An Introduction to Unicode

Henri Sivonen
What’s Unicode?

- 21-bit coded character set
- Includes property data, rules and algorithms
- Aims to cover all human writing systems currently in use
- Also covers some obsolete systems for scholarly use
ISO-10646

• A standard list of characters that is the same as the Unicode list of characters
• Looks more official as a reference
• The Unicode Standard is more than the list
• Just refer to Unicode
• Specs that are available on the Web win
Why Unicode?

• Multiple encodings are trouble
• Legacy repertoires often too narrow
• Mutually exclusive repertoires are bad
• Why should the user have to pick either German or Russian support?
• Display layer late binding prevents smart processing based on character semantics
Resistance is Futile

- Immense momentum towards Unicode
- XML, HTML 4…
- Java, C#, Python, Perl 5.8, JavaScript…
- Mac OS X, Windows 2000, Gnome 2…
- Apple, Microsoft, IBM, Sun, Gnome Foundation, W3C, IETF all pulling to the same direction!
You Will be Assimilated

• Better to conform now than to fight and conform later

• Your boss wants XML; XML wants Unicode

• Need €? ISO-8859-15 is just fire fighting!
Free Your Mind

- People have a lot of prior assumptions that are not true of Unicode
- Some of them were true with more primitive text encodings and fonts
- It helps not to assume these things
- For example, there’s no single “Unicode encoding” for interchange
Misconceptions

• Unicode character = 16 bits
• Character = glyph
• Code point = glyph index
• Selection unit = glyph
• Key press = character
• Caret moves character by character
More Misconceptions

• I am European / American / Japanese. I don’t need to know about Unicode.

• Displaying Chinese is the hardest problem

• Once you’ve tackled CJK, you’re done

• Unicode is just “wide ISO-8859-1”—the same way ISO-8859-1 is “wide ASCII”

• Klingon is in Unicode
Glyph

- An atomic shape in a font
- Different glyphs: a a A A A
- One glyph: ä fi
- Two glyphs: å fi
- Glyph sharing between Latin, Greek and Cyrillic possible (leads to Latin dominance)
Grapheme

- Fuzzy concept
- A graphical unit as perceived by a user
- May consist of multiple glyphs
- Eg. base character plus diacritics
Abstract Character

- A is A regardless of font

- Α Α Α Α Α Α Α Α Α Α Α Α Α Α Α Α Α Α Α

- Greek A and Cyrillic A are distinct

- Upper and lower case are distinct
Control Characters

- Ambiguous controls from ASCII
  - Line feed, carriage return, etc.
- New ones
  - Ligature modifiers, less ambiguous paragraph separators, etc.
Combining Characters

- How many characters: ä?
  - One: LATIN SMALL LETTER A WITH DIAERESIS
  - Two: LATIN SMALL LETTER A + COMBINING DIAERESIS
- Precomposed vs. decomposed
  - Canonical equivalence
  - Normalization forms
Presentation Forms

- \textit{fi}: LATIN SMALL LETTER F + LATIN SMALL LETTER I
- \textit{fi}: LATIN SMALL LIGATURE FI
- Presentation forms as characters for compatibility with legacy encodings
  - Compatibility equivalence
  - Normalization forms
## Normalization Forms

<table>
<thead>
<tr>
<th>Compatibility</th>
<th>Precomposed</th>
<th>Decomposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>chars intact</td>
<td>NFC</td>
<td>NFD</td>
</tr>
<tr>
<td>decomposition</td>
<td>NFKC</td>
<td>NFKD</td>
</tr>
</tbody>
</table>
## Normalization Forms

<table>
<thead>
<tr>
<th>NFC</th>
<th>NFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFKC</td>
<td>NFKD</td>
</tr>
</tbody>
</table>

Diagram illustrating the relationships between different normalization forms.
Unicode as “Wide ASCII”

- Requires precomposed form
- Workable with
  - Latin, Greek, Cyrillic, Armenian, Georgian
  - Chinese, Japanese, modern Korean
  - Ogham, Runic, …
What’s Latin?

• Not just A–Z with a mix of diacritics
• IPA
• IPA-based characters in African writing
  • The poorest people have the strangest characters
• Font availability problems
Latin Complications

- Sorting with local conventions
- Searching
  - Case-insensitive?
  - Diacritic-insensitive?
- Turkish i
Sorting

• How to sort ä?
  • Finnish, Swedish: letter on own right; sort after z and å
  • English, French: a with diacritic; sort after a
  • German phonebook: alternative of ae; sort between ae and af
Case Mapping

- German ß
- Turkish i
- Croatian digraphs
- Greek also: Final sigma
Diacritic Appearance

- Caron and cedilla may look different
- Naïve combinations in Gill Sans: ę ķ ė ě
- Helvetica: ģ ķ ď ĭ
- Some fonts have alternative glyphs
- Core fonts biased towards bigger markets
Han Unification

- CJK ideographs share a Chinese origin
- If encoded thrice, even common ideographs wouldn’t fit in the BMP
- An ideograph that appears across CJK is considered one character (unified)
- Controversial: Imposed by Westeners
GB18030

