Objectives

- Describe the standards that define the architecture & principles for I18N on the web
- Scope limited to markup languages
- Provide practical advice for working with international data on the web, including the design and implementation of multilingual web sites and localization considerations
- Be introductory level

Legend For This Presentation

Icons used to indicate current product support:

- Supported:
- Partially supported:
- Not supported:

Web Internationalization Agenda

- Emphasis on Character Processing
- Updates for HTML5

This presentation and part 2 and example code are available at:

www.xencraft.com/training/webstandards.html

Richard Ishida and W3C test I18n features for numerous browsers and versions (X)HTML:

www.w3.org/International/tests/
Web Internationalization – Standards and Practice

Web I18n Part 1- Character Processing

Character Encodings
Character Encoding Negotiation
Reference Processing Model
Character Escaping
Unicode in Markup
Normalization
Identifiers

A Simple HTML Example Page

Nous espérons que vos applications e-business fonctionneront en français.
Les Français achètent les produits suivants sur internet:
vêtements
produits de beauté

We hope your e-business applications will work in French.
French people buy the following products on the internet:
clothing
beauty products

Don’t forget to put prices in Euros (€) not Dollars ($)
A Simple HTML Example Page

Here is how the same HTML looks in Japan

Nous espérons que vos applications en français les Français achètent les produits de beauté.

We hope your e-business succeeds.

French people buy the following clothing, beauty products.

Don’t forget to put prices in Euros (€) not Dollars ($).

Some of the problems may not be obvious to the reader.
Changing the euro symbol to a bullet, might cause a significant financial error.

Character Encodings

Encoding disagreement is one problem for text. We also consider the following problems and solutions.

(See: Character Model for the World Wide Web
www.w3.org/TR/charmod/)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding disagreement</td>
<td>Encoding negotiation</td>
</tr>
<tr>
<td>Encoding diversity</td>
<td>Reference processing model</td>
</tr>
<tr>
<td>Encoding limitations</td>
<td>Character escaping</td>
</tr>
<tr>
<td>Unicode vs. markup</td>
<td>Markup preferred on the web</td>
</tr>
<tr>
<td>String matching</td>
<td>Early uniform normalization</td>
</tr>
<tr>
<td>String indexing</td>
<td>Character counting guidelines</td>
</tr>
</tbody>
</table>

Character Encodings

First though: What are character encodings?

Letters: ABCDEFG…abcdedef…ÅÅÅÆßÑ

Punctuation: . , ; ? ! ( ) – ...

Numeric, Arithmetic: 0 1 2 3 4 5 6 7 8 9 + – ± * / < = ≥ > % %0 # ¼ ½ ¾ ⅓ ⅔ ⅓ ⅔ IV XII

Business: $ ¢ £ ¥ € ¥ ℅ ℂ ℉ ℉ ℆ ℅ ° ∞

Mathematics: ∀ ∃ ∏ ∫ ≤ ≥ ≡ ≠ ≈ ∝ ± \⊕ ⊞ ⊙ ⊠ ⊣ \oplus \cup ⊃ ⊇ ⊆ ⊇ ⊆ ⊆ ⊇

Other applications: Proofreading, games, music...

The Character Encoding Model – Unicode Tech. Report 17
Character Encodings

ACR = Abstract Character Repertoire

- The set of characters you need to represent
  - (aka Character Set).
- Characters may be composable. E.g. Å = A + °

Character Encodings

CCS = Coded Character Set

Maps each character to a non-negative unique number.
- Note this example uses hexadecimal numbers.
- The “U+” indicates use of Unicode’s numbering.
- The grapheme Å consists of two characters A + °
- Unicode calls these “Unicode Scalar Values”

Character Encodings

CEF = Character Encoding Form

Note the relationship between the CEFs is not so simple.
Map CCS to fixed width units (e.g. 32, 16, or 8-bit)

Character Encodings

Map CCS to fixed width units (e.g. 32, 16, or 8-bit)
Note the relationship between the CEFs is not so simple.
Character Encodings

<table>
<thead>
<tr>
<th>CES = Character Encoding Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
</tr>
<tr>
<td>CCS</td>
</tr>
<tr>
<td>CEF 16-bit</td>
</tr>
<tr>
<td>CES UTF-16BE</td>
</tr>
</tbody>
</table>

CES: Mapping the CEF(s) to serialization of bytes

Character Encodings

- Many character sets exist and in popular use
- Many encoding schemes, even for 1 character set
  - ISO 8859-1 ≈ IBM 850
  - UTF-8 = UTF-16 = UTF-32

- Given just bytes, the character set and the encoding scheme can be indeterminate.

How can a browser know how to decode a web page?

Encoding Identification

- Given just bytes, encoding is indeterminate.
- How can an encoding be identified?

- There are 2 requirements:
  - Agreement on names for encodings
  - Mechanisms for labeling text with encoding

Character Encoding Names

IANA (Internet Assigned Numbers Authority)
- Maintains registry of official names for character sets (actually encodings) used on the internet and in MIME (mail)
- Registry Names
  - ASCII, printable characters
  - Case-insensitive
  - Maximum length 40 characters
  - Aliases (alternative names) are also registered
  - The preferred name is indicated

www.iana.org/assignments/character-sets
### Unregistered Encoding Names

- Conventions for Unregistered Character Encoding Names
  - Name begins with “x-”
  - Example: “x-Tex-Yves-encoding”
  - Useful for private encodings or very new encodings

**Not useful on the web, except for private exchange**

### Character Encoding Names

- IANA Name and Alias Examples
  - ISO_8859-1:1987 (ISO_8859-1, ISO-8859-1, latin1, L1, IBM819, CP819, csISOLatin1)
  - Windows-1252, GB2312, BIG5, BIG5-HKSCS
  - SHIFT_JIS, HP-Legal
  - Extended_UNIX_Code_Packed_Format_for_Japanese
  - Adobe-standard-encoding
  - UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32

- Registry contains many useless names
- Preferred names indicated. Use them.

