Heisei Mincho W9", serif }
*:lang(zh-tw) { font: 800 14pt/16.5pt "Li Sung", serif }
```

In this example, different weights, sizes and line heights have been chosen to allow these two fonts to exist harmoniously together.

3.2 Language-dependent quoting

Quotation marks are often inserted literally into the document, but may also be auto-generated. The HTML 4.0 `Q` element, for example, is specifically required to use generated quotes. CSS2 introduced the quotes property to ease the creation of this generated text. It takes a list of pairs of character strings, which are to be used for each level of embedding of quotations. For example, to format a book written in both French and German:

```
:lang(fr) > Q { quotes: '« ' ' »' }
:lang(de) > Q { quotes: '»' '«' '\2039' '\203A' }
```

This pair of rules sets the `quotes` property on `Q` elements according to the language of its parent - the > operator means “child of” in CSS. This is done because the choice of quote marks is typically based on the language of the element around the quote, not the quote itself; for example a snippet of Italian “bella casa” uses English quotation marks because of the surrounding English context.

In the example above, for quotes in a French context, the left double guillemet is used to open the quote and there is a space after it. For quotes in a German context, the convention is to use the guillemet the other way round and not to have extra space between the quotation marks and the quoted matter.

Also, for nested quotes in a German context, a different pair of quotation marks are used, specified by their hexadecimal Unicode character positions. In CSS, any Unicode character may be used for generated text, even if it does not fall within the character repertoire of the encoding used in transmission. The same is true of XSL, which uses the numeric character references in XML to achieve the same result.

4 Coordinate systems

Positions on a page - and this applies equally to Web pages as to printed pages - can only be described by reference to a coordinate system. There are three main types:

1.Absolute coordinate systems

2.Writing direction relative coordinate systems

3.Page binding relative coordinate systems

4.1 Coordinate systems in CSS

CSS primarily uses an absolute coordinate system - the left margin is always the left margin, regardless of whether writing starts at that margin or ends there. Coordinate systems are used in CSS to specify margins, the width, color and style of borders, the size of padding between content and the edge of the background, clipping and scrolling behavior, and when using positioned elements. For example:

```
chapter > section { border-left: thin solid green }
```

There are four properties: `left right bottom` and `top`, which are used to place positioned elements to achieve non-linear, layered designs. The style sheet author will pick the appropriate two of these depending on the writing direction for which the style sheet is being designed. In consequence, the same style sheet will not work for both left-to-right and right-to-left languages.

4.2 Coordinate systems in XSL

XSL generally uses writing direction relative coordinate systems, although the page margins are specified in an absolute system (left, top, etc). Page binding relative coordinate systems are not currently used in XSL, although it is recognised that they are useful, particularly for specifying the inner and outer page margins in a work to be printed for binding.

The writing direction for a given formatting object, or for the document as a whole, is selected from an enumerated set. For example for Latin languages, it would be set to `lr-tb` which specifies:

- an inline-progression-direction of left-to-right.
- a block-progression-direction of top-to-bottom.
- a line-progression-direction of top-to-bottom.
- a shift-direction of bottom-to-top.

This is illustrated in Figure 1.

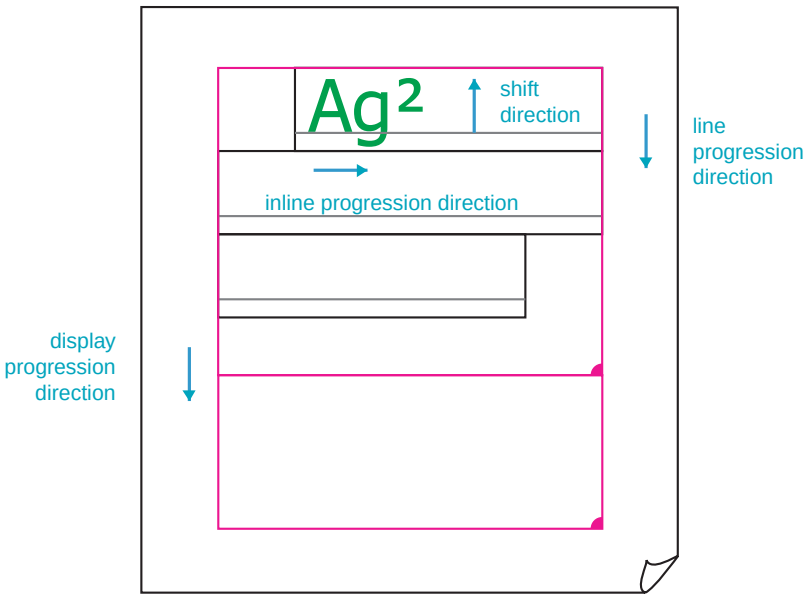

Figure 1 : XSL relative coordinate system, Latin

The corresponding specifications given a writing direction of rl-tb are shown in Figure 2.