- Instead of endorsing Unicode, China made a new standard on its own…
- …And outlawed the sale of non-conforming software products!
- The sane conformance strategy: Unicode internally, Unicode extended to cover GB18030, converters for IO, huge font (even if ugly) provided with the OS
Beyond “Wide ASCII”

- One-to-one character to glyph mapping and left to right glyph placement on the baseline not enough for all writing systems
- Right to left, ligatures, positional forms, combining marks, reordering…
Different Cultural Expectations

- Latin
- History of adapting writing to technology
- Dumbed-down typography tolerated
- Arabic
  - Calligraphic appearance retained in print
  - Contextual shaping expected up front
Progressive Latin Features

ALL UPPER CASE MONOSPACE
English with lower case
Éuröpéän çhäräçtérs
Variable-width glyphs
“Quotes”—even dashes
Type with kerning pairs
Specific automatic ligatures
Aðbritràry diacritics

Full support for arbitrary shaping
Bidi

- Bidirectional layout needed for Hebrew, Arabic, etc.
- Characters stored and typed in logical order
- Characters have inherent directionality: LTR (eg. a), RTL (eg. א) or neutral (eg. ?)
- Need to know dominant direction
Positional Forms

- Required for Arabic
- Abstract character stored – glyph varies
- Isolated ف, final ف, medial ف, initial ف
- Can be used as an effect with Latin: aa
# Grapheme Boundaries

<table>
<thead>
<tr>
<th></th>
<th>उदाहरण</th>
<th>यूनिकोड क्या है</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Caret stops</strong></td>
<td>🉁Example�</td>
<td>🉁Example�</td>
</tr>
<tr>
<td><strong>Backspaces</strong></td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td><strong>Characters</strong></td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Hangul

- Alphabet–syllabary duality
- A syllable block (한) consists of alphabetic letters called jamo (ㄱ ㅏ ㄴ)
- When treated as an alphabet, layout software needs to group letters as blocks
- Precomposed syllable characters for modern Korean only
Fonts

• Type 1 format inadequate
• TrueType more extensible
  • Extended TrueType (.ttf)
  • OpenType (.otf)
• Apple Advanced Typography (.dfont)
Extended TrueType

• Like old TrueType but with a larger repertoire and Unicode mapping

• May contain additional tables for OpenType “smart font” features
OpenType

- Extended TrueType with Type 1 geometry
- Provides a migration path for foundries with a heavy investment in Type 1 fonts
- Backed by Adobe and Microsoft
Apple Advanced Typography

- Resurrected GX
- More advanced shaping than in OpenType
- Features overlap with OpenType
- Only supported by Apple
- Advanced features not supported by Adobe’s cross-platform font engine
Printing

- PostScript and PDF have an old-style notion of a font
- A font is basically an array of hinted glyphs (with advances)
- Need to build magic into a printing library that lets apps use new-style fonts and complex text layout
Printing, continued

• Auto-generate embedded fonts with up to 256 glyphs in each
• Type 1 or 42 depending on glyph data
• Position glyphs individually
• Recovering intelligible text gets ugly
• PDF *may* contain reverse mappings
Unicode Encoding Forms and Schemes

• More than one way to store sequences of code points

• Unicode Encoding Form: Representation as in-memory code units (32, 16 or 8 bits)

• Unicode Encoding Scheme: Representation as bytes for interchange

• Encoding Form + byte order
UTF-32

- 32-bit code units
- One code unit per code point
- Straight-forward
- Wastes space
- Byte order issues with serialization
- Don’t use for interchange
UTF-16

- 16-bit code units
- Extension to the original UCS-2 encoding
- BMP characters take one code unit
- Astral characters take two code units (surrogate pair)
Surrogates

- Chars above U+FFFF don’t fit in 16 bits
- Represented in UTF-16 as a surrogate pair consisting of two 16-bit code units

21-bit scalar

\[
\begin{array}{c}
110110wwwwwwxxxxxxx \\
110111xxxxxxxxxxx
\end{array}
\]

High surrogate

Low surrogate

Where \( wwww = uuuuu - 1 \)
Byte Order Mark (BOM)

- U+FEFF written at the start of a data stream
- U+FFFE guaranteed to be unassigned
- If a UTF-16 data stream starts with 0xFFFFE, swap bytes
- Also considered an encoding signature or magic number for UTF-16
UTF-8 – One Encoding to Rule Them All

- 8-bit code units
- A character is encoded as 1…4 bytes
- Invented by Ken Thompson (Yes, *that* Ken Thompson)
- “Is UTF-8 a racist kludge or a stroke of genius?” – Tim Bray
UTF-8 Byte Sequences

0xxxxxxx

110xxxxx  10xxxxxx

1110xxxx  10xxxxxx  10xxxxxx

11110xxx  10xxxxxx  10xxxxxx  10xxxxxx
Racist Kludge?

- Compared to UTF-16…
  - English text shrinks by 50%
  - Asian text expands by 50%
- The status of ASCII is a historical reality
- Not a real technical problem: Use gzip!
- One ideograph vs. many alphabetic letters
Stroke of Genius?