### Markup and Encoding Names

- HTTP
- HTML
- XML
- CSS
- Links
  - HTML <LINK>
  - HTML <... HREF>
  - XML <... HREF>

### HTTP and Encoding Names

**Mechanism for labeling HTTP with encoding**

**HTTP Response**

```
200 OK HTTP/1.1
Content-Type: text/html; charset=UTF-8
--- Blank Line
document
...```

---
HTML, XML & Encoding Names

**HTML**

```html
<META HTTP-EQUIV="Content-type" CONTENT="text/html; charset=UTF-8">
```

- HTML does not specify a default.

**XML**

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

- Alternative declaration: Begin with **Byte Order Mark (U+FEFF)**, for UTF-16 or UTF-8
- Note UTF-16 MUST begin with a BOM
- The **default** encoding is UTF-8.

New in HTML 5: `<Meta Charset=>`

```html
<meta charset="UTF-8">
```

- Must be in the first 512 bytes of the page
- Use “preferred MIME name” in IANA registry
- Use BOM instead for UTF-16.
- Supported by most browsers
- Simpler and less error-prone then

```html
<META HTTP-EQUIV="Content-type" CONTENT="text/html; charset=UTF-8">
```

- Byte Order Mark now recognized
- UTF-32, EBCDIC, others not recommended
- UTF-7, SCSU, et al must not be supported

CSS2 and Encoding Name

**CSS2**

- Only used in the **first line** of external style sheets

```css
@charset "UTF-8";
```

- CSS 2.1 added Unicode Byte Order Mark (BOM, U+FEFF) as an encoding indicator.
- Encoding is unspecified if BOM and @charset conflict.

LINKs and Encoding Name

Declaring the charset of a LINKed document

- **HTML**

```html
<Link title="Arabic text" type="text/html" charset="ISO-8859-6" rel="alternate" href="arabic.html">
```

- **XML**

```xml
<?xml--stylesheet href="…" type="…" charset="UTF-16" ?>
```

Internationalization and Unicode Conference 7

Tex Texin, XenCraft
Notes: Declaring Encoding Names

- Charset on links can be incorrect if the document’s encoding on the server changes.
  - The encoding for the `<META… charset=…>` is unknown until the statement is processed.
    - ASCII is recommended for this statement.
    - Place it as early as possible in the document.
    - Else, prior statements may be decoded incorrectly.
- Note:
  - Transcoders do not generally correct charset ID.

HTML5: LINKs and Encoding Name

Declaring the charset of a LINKed document
- Deprecated in HTML5

```html
<LINK title="Arabic text" type="text/html" charset="ISO-8859-6" rel="alternate" href="arabic.html">
```

```html
<a href="http://www.unicode.org" charset="UTF-8">Unicode</a>
```

---

HTML4 Encoding Priorities

- Prioritization is used to resolve conflicts.
- From high to low priority, HTML uses the encoding of:
  1. HTTP “Content-Type” charset
  2. `<META http-equiv “Content-Type” charset>`
  3. LINK or other syntax for external documents
  4. Charset-detecting heuristics

- Many user agents (browsers) support a user override for charset (highest priority)

---

HTML5 Encoding Priorities

- From high to low priority, HTML5 uses the encoding of:
  1. User override for charset
  2. HTTP “Content-Type” charset
  3. Byte Order Mark
  4. Either (Must only be one)
    - `<META http-equiv “Content-Type” charset…>`
    - `<META charset= “UTF-8”>`
  5. Character set-detecting heuristics
CSS2 Encoding Priorities

Prioritization is used to resolve conflicts.
- From high to low priority, CSS 2.1 external style sheets use the encoding of:
  1. HTTP “Content-Type” charset
  2. BOM/@charset rule in the style sheet
  3. LINK or other syntax in referencing document
  4. Charset of the referencing document
  5. Assume UTF-8

XML Encoding Priorities

- Encoding name processing is more carefully specified for XML.
- As with HTML, protocol or external information can supercede declaration, BOM or default of UTF-8.
- XML Appendix E (non-normative): Prioritization should be specified by protocols.
  - Recommends use of BOM or encoding declaration for files (rather than an external source).
  - Refers to RFC 3023
    - RFC 3023 specifies several encoding scenarios based on MIME media type: text/xml, application/xml, etc.

Web I18n Part 1 - Character Processing

Character Encodings
Character Encoding Negotiation
Reference Processing Model
Character Escaping
Unicode in Markup
Normalization
Identifiers

Character Encoding Negotiation

Unix user
GB2312
html

Windows user
1252
html

Which encoding is served?
Chinese
GB2312
Norwegian
BIG5
UTF-8
ISO 8859-1
ISO 8859-15
Typical Browser-Server HTTP Sequence

1. Browser issues GET URL
2. Server sends RESPONSE
3. Browser displays document in RESPONSE
4. Browser POSTs Form with user data (text)
5. Web Server receives data, database application stores text.

Which encoding is sent by the server?
Which encoding is returned by the browser?

Character Encoding Negotiation

- Most browsers let you set your language preferences and priorities
- Encoding capabilities are not settable (since they are software dependent).
- Microsoft IE doesn’t send ACCEPT-CHARSET.
- (U.S.) NS 7: ISO-8859-1, UTF-8;q=0.66, *;q=0.66
- Opera 6.0 sends:
  Windows-1252;q=1.0, UTF-8;q=1.0, UTF-16; q=1.0, iso-8859-1;q=0.6, *;q=0.1

Character Encoding Negotiation

GET / HTTP/1.1
Accept-Language: en-us,en,hr;q=0.5
Accept-Charset: iso-8859-1,utf-8;q=0.75,*,q=0.5

The browser’s HTTP GET request can list the languages and the encodings it can make use of, to guide the server.
- “q” is a relative measure of the usefulness (quality) of an entry.

The above example indicates:
- US English preferred, other English, Croatian are also ok.
- ISO 8859-1 preferred, then UTF-8, then anything else.

Character Encoding Negotiation

200 OK HTTP/1.1
Content-Type: text/html; charset=iso-8859-1

Blind Line
HTML document
...

The server returns a document.
The encoding is declared in the RESPONSE header.
(Web administrators or content authors need to inform the server about document encodings.)
The browser adapts the document for operating System display.

Form Data Set