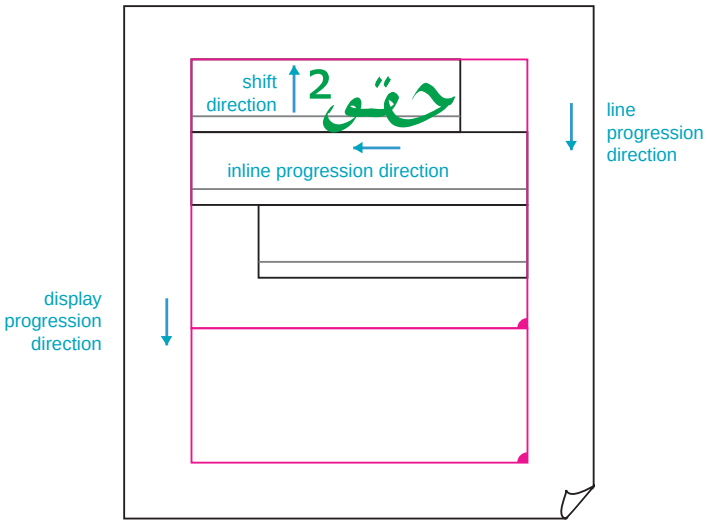

Figure 2 : XSL relative coordinate system, Arabic

XSL can specify a large number of writing directions, including the one used for vertical Japanese (see Figure 3) as well as other writing directions. The advantage of using a single enumerated value to indirectly specify a set of directions is that it is extensible and also that it copes with those writing directions, mainly historical, where for example the inline progression direction is not constant but takes the opposite value on even and odd lines. Ancient Greek, written in bous-

trophedon (as the ox ploughs) is one example of such a writing direction, which would be specified as `lr-alternating-rl-tb`.

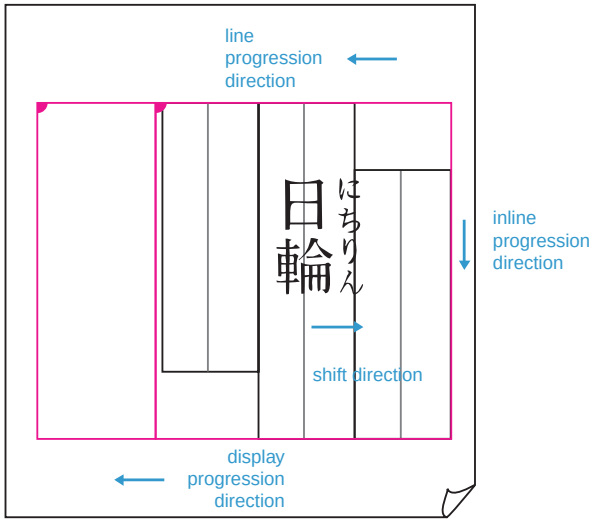

Figure 3 : XSL relative coordinate system, vertical Japanese

4.3 An example of coordinate system use - Ruby

When written horizontally, in Japanese the shift direction (which among other things controls the positioning of Ruby) is bottom to top. When written vertically, the shift direction is left to right as in Figure 4. This is not sufficient to fully specify Ruby, of course, just to indicate that when the stylesheet asks for the Ruby to be positioned closer to or further from the main text, what direction that means.

Figure 4 : Vertical Japanese text with Ruby

5 Web Fonts

5.1 Font features of CSS1 and XSL

Both the CSS and XSL specifications allow the setting of various font properties on XML elements. The properties include the family name (e.g. Times, Arial), weight (e.g. normal, bold), style (e.g. italic) and size (e.g.12pt). These properties can be inherited and then modified by child elements (CSS) or child formatting objects (XSL). For example, various font properties may be set on a paragraph, and a bold element within that paragraph can be made to have the same font - but in a heavier weight - by simply altering the font-weight property.