- ASCII is ASCII (one byte per character)
- Including control characters!
- Other characters don’t overlap with ASCII
- No byte order issues
- Byte-wise lex sort = code point lex sort
- Implemented using bitwise operations – no multiplication, division or look-up tables
Benefits of ASCII
Identity of UTF-8

- \0 termination
- Unix file system compatibility
- Retrofitting text terminals with Unicode
- Works over SMTP without Base64
- Byte-oriented parsing of grammars where non-ASCII occurs only in string literals
UTF-8 Disadvantages

- No O(1) random access by character index
- Not such a big deal
- Doesn’t work with UTF-16, either, in the presence on astral characters
- Harder to look inside a string than with UTF-16
- Space requirement for Asian text
Other Unicode Encoding Schemes

- UTF-7
  - RFC 2152; obsolete email encoding
- CESU-8
  - Formalization of broken UTF-8
- Punycode
  - RFC 3492; only for IDNs
Compressed Representations

- SCSU
  - Not deterministic
- BOCU-1
  - MIME text/* compatible
  - Byte-wise lex sort = code point lex sort
  - Deterministic
Dealing with Encodings

- Unicode is designed to be round-trip compatible with legacy encodings
- Legacy encodings can easily be converted to Unicode
Encodings on Input

- Convert input into your internal Unicode encoding form at the first opportunity
- When dealing with XML, let the XML processor do this for you
Encodings on Output: XML

- XML processors are required to support two encodings: UTF-8 and UTF-16

- Using any other encoding takes more work and is unsafe

- Use explicit XML declaration with UTF-8

- Use xml:lang for CJK disambiguation

- Don’t use text/xml; use application/xml
Encodings on Output: HTML

- Use UTF-8
- The only serious browser in recent memory that does not support UTF-8 is Opera 5
- Even Netscape 4 and Lynx support UTF-8
Encodings on Output: text/plain Mail

- The lazy way: Use UTF-8
  - Tell pine users to install the iconv patch
- The compatible way: Adaptive encoding
  - Try ASCII, ISO-8859-1, Windows-1252…
  - UTF-8 as last resort
- Always declare the encoding properly
Normalization and IO

• Unless otherwise required by protocol, use NFC for output

• To be safe, normalize input data to your required form yourself
• \0-terminated UTF-8 strings
• Preferred by Gnome libraries
• Smuggling Unicode through legacy code
• 0x0000-terminated UTF-16 strings
• Preferred by APIs from Apple, Microsoft and IBM
UTF-16 in C

- wchar_t not portable
- Can be 1, 2 (MS) or 4 (GNU) bytes wide
- Everyone has a typedef for UTF-16
  - UniChar, UChar, gunichar2, PRUnichar, …
String Tools for C

- ICU from IBM
- glib
- CoreFoundation
C APIs for Imaging

- ATSUI (Mac OS X)
- Pango aka. Παν (Gnome)
- Uniscribe (Windows)
C APIs for Imaging, continued

- Handle hit testing / selection / caret movement on behalf of the app
- At their best when driven with paragraph-sized chunks
- Problematic with apps that expect to do almost everything themselves
C++

• No universally accepted Unicode string class library (as usual with C++…)

• C-style UTF-8 or UTF-16 strings needed as common ground between libraries
Java

- Originally assumes character = 16 bits ("Wide ASCII" mindset in API design)
- Treat Strings and char[]s as UTF-16
- Normalization and other cool tools available in ICU4J by IBM
- Never trust the platform default encoding! Know what encoding you are using for IO!
C#

- Strings are indexed by UTF-16 code units as in Java
JavaScript

- Strings are indexed by UTF-16 code units as in Java
Objective-C (on OS X)

- NSStrings are indexed by UTF-16 code units as in Java
- NSString provides methods for normalization
- Comparison considers canonical equivalence
Python

- Byte strings and Unicode strings since Python 2.0

- UTF-16 or UTF-32 depending on how the interpreter was compiled! (Cf. PEP 261)
  - UTF-16: Jython, Apple
  - UTF-32: Debian
Perl

- Byte and Unicode strings since Perl 5.6
- Avoid versions earlier than Perl 5.8
- Strings are indexed by UTF-32 code units
- Normalization in Unicode::Normalize
- Character class & name data
AppleScript

- Legacy MacRoman strings (string)
- UTF-16 strings (Unicode text)
  - Badly documented and supported
  - Script Editor can’t display astral chars
- “International Text” means locale-specific legacy Mac encodings
PHP4

• No notion of a Unicode string
  • Strings are byte strings (can hold UTF-8)
• No supporting library functions by default, either
  • Optional iconv and mb_ functions
Don’t Trust the Documentation

- “Unicode character” in API docs often means a UTF-16 code unit
- Even when docs say “UCS-2”, UTF-16 may be supported
- When docs say “UTF-8”, the implementation may use CESU-8
- Always test with astral chars yourself!
References

- http://www.unicode.org/versions/Unicode4.0.1/
- http://www.unicode.org/reports/tr15/
- http://www.omniglot.com/
- http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF
- http://www.pango.org/
- man perlunicode
- man perluniintro