```html
<form name="input" method="GET" action="http://www.xencraft.com/cgitest"
    enctype="application/x-www-form-urlencoded">
    Name: <input type="text" name="Name" size="10" /> Name/Tex  
    <input type="radio" name="sex" value="m"> Male  
    <input type="radio" name="sex" value="f"> Female  
    <input type="submit" value="Send">  
</form>
```

Form Data Set Submission

3 Submission Methods

- GET + HTTP URI
  Form Data Set appended to URI +"?" encoded as  
    - “application/x-www-form-urlencoded“

- POST + HTTP URI
  Form Data Set sent in body, encoded as either  
    1) “application/x-www-form-urlencoded“ or  
    2) “multipart/form-data“ (MIME, RFC 2045)
Form Data Set- GET Method Submission

```html
<form name="input" method="GET" action="http://www.xencraft.com/cgitest" enctype="application/x-www-form-urlencoded">
  Name: <input type="text" name="Name" size="10"/>
  <input type="radio" name="sex" value="m"> Male
  <input type="radio" name="sex" value="f"> Female
  <input type="submit" value="Send">
</form>
```

This simple form will submit an HTTP GET with:
http://www.xencraft.com/cgitest?Name=Tex&sex=m

---

Form Data Set Encoding

Application/x-www-form-urlencoded

Name=Value&Name2=Value2&Name3=Value3

- Pairs of control names and current values.
- Names separated from values by =
- Name/value pairs separated by &
- Spaces replaced by +
- Line breaks represented as CR LF: %0D%0A
- Non-alphanumeric and non-ASCII characters and ‘+’, ‘&’, ‘=’, are replaced by %HH
- Browsers map current encoding byte values to %HH
- If the server doesn’t know browser’s character encoding, it may decode form data incorrectly.

---

Character Encoding Negotiation

Accept Charset x,y,z

Browser Get URL Server CHARSET=x

Browser Response Server CHARSET=x

Submit Form (GET or POST)

Browser

O/S Charset =z encoding=x-www-form-urlencoded

Modern browsers send x-www-form-urlencoded data to the server in the CHARSET that was determined to be that of the *form*, however that determination was made (HTTP, <meta>, default, user override).
Character Encoding Negotiation

Returning data in the encoding received
- Generally works in principle
- Document ‘charset’ must be correctly identified (and has often been wrong)
- Fails with multiple encodings handled by a single CGI
- Fails with transcoding proxies (not allowed to change URIs).
- **Recommend using UTF-8 in both directions**

Character Encoding Negotiation

<table>
<thead>
<tr>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get URL</td>
<td>CHARSET=x</td>
</tr>
<tr>
<td>Response</td>
<td>CHARSET=x</td>
</tr>
<tr>
<td>Submit Form (POST)</td>
<td>multipart/form-data (MIME)</td>
</tr>
</tbody>
</table>

Each control name/current value pair is a separate part. Each part can be a different charset or content-type encoding. Supports file uploading (RFC1867).

Character Encoding Negotiation

Multipart/form-data
- More efficient than x-www-form-urlencoded for non-ASCII data, binary data, and files
- Does not have the length limit that browsers impose on URLs (can be as low as 250 for some devices)
- Is now well supported
- Recommended for POST of all form data

Character Encoding Negotiation

Other solutions to identifying encodings:
- XFORMS fixes the failure cases:
  - http://www.w3.org/MarkUp/Forms/
  - http://www.w3.org/TR/xforms/
- **TIP**: Use with older browsers:
  - Hidden fields containing encoding name or carefully chosen text (tracks transcodings). CGI script performs analysis.
    - e.g. Microsoft’s `_CHARSET_`
Web I18n Part 1 - Character Processing

- Character Encodings
- Character Encoding Negotiation
- Reference Processing Model
- Character Escaping
- Unicode in Markup
- Normalization
- Identifiers

Reference Processing Model

- Different encoding schemes require different decoding/parsing/processing methods
  - Single, and Multi-byte character sets (e.g. EUC)
  - Character encoding-switching schemes (ISO 2022)
  - Forward combining (accent-base letter)
  - Backward combining (base letter-accent)
  - Logical ordering/Visual ordering

- Variety bothers implementers and spec writers
- Adopting a single universal encoding obsoletes most of the existing data
- Instead, use a character abstraction

Reference Processing Model

- Logically, characters are Unicode characters
  - Specifications are in terms of Unicode characters
  - Implementations do NOT have to use Unicode, only behave as if they did

- Benefits
  - Removes ambiguity, simplifies specifications
  - Allows flexibility for common local encodings
  - Backward compatible for older HTML browsers
  - Supports internationalization (large character set)
  - Removes dependencies/orientation on byte values

Reference Processing Model

- Abstraction Layer using Unicode
- Any encoding on the wire
- In/out
- HTML
- XML
- CSS
- Abstraction Layer using Unicode
- Any encoding for internal implementation
- Internal

Internationalization and Unicode Conference 14

Tex Texin, XenCraft
Web Internationalization – Standards and Practice

Reference Processing Model

• Examples using Reference Processing Model
  – HTML 4.0 declares Unicode as its SGML Document Character Set
  – CSS “sequence of characters from UCS”
  – XML “A character is an atomic unit of text as specified by ISO/IEC 10646”
• Any encoding can be used internally, but Unicode often makes the most sense.
• XML requires parsers to accept UTF-8 and UTF-16, making Unicode best internal choice
• Some Recommendations require Unicode
  – e.g. DOM requires UTF-16

Web I18n Part 1 - Character Processing

Character Encodings
Character Encoding Negotiation
Reference Processing Model
Character Escaping
Unicode in Markup
Normalization
Identifiers

Character Escaping

Mechanisms to represent characters
• Numeric Character References (NCRs)
  – HTML and XML
    Hexadecimal: &#xhhhhhh;
    Decimal &d#ddd;
  – CSS2
    “hh” (note terminating space), \hhhhhh
• Character Entity References (HTML only)
  &aring; &Aring; (note case-sensitivity)

Character Escaping

• Useful for:
  – syntax-significant characters
  – e.g. &lt; (<), &gt; (>), &amp; (&), &quot; (“)
  – characters outside current encoding
  – eliminating visual or other ambiguity
    &#x00AD; (soft-hyphen),
    &#x002D; (hyphen-minus)
    &x0020; (space)
    &x00A0; (no-break space)
Character Escaping

- Relies on Reference Processing Model
  - Always references Unicode scalar value
    - Same value regardless of encoding
    - Same value for UTF-8, UTF-16, UTF-32
    - One value for supplementary characters, not two (e.g. &x12345; not &xD800; &xDF45);
  - Simplifies transcoding (no parsing or conversion)
  - Allows any Unicode character in any document (if it is legal in the language of the document)

Don’t use Windows 1252 code points instead of Unicode, for values 128-159 (0x80-0x9F)

- e.g. Euro is &#8364; or &euro; not &#128;
- www.i18nguy.com/markup/ncrs.html

Don’t simulate characters with special fonts (e.g. Symbol), or you can get erroneous:

- Display, depending on font availability
- Font fallbacks
- Searches by Search engines
- Behavior from Style sheets
- Database contents

Windows 1252 vs Unicode & ISO 8859-1

1252 is identical to Unicode and ISO 8859-1 except in 80-9F. Unicode and ISO 8859-1 assigns control codes. Windows-1252 assigns Euro, Smart quotes, TM, and others.

Selecting A Character Encoding

Choose an encoding that minimizes the need to escape characters.

- Unicode is always a candidate.