In CSS1 and the first working draft of XSL, fonts are assumed to be present on the client system and are identified solely by name. Several choices of fonts may be listed and are tried in order until a font is found that can display the required characters. If a font is available to the client that is a close stylistic match to the requested font but has a different name, it is not possible for a CSS1 implementation to select it. Generic font families such as ‘serif’ and ‘script’ are available as fallbacks if none of the listed fonts are available.

Because CSS and XSL honor the character-glyph model, it is *not* possible to apply, say, the Symbol font onto Latin text to get a semblance of Greek. The font will fail to match and the next font in the sequence will be used. This property allows document authors to specify several fonts for a single element, and the appropriate one will be used automatically. For example, in this fragment:

```
<xsl:template match="slide">
  <fo:block font-family=”Palatino, Bukinst,‘Heisei Mincho W3’,serif”>
    <xsl:process-children/>
  </fo:block>
</xsl:template>
```

Here, Palatino covers Basic Latin and Latin-1 Supplement, Mincho covers some of the CJK Unified Ideographs using Japanese glyphs and Bukinst covers Cyrillic. Bukinst is placed before the Mincho font because some Japanese fonts also contain glyphs for cyrillic and we want the ones from Bukinst to be used in preference. In a mixed French, Japanese and Russian document, the correct font will, if available, be selected for each character without the necessity of special markup around each run of characters from a particular script. The last font in the list, serif, is a generic fallback font which is defined to exist by the CSS and XSL specifications.

5.2 CSS2 intelligent name matching

The first additional feature provided by the Web Fonts extension in CSS2 is intelligent matching. This was not described in the first Working Draft of XSL, but it is planned to add it in a subsequent draft.

Intelligent matching involves using more information than just the name of the requested font to select an existing, accessible font that is the closest match in appearance to the requested font.The metrics might not match, resulting in different line breaks. The matching information includes information about the kind of font (textual or pictorial), the style of serifs (or analogous stroke terminations), weight, cap height, x height (where these can be defined), typographic ascent and descent, slant of vertical strokes, and so on. This is essentially an open-ended and extensible set of font characteristics, initially derived from Latin typography.

Problems come from inconsistent definitions of basic typographical terms on different platforms or with different font formats. These differences have always been there, of course, but are less obtrusive in a paper-oriented, non-distributed environment where document generation and document rendering happen on the same computer and where manual verification and correction of the layout is possible before generating a fixed printed result. Panose-1 is a widely used classification scheme based on defined measurements of particular Latin characters. A Panose-1 measurement can be used to select a font which is similar in character to another font, provided both have Panose-1 measurements. Panose-1 could also be used for some non-Latin scripts such as Greek and Cyrillic, but the measurements to do so have not been defined.

Panose-2 is a proposed more extensive classification scheme which can relate different measurements made on glyphs from different scripts. It may be useful in the future for automatically selecting similar fonts for different scripts.

Other problems are due to characteristics being specific to particular scripts. For example, height of flat-topped unaccented lowercase letters (the x-height), as shown in Figure 5, is a very useful indicator of the style of a Latin font, particularly when expressed as a ratio of the x-height to the height of flat-topped capital letters (the cap height). Fonts of dissimilar x-height will look very different.

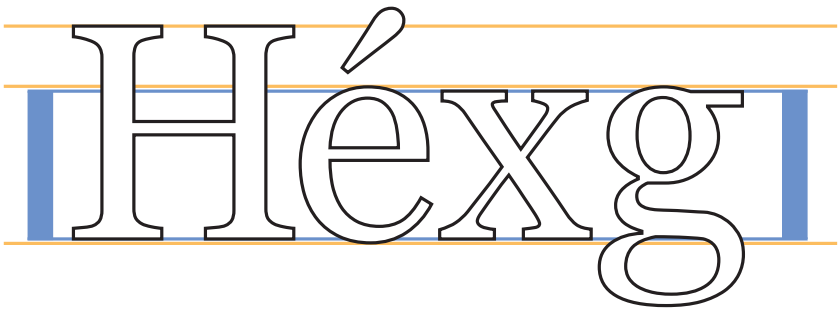

Figure 5 : x-height is a useful characteristic for bicameral fonts

Most scripts are unicameral - they have only a single case. They are often set at a height midway between the x-height and cap-height, as shown in Figure 6. Dropping the x-height and cap-height properties on the grounds that they only apply to some scripts, however, would penalise those scripts for which these characteristics are important. Another alternative is to assign the x-height and cap-height the same values, so that the ratio (ie, 1) can be compared with that of bicameral scripts.