- Unicode is supported by all but the oldest browsers.
- Is the largest character set, and can be expanded.
- Therefore it is often the best choice both for minimizing escapes and anticipating future character requirements.
- e.g. New currency symbols
Web Internationalization – Standards and Practice

Web I18n Part 1- Character Processing

- Character Encodings
- Character Encoding Negotiation
- Reference Processing Model
- Character Escaping
- Unicode in Markup
  - Normalization
  - Identifiers

Unicode Vs. Markup

- +100,000 characters as of Unicode 5.0
  - Should we use them all?
  - Are there any we shouldn’t use?
  - Does Unicode’s capabilities, needed for plain text, interfere with markup?
- Markup can do some things better than character codes. Not all Unicode characters are needed.

Unicode Vs. Markup

Potential problem areas
- Redundancies impact searching
  - “Å” A-ring “Å+” A+ring “Å” Angstrom
- Formatting characters vs. Markup
  - E.g. Bidi controls, interlinear annotation characters
- Characters with style vs. Markup
  - E.g. Superscript, subscript
- Object Replacement Character vs. Markup
  - Better to use markup to include an image

Solution types
- Restrict characters so they cannot be used
- Replace redundancies (normalization)
- Replace with Markup
  - Extensible
  - Presentation can be separate from content

Joint W3C and Unicode recommendations in:
“Unicode in XML and other Markup Languages”
http://www.w3.org/TR/unicode-xml/
http://www.unicode.org/unicode/reports/tr20/
Web Internationalization – Standards and Practice

Web I18n Part 1 - Character Processing

Character Encodings
Character Encoding Negotiation
Reference Processing Model
Character Escaping
Unicode in Markup

String Indexing

Normalization
Identifiers

String Indexing

• How long is this string? Å ≠ A

String Indexing

• Which units should be used for counting?

Graphemes: 3
Characters: 4
Code units: 5
Bytes: 10

Character Model recommendations

• Character counting is recommended for most programming interfaces (e.g. XML Path)
• Code unit counting may be used for internal efficiency (e.g. DOM)
• Graphemes may be useful for user interaction, once a suitable definition exists

CAUTION: Avoid creating API with single unit arguments
e.g. “SS” = Uppercase(“ß”)
Normalization

- Representing data in more than 1 way leads to errors
- E.g. The Mars Climate Orbiter mission was disastrous. Information expected to be metric, was sent in English units
- Solution- Adopt a standard representation-

Normalization

Early Uniform Normalization

Unicode characters can have more than 1 representation

• Canonical equivalence
  - Indistinguishable, fundamental equivalence
  - E.g. combining sequences, singletons
  - “Å” U+00C5 (A-ring pre-composed)
  - “Å” U+0041 + U+030A (A + combining ring above)
  - “Å” U+212B (Angstrom)

• Compatibility equivalence
  - E.g. Formatting differences, ligatures
  - “ｶ” U+FF76 “ｶ” U+30AB (KA half and full width)
  - “fi” U+FB01 (ligature fi)

Early Uniform Normalization

• Unicode Consortium has defined canonical and compatibility decomposition formats and 4 different sets of rules for normalization:

  “Unicode Normalization Forms”
  http://www.unicode.org/unicode/reports/tr15/

• The W3C Character Model recommends Normalization Form C (NFC)
  - Brings canonical equivalences to composed form
  - Leaves compatibility forms as distinct
  - Most legacy text is composed, and is unchanged

Early Uniform Normalization

Text on the web SHOULD be Fully Normalized.

Fully Normalized text is either:

1. Unicode text in Normalization Form NFC, and
2. Does not contain character escapes or includes that upon expansion would undo point 1, and
3. Does not begin with a composing character.

or:

1. Legacy encoded text, which transcoded to Unicode satisfies the above.
Early Uniform Normalization

- Examples of Fully Normalized Text
  - “suçon”, “su&#xE7;on”
  - “sub,on”, “sub&#x0327;on”
  - Note: Unicode does not have a composed b-cedilla.

- Examples that are not Fully Normalized
  - “suc,on”, “suc&#x0327;on”
  - Reason: should use composed character “ç”
  - “,on”, “&x0327;on”
  - Reason: should not begin with combining character

New “International” Features in HTML5

- Some attributes that now apply to all elements:
  - dir (Direction)
  - lang (Language)

- lang attribute now supports empty string indicating the primary language is unknown

- xml:lang supported, provided it has same value as lang

- hreflang attribute added to <area>
  - For consistency with <a> and <link>

New “International” Features in HTML5

- Ruby annotation support: <ruby>, <rt>, <rp>

- Native support for IRI and IDNA
  - For IRI, document encoding must be UTF-8 or UTF-16 and query component must %hh escape any non-ASCII characters

- Encodings
  - New encoding declaration <Meta charset>
  - Support for BOM
  - Charset attribute removed on <link> and <a>
Questions

Web Internationalization Agenda

- Part 1 – Character Processing
- Coffee Break
- Part 2
  - Layout and Typography
  - Designing International Web Sites

Language Identification

Same mechanism for all:
- Tags (identifiers) defined by RFC 3066.
  2-letter and 3-letter language codes (ISO-639) with optional 2-letter country codes (ISO-3166) separated by a character ‘-’ (not ‘_’).
- Tags are case insensitive (even in XML).
- In mark-up: the language attribute is inherited by the children of the element where the attribute is defined.
Language Identification

RFC 3066 Rules:
• 3-letter codes should be used only for the languages that have no 2-letter code.
• Always use the Terminological form of the 3-letter codes, not the Bibliographical form.
• Avoid user-defined codes (x-myCode)

Language Identification

• RFC 3066 does not cover all needs.
  – e.g. Latin-Amer. Spanish, Script distinctions
  – Addressed case-by case through registrations
• No clear distinction of the identifiers of a "language" and a "locale".
  – (See past IUC locales talks for more information.)
• Standards groups considering these issues: IETF, ISO TC37, SIL, W3C, et al

Language Identification: BCP47

• BCP47 evolution
  – RFC 4646, 4647
    • RFC 4646 replaces RFC 3066.
    • language-country becomes language-script-country
    • Registry expanded to include all valid entries
    • New matching rules in RFC 4647
  – RFC 5646 just released
    • 7,000 three-letter ISO 639-3, ISO 639-5 language codes, 7 region codes
    • 220 'extended language' subtags, for backwards compatibility.

Language Identification

• HTTP: Content-Language header
• HTML: LANG attribute (e.g. in <html>)
• XML: xml:lang attribute
• XHTML 1.0: Both lang and xml:lang
  <p xml:lang="la" lang="la">Verba.</p>
• XHTML 1.1: xml:lang attribute
Language Identification

The `lang()` function in XPath:
- True if the selected node has `xml:lang` set to the given language code.
- Match is done as a sub-string from the start of the value: 'en' matches 'en', and 'en-us'.
- Match is case insensitive: 'en' matches 'EN', 'En-us', etc.

Example: Input, Languages.xsl, Output.

Language Identification – Input file

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="Languages.xsl"?>
<MyData>
  <Msg id="100">
    <Text xml:lang="en">Message 100 in English.</Text>
  </Msg>
  <Msg id="200">
  </Msg>
  <Msg id="300">
    <Text xml:lang="fr">Message 300 en québecouis.</Text>
  </Msg>
  <Msg id="400">
    <Text xml:lang="EN-GB">Message 400 in British English.</Text>
  </Msg>
</MyData>
```

Language Identification – Style-sheet

```xml
<?xml version="1.0" ?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:param name="Language">en</xsl:param>
  <xsl:template match="text()"/>
  <xsl:template match="Text">
    <xsl:if test="lang($Language)">
      <p><xsl:value-of select="."/></p>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Language Identification – IE Output

Message 100 in English. (en)
Message 200 [insertion in French] in American English. (en-us)
Message 400 in British English. (EN-GB)
Language Identification – FF Output


Language Identification – Opera Output


Language Identification – CSS

There are two methods to refer to the language attribute in CSS:

- The `lang` pseudo-class.
  ```css
  *:lang(zh) { font-family:SimSun }
  ```

- The attribute selector.
  ```css
  *[lang|=fr] { font-weight: bold }
  ```

- Both use the same matching mechanism as the `lang()` function in XPath.

  ➤ Example: LanguagesCSS.htm

Language Identification – CSS

```html
<html lang="en">
<head>
  <style>
    *:lang(en-us) { font-weight: bold; }
    *[lang|=fr] { font-style: italic; color: red; }
  </style>
  <title>Test Language and CSS</title>
</head>
<body>
<p>Text in English.</p>
<p lang="en-us">Text in American English.</p>
<p lang="en">Text in generic English.</p>
<p lang="fr-ca">Texte en québecquois.</p>
<p lang="fr">Texte en français.</p>
<p lang="en-gb">Text in British English.</p>
</body>
</html>
```
Language Identification – FF Output

Text in English.
Text in American English.
Text in generic English.
Texte en québecquois.
Texte en français.
Text in British English.

Quotes – HTML

- The `<q>` element for in-line quotations (auto-quotation marks expected).
- The `<blockquote>` element for paragraph-type quotations (indented, and no auto-quotation marks expected).

=> Example: Input, Output: Quotes.htm.

Quotes – Using CSS

- CSS allows control of the type of quote to use according to the language.