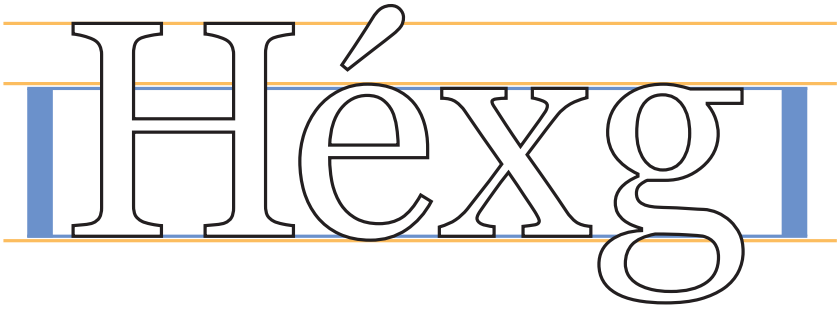

Figure 6 : Glyph from a unicameral script

This allows Latin fonts with low x-height to be chosen for use with Arabic, and Latin fonts with high x-height for use with Hebrew. The best solution is for such matching to be done by the stylesheet designer, using similar or contrasting faces that work well together from a design standpoint. The automatic matching is then a fallback if the requested fonts cannot be used for whatever reason.

There is certainly a need for the designer of a style sheet to be aware of the different scripts that are going to be used in a document and to plan accordingly, and to decide whether close matching on the x-height or on the font size is desirable. Compare the two samples in Figure 7 and Figure 8.

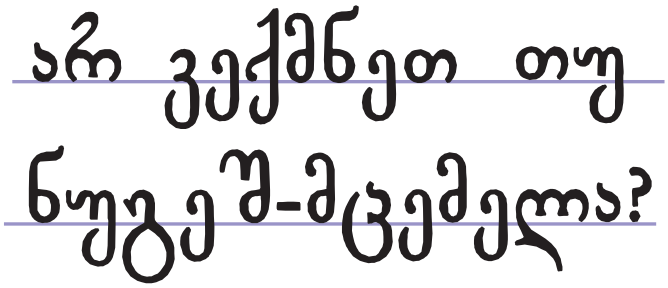

Figure 7 : Georgian sample showing deep descenders

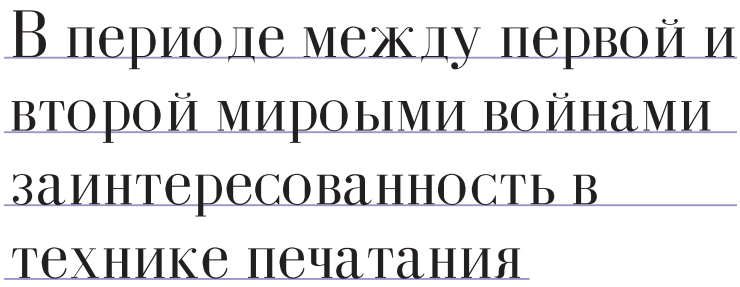

Figure 8 : Russian sample showing shallow descenders

Another characteristic that is script dependent is alignment. Latin scripts have a clearly defined baseline, with descenders. Other scripts have no descenders, or are aligned on a centerline - for example, CJK Unified Ideographs, as seen in Figure 3 - or a top line (hanging baseline) - for example, North Indian scripts. The Web Font extension allows all these alignments to be defined on a per-font basis.

5.3 Font Download

The Web Font extension, now fully integrated into CSS2 and planned for a subsequent Working Draft of XSL, allows URLs to be added to the style sheet which point to fonts. There are techniques such as site locking, digital signatures, and format transformation which can be used to protect the intellectual property rights of the font designers; these techniques are not addressed here.

The stylesheet can also indicate, on a per-font bases, the range of Unicode characters for which it has some glyphs. Most fonts have sparse coverage of Unicode. This property is used to determine whether a font might have glyphs and thus whether to download it or search it.