```
*{lang|=fr} { quote:\ab\a0 'a0\bb }
qo:before    { content:open-quote }
qo:after     { content:close-quote }
```

- Examples
  => HTML: Input, CSS, Output: QuotesWithCSS.htm.

Quotes – HTML Input

```
...<body>
  <p lang="en">English text with <q>English quoted text</q>.</p>
  <p lang="fr">Text en Français avec <q>English quoted text</q>.</p>
  <p lang="fr">Text en Français avec <q lang="en">English quoted text containing a <q>quote</q> itself</q></p>
  <p lang="fi"><q>Quotes</q> in Finnish.</p>
  <p lang="pl"><q>Quotes</q> in Polish.</p>
  <p lang="ja"><q>Quotes</q> in Japanese.</p>
  <p lang="de"><q>Quotes</q> in German.</p>
  <p lang="nl"><q>Quotes</q> in Dutch.</p>
  <blockquote lang="fr">A paragraph using blockquote.</blockquote>
</body>
</html>
```
### Some Unicode Characters

- U+2018 ‘  Left Single Quotation Mark
- U+2019 ’  Right Single Quotation Mark
- U+201C “  Left Double Quotation Mark
- U+201D ”  Right Double Quotation Mark
- U+201E „  Double Low 9 Quotation Mark
- U+201F „  Double High Reversed 9 Q. M.
- U+300C 「  Left Corner Bracket
- U+300D 」  Right Corner Bracket
- U+00AB «  Left Pointing Double Angle Q. M.
- U+00BB »  Right Pointing Double Angle Q. M.
- U+00A0  No Break Space

### Quotes – CSS Style-sheet

```css
q:before { content: open-quote; }
q:after { content: close-quote; }
blockquote:before { content: open-quote; }
blockquote:after { content: close-quote; }

[lang|='en'] > * { /* English */ quotes: "\201C" "\201D" }
[lang|='fr'] > * { /*guillamets*/ quotes: "\AB\A0" "\A0\BB" }
[lang|='fi'] > * { /*same direction*/ quotes: "\201D" "\201D" }
[lang|='de'] > * { /* German */ quotes: "\201E" "\201C" }
[lang|='ja'] > * { /* Japanese */ quotes: "\300C" "\300D" }
[lang|='nl'] > * { /* Dutch */ quotes: "\2018" "\2019" }
[lang|='pl'] > * { /* Polish */ quotes: "\201E" "\201D" }
```

### Quotes – Firefox 3 Output

English text with “English quoted text”.
Text en Français avec « English quoted text ».
Text en Français avec « English quoted text containing a “quote” itself ».
”Quotes” in Finnish.
„Quotes” in Polish.
「Quotes」 in Japanese.
„Quotes“ in German.
‘Quotes’ in Dutch.
“A paragraph using blockquote.”

### Quotes – Opera Output

English text with “English quoted text”.
Text en Français avec « English quoted text ».
Text en Français avec « English quoted text containing a "quote" itself ».
”Quotes” in Finnish.
„Quotes” in Polish.
「Quotes」 in Japanese.
„Quotes“ in German.
‘Quotes’ in Dutch.
“A paragraph using blockquote.”
Casing

- CSS2 provides the property `text-transform` with 5 values: uppercase, lowercase, capitalize, none, and inherit.
- CSS2 allows user agents to ignore it for non Latin-1 characters and for unusual case conversion (making it useless from an i18n viewpoint). CSS3 (working draft) forces Unicode casing conformance. This property is deprecated in XSL 1.0.

Casing – Firefox Output

Original = This text should be all uppercased.
Transformed = THIS TEXT SHOULD BE ALL UPPERCASED.

Original = THIS TEXT SHOULD BE ALL LOWERCASED.
Transformed = this text should be all lowercased.

Original = THIS TEXT SHOULD BE CAPITALIZED.
Transformed = THIS TEXT SHOULD BE CAPITALIZED.

Original 1 = tHIS tEXT sHOULD bE cAPITALIZED.
Original 2 = this text should be capitalized.

[de] Original = ß (sharp-s), ö (o-diaeresis)
Transformed = SS (SHARP-S), Ö (O-DIAERESIS)

[tr] Original = i (i-with-dot)
Transformed = I (I-WITH-DOT)

Casing – Clipboard Copy (unchanged)

Original = This text should be all uppercased.
Transformed = This text should be all uppercased.

Original = THIS TEXT SHOULD BE ALL LOWERCASED.
Transformed = THIS TEXT SHOULD BE ALL LOWERCASED.

Original 1 = tHIS tEXT sHOULD bE cAPITALIZED.
Original 2 = this text should be capitalized.

[de] Original = ß (sharp-s), ö (o-diaeresis)
Transformed = ß (sharp-s), ö (o-diaeresis)

[tr] Original = i (i-with-dot)
Transformed = i (i-with-dot)

Numbered Lists

With CSS2

• CSS2 offers the list-style-type property to specify the type of numbers for lists. Supports only a limited set of pre-defined styles (e.g. has Armenian but not Thai).

⇒ Example NumberedLists.htm

Numbered Lists

...<head>
<style>
.list_heb {list-style-type:hebrew}
.list_geo {list-style-type:georgian}
.list_arm {list-style-type:armenian}
.list_cjk {list-style-type:cjk-ideographic}
</style>
</head>
<body>...
Numbered Lists

With XSL

- XSL provides more flexibility as the format and the type of the numbers can be changed using `<xsl:number/>`.

→ Example: Input, ListNumbers.xsl, Output.

Numbered Lists – Firefox Output

List numbered in Hebrew: List numbered in Armenian:

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6

List numbered in Georgian: List numbered in Han character (cjk-ideographic):

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6

Number Formatting

The function `format-number()` in XSL allows the formatting of numbers based on a given pattern.

- Uses same patterns as Java 1.1 `java.text.DecimalFormat` patterns.
- Use `<xsl:decimal-format/>` to overwrite the default symbols (i.e. decimal separator, grouping separator, etc.).

→ Example: Input, XSL File, Output.

Numbered Lists

... `<body>`

<p>List numbered in Hebrew:</p>

```html
<ol class="list_heb">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
  <li>Item 6</li>
</ol>
```

<p>List numbered in Georgian:</p>

```html
<ol class="list_geo">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
  <li>Item 6</li>
</ol>
```

<p>List numbered in Armenian:</p>

```html
<ol class="list_arm">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
  <li>Item 6</li>
</ol>
```

<p>List numbered in Han character (cjk-ideographic):</p>

```html
<ol class="list_cjk">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
  <li>Item 6</li>
</ol>
```
Bi-directional Text in HTML

- The `dir` attribute:
  - `dir="ltr"` (default), `dir="rtl"`
  - Affects the default value of `align`.
  - Inherited (use it in `<html>` to set the base for the whole document).

- The `<bdo>` element:
  - Overrides implicit directional properties of content.
  - Requires the `dir` attribute.

Bi-directional Text for XML (CSS2)

- Use the `direction` and `unicode-bidi` properties. The `unicode-bidi` property specifies the behavior for inline levels elements (15 maximum levels of embedding).
- Based on Unicode bidi algorithm (UAX#9)

```css
para.bidi { direction:rtl; unicode-bidi:embed }
```

Example: BidiText.htm

---

Text Flow – Bidi Example Source (1/2)

```html
<p style="direction:rtl; unicode-bidi:embed">
Using CSS:<br/>
חברת Pepper Creek LLC,<br/>
שנוסדה זה-עתה, מונה יותר מ-550 עובדים.
</p>

<p dir="rtl">
Using dir="rtl":<br/>
חברת Pepper Creek LLC,<br/>
שנוסדה זה-עתה, מונה יותר מ-550 עובדים.
</p>
```

Text Flow – Bidi Example Source (2/2)

```html
<p dir="rtl">
<span dir="ltr">
Using dir="ltr-
span":</span>
</p>

<p dir="rtl" (wrong):<br/>
חברת Pepper Creek LLC,<br/>
שנוסדה זה-עתה, מונה יותר מ-550 עובדים.
</p>

```html
<p dir="ltr">
Using dir="ltr" (wrong):<br/>
חברת Pepper Creek LLC,<br/>
שנוסדה זה-עתה, מונה יותר מ-550 עובדים.
</p>
```
Text Flow – Bidi Output

: Using CSS
Pepper Creek LLC, שמות זה-עשתה, מזון ירח מ-550 שבידך.
"Using dir="rtl"
Pepper Creek LLC, שמות זה-עשתה, מזון ירח מ-550 שבידך.
Using dir="ltr-span":
Pepper Creek LLC, שמות זה-עשתה, מזון ירח מ-550 שבידך.

Using dir="ltr" (wrong):
Pepper Creek LLC, שמות זה-עשתה, מזון ירח מ-550 שבידך.

Text Flow

Vertical Text

- Use the writing-mode property (CSS3).
- For example, to display top-to-bottom, and right-to-left text use:

```
div.vertical { writing-mode: tb-rl }
```

⇒ Example in [HTML](#), and in [SVG](#).

Text Flow – Vertical, HTML

```
<p style="writing-mode: tb-rl"> Example of vertical text (tb-rl).</p>
<p style="writing-mode: tb-rl"> Example of vertical text with 
  <span style="writing-mode: lr-tb"> horizontal</span> insert.</p>
```

Text Flow – Vertical, HTML Output

Example of horizontal text (rl-tb).
Text Flow – Vertical, SVG

```xml
<?xml version="1.0" >
<svg width="330" height="330">
  <g style="font-size:24;">  
    <text x="20" y="26" style="writing-mode: lr;">Horizontal Text</text>
    <text x="20" y="56" style="writing-mode: tb;">Example of vertical text</text>
  </g>
</svg>
```

Text Flow – Vertical, SVG in HTML

```html
<html>
<body>
<p>
<object data="Vertical.svg" type="image/svg+xml" width="330" height="330" />
</p>
</body>
</html>
```

Text Flow – Vertical, SVG Output

Ruby Annotation

Annotation in smaller characters running above or below a base text.

- Used in Japanese for pronunciation of Kanji characters (Furigana).
- W3C Ruby Module: `<ruby>` element with `<rb>` for the base text, `<rt>` for the ruby text, `<rbc>` and `<rtc>` for complex annotations.