Other information about the font can be added, such as the size of the design grid, the position of the various baselines (low baseline for Latin, Greek and Cyrillic; central baseline for Ideographic scripts and top baseline for Indic scripts), the x-height and ca-height, Panose-1 number, and so on. Other descriptors can readilly be added to better describe fonts for scripts that are presently not covered so well, such as Arabic.

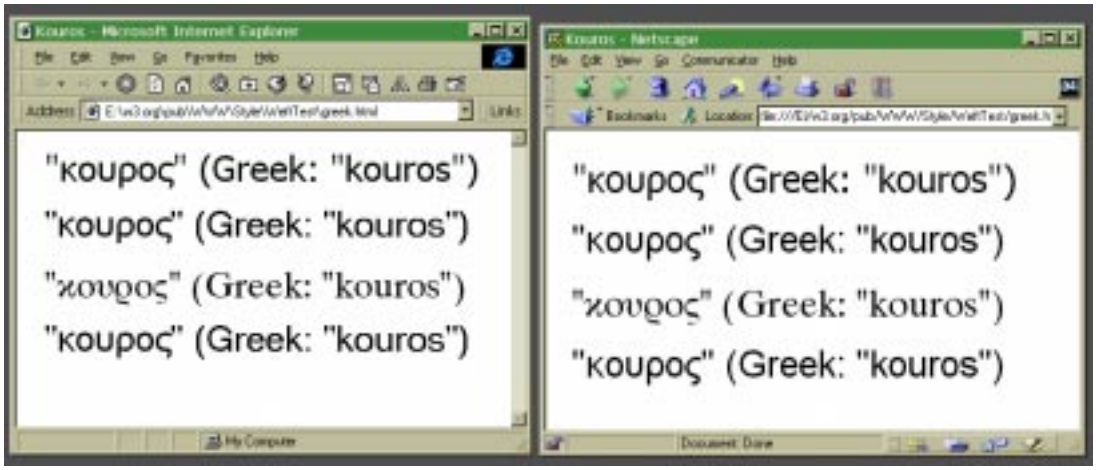

Figure 9 : downloaded Greek fonts in two browsers

Multiple font formats are in use on many different platforms, so in consequence HTTP content negotiation is required for downloadable fonts. The same stylesheet can point to multiple fonts in different formats, indicating which format is available at which URL; this is particularly handy given that the first two implementations of Web Fonts chose different font formats.

This example is for font downloading, no information is provided to enable font synthesis or matching. It defines a composite font split over three files.

```
<STYLE>
  @font-face {
  font-family: Excelsior;
  src: Excelsior Roman, url(http://site/er) font/opentype
  unicode-range: U+00xx /* Latin-1 */
  }
  @font-face {
  font-family: Excelsior;
  src: Excelsior EastA Roman, url(http://site/ear) font/intellifont;
  unicode-range: U+01xx-022x /* Latin Extended A and B */
  }
  @font-face {
  font-family: Excelsior;
  src: Excelsior Cyrillic Upright, url(http://site/ecu) font/truedoc;
  unicode-range: U+04xx /* Cyrillic */
  }
</STYLE>
```

6 Conclusion

Both CSS and XSL allow desired formatting to be expressed in style sheets and applied to XML documents, which are considered to be a collection of Unicode characters which will require multiple writing directions. These style sheet languages are the first steps towards true multilingual typography on the Web.

7 References

[639] ISO 639:1988 - Code for the representation of names of languages - The International Organization for Standardization, 1st edition, 1988 17 pages .

[3166] ISO 3166:1988 - Codes for the representation of names of countries - The International Organization for Standardization, 3rd edition, 1988.

[CSS2] Cascading Style Sheets, level 2, W3C Recommendation
`http://www.w3.org/TR/WD-CSS2`

[Harm] Character Set considered Harmful
`http://www.w3.org/MarkUp/html-spec/charset-harmful.html`

[Pan2] Panose-2 white paper
`http://www.w3.org/Fonts/Panose/pan2.html`

[Self] This paper
`http://www.w3.org/people/chris/IUC13/multilingrend.pdf`

[XML] Extensible Markup Language, W3C Recomendation
`http://www.w3.org/TR/REC-xml`

[XSL] Extensible Style Language, Working Draft
`http://www.w3.org/TR/WD-xsl`