Example: Ruby.htm
Simple Ruby test:

```html
<p>Ruby complex:</p>
<p>Ruby with parenthesis text, used if ruby is not implemented:</p>
</ruby>
</p>
<p>Ruby complex:</p>
<p>Ruby complex:</p>
<p>Ruby complex:</p>
<p>Ruby complex:
  Month  Day  Year  Expiration Date
  10 31 2002  Month  Day  Year  Expiration Date</p>
Sorting

XSL offers the `<xsl:sort/>` element to collate lists of items.

- Use `lang` (not `xml:lang`) to specify the language to use for the sorting rules.
- Results depend on the implementation of the XSL engine.

➤ Example: `Sorting.xml` sorted for English and Norwegian. (`Sorting_EN.xsl` and `Sorting_NO.xsl`).

Sorting

Version 2.0 of XSL has new features for `<xsl:sort>`

http://www.w3.org/TR/xslt20/#dt-collation

- **case-order attribute** specifies whether to sort uppercase or lowercase first.
- **collation attribute** names an implementer-defined collation to use.
  - if given, `lang` and `case-order` are ignored.

Summary Standards Text Layout

<table>
<thead>
<tr>
<th>Feature</th>
<th>Y</th>
<th>N</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lang()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lang pseudo-class</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Lang attr selector</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Quot:qo</td>
<td>N</td>
<td>Y</td>
<td>½</td>
</tr>
<tr>
<td>Text-transform</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Css list-style-type</td>
<td>N</td>
<td>Y</td>
<td>½</td>
</tr>
<tr>
<td>Xsl number</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Xsl format-number</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Html bi-directional text</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Css bi-directional text</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Vertical text (SVG losing ground)</td>
<td>Y/N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Ruby annotation</td>
<td>½</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Css3 combined sort</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Xsl:sort</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Web Internationalization Agenda

- Part 1 – Character Processing
- Coffee Break
- Part 2
  - Layout and Typography
  - Designing International Web Sites
Requirements

- Requirements are not always compatible:
  - **Business requirements.**
    - Web site ranked high in search engines;
    - Single look and feel across sites in different languages
  - **Localization requirement.**
    - Avoid changing links in localized pages;
    - To have locale-specific content; etc.

- Solutions depend on technologies used
  - (static Web site, client-side scripting, server-side scripting, databases, multiple addresses, etc).

Domain Names

Easier if each language has its own domain name: www.xyzcorp.fr, www.xyzcorp.de, etc.
→ One domain = One language.

Unfortunately:
- Most site have only one address for many languages.
- Even ‘country-specific’ sites may have several languages: www.xyzcorp.ca
  → English, French, Inuktitut.

Directories and Files

One possible solution:
- Home page of the ‘main’ language is the entry point of the directory structure.
  (e.g. index.html)
- Language home pages are also at the root and have a language identifier in their name.
  (e.g. index_fr.html)
- Other pages have identical names across languages, but are in different language directories.
Directories and Files

- Allow search engines to retrieve meaningful information (but emphasis for the main language).
- Maximize the use of relative URLs (no link change, except to the home page).
  If scripting is available, you can have the links resolved at run-time.
- Allows room for locale-specific content if necessary.

Directories and Files

- Use cookies if you want to remember the preferred language of the user and redirect him/her to the relevant set of files.
- Use common directory for shared files.
- Use meaningful directory and file names.
  - Avoid translating directory and file names.
    – However, this hurts SEO.
- Treat the source language just like another language as much as possible.

Language Selection

- List box of language names in native language
  – Make sure characters display correctly (fonts)
  – Graphics are always displayed correctly.
- Destination Choice
  – The same page in the new language.
  – The main page in the new language.
  (for country-specific sites, etc.)

Good Practices – IDs

IDs are VERY useful for re-use of translation, and for re-use of text across documents.

- in HTML IDs can be set for all elements containing text, except the <title> element.
- Make sure to provide an ID attribute for the translatable elements of your XML vocabularies, so it can be utilized for re-use, leveraging, etc.
Good Practices – Attributes

When creating new XML vocabularies: Avoid using attributes for storing translatable text.

- Impossible to add needed bidi tags in an attribute.
- Cause segmentation issues in many tools.
- Much more difficult to have metadata for attributes than for elements.
- You cannot set different languages for two attributes in the same element.
- More tricky to set unique IDs for attributes.

Good Practices – Embedded Data

Data that are not text content (e.g. scripts, SQL queries, etc.).

- Keep them outside of the document if possible (e.g. using include mechanisms).
- At least, make sure elements with such data are identified for the localizer (who might need to apply a process different than for the rest of the document content).
- Internationalize your scripts/queries/etc.

Good Practices – Use Style-sheets

• Separate the function of a term (a title, a link, an important term) from its display (bolded, underlined, in 12-points Courier, etc.)
  - Type of display for the target language(s) may be different than for the source language.
  - Force author/developer to think about the structure of the document.
• Avoid `<br />` -like elements when possible: Use styles to format, not tags.

Good Practices – CDATA Sections

Avoid CDATA sections if possible.

- Translation tools do not handle CDATA well.
- Keeping track on inline CDATA leads to meaningless inline codes in segments (and can affect leveraging).
- NCRs are not allowed in CDATA. This may cause problems if the document is converted to an encoding where some characters need to be written as NCRs.
  By the way: CDATA does NOT preserve spaces.
**Conclusions**

1. Web technologies are among the best ways to store, manipulate and represent data in different languages.

2. Implementation of Web standards is incomplete and inconsistent

---

**Acknowledgements**

- Parts of this presentation are based on “Weaving the multilingual Web: Standards and their implementations” by François Yergeau and Martin Dürst, given at previous Unicode conferences.
- Yves Savourel (ENLASO Corporation) created the test programs and the initial versions of the best practices content.
- Thanks to Richard Ishida and Martin Dürst for their extensive review.

---

**Additional Resources**

- W3C Internationalization Work Group http://www.w3.org/International
- Unicode in XML and other Markup Languages http://www.w3.org/TR/unicode-xml
- Character Model for the World Wide Web http://www.w3.org/TR/charmod
- Richard Ishida’s paper on “Localisation Considerations in DTD Design” http://www.w3.org/People/Ishida/writing.html#dtd
- XML Internationalization FAQ http://www.opentag.com/xmli18nfaq.htm