Internationalization: A Guide for the Perplexed

by
Peter Saint-André
for
RFC Editor Team, November 2013
Topic #1: What's the Problem?
in the beginning was ASCII
and ASCII was with the Internet
and ASCII was [a false] god :)
problem: ASCII is extremely limited
there are thousands of languages and scripts
can't we force everyone to use ASCII?
sorry, but that's incredibly naïve
we need to encode more than [A-Z][a-z]
i.e., we need internationalization
specifically, we need Unicode...
[unicode.org]
a set of every* character humans care about

[* sorry, no Klingon, Elvish, or Dwarvish *]
technically, Unicode is a “coded character set” (RFC 6365)
i.e., each character has an alphanumeric code assigned to it
in Unicode, these codes are hexadecimal (i.e., base 16 instead of base 10)
we use the convention

U+xxxx

[ RFC 5137 ]
e.g., (ASCII) SPACE is the 32nd code point, i.e., U+0020
similarly, SOLIDUS “/” is the 47th code point, i.e., U+002F
\[ P = U+0050 \]
p = U+0070
\[ \pi = \text{U+03C0} \]
平 = U+5E73
and so on up to $\infty$
(well, up to 1,114,111 :)
each character also has various properties
letter, number, symbol, punctuation, etc.
case: UPPER vs. lower vs. Title
status as a modifier
  (e.g., accent mark)
width: full, half, narrow, even zero!
left-to-right vs. right-to-left
etc.
each character looks* and behaves differently

[ * mostly – we’ll talk about confusable characters... ]
we handle characters based on properties
but there be dragons!
case folding
character equivalence
decomposition and recomposition
normalization
various encodings
string comparison
mappings (e.g., based on user locale)
right-to-left vs. left-to-right scripts and characters
confusable characters
enforcement in protocols
(or documentation :)

Saturday, November 2, 13
registration policies
versioning
user interface issues
reliance on rendering engines
and more!
plus, many rules have exceptions!
lots of messy complexity
are you scared yet?
if not,
you will be soon :)

Topic #2: Case
A \rightarrow a

[U+0041] \rightarrow [U+0061]

(note: symbols like \rightarrow are only my personal convention)
I $\rightarrow$ i

[U+0049] $\rightarrow$ [U+0069]
and so on, right?
not so fast!
only a few scripts have the notion of "case"
some characters don't map cleanly
e.g., German esszett, which is only lowercase
$\beta \equiv SS \downarrow ss$

$\mathrm{[U+00DF]} \Rightarrow [\mathrm{U+0053}][\mathrm{U+0053}] \downarrow [\mathrm{U+0073}][\mathrm{U+0073}]$
	herefore

FUSSBALL $\downarrow$ fussball, not fußball
e.g., Greek final sigma
\[ \zeta \not\Rightarrow \Sigma \not\Rightarrow \sigma \]

[u+03C2] \not\Rightarrow [u+03A3] \not\Rightarrow [u+03C3]

therefore:

ΠΑΙΔΟΣ \not\Rightarrow παιδοσ, not παιδος
even worse, locale & context matter...
e.g., "Turkish dotless i"
I ↔ i & ì → ì

[U+0049] ↓ [U+0131] & [U+0069] → [U+0130]
these differences can have consequences
thankfully, not *that* many exceptions
[ approximately 2,000 ]
Topic #3: Character Equivalence
one character can be equivalent to another
Å Å

[U+212B] = [U+00C5]
(angstrom sign) = ("a" with ring above)
one character can be equivalent to a sequence of characters
Å ≡ A + °

[U+00C5] = [U+0041] + [U+030A]
$\zeta \equiv C + \,,$

$[\text{U+00E7}] = [\text{U+0063}] + [\text{U+0327}]$
\[ \tilde{\mathcal{R}} \equiv \mathcal{R} + \gamma \]

\[ [U+0158] = [U+0052] + [U+030C] \]
Ř is a "composite character"
R + ¯ is a combining sequence
composite characters are your friends :-)
two kinds of equivalence...
(a) Canonical Equivalence
κανών = rule, standard, measure
"this character is the standard for that one"
characters look and behave the same
Å Å

[U+212B] = [U+00C5]

(angstrom sign) = ("a" with ring above)
Å ≡ A + °

[U+00C5] = [U+0041] + [U+030A]
(2) Compatibility

Equivalence
compati = "suffer with"
"this character suffers with that one"
a.k.a. we suffer with compatibility equivalence :-)
often for the sake of backward compatibility
characters might look and behave differently
$\ IV \ \approx \ I \ + \ V$

$[U+2163] \ \approx \ [U+0049] \ + \ [U+0056]$

(roman \ numeral \ four) \ \approx \ (uppercase \ "i") \ + \ (uppercase \ "v")$
\[ \text{fi} \approx f + i \]

[U+FB01] ≈ [U+0066] + [U+0069]

(ligature “fi”) ≈ (lowercase “f”) (lowercase “i”)
\[ f \approx s \]

[U+017F] ≈ [U+0073]

("long s") ≈ (lowercase "s")
canonical vs. compatible is a key to Unicode!
many forms of compatibility...
Compatibility

- “standard”, denoted by `<compat>`
- `<sub>`, e.g., $F_2$ (U+2082)
- `<super>`, e.g., $2^4$ (U+2072)
- `<circle>`, e.g., orElse (U+2467)
- `<fraction>`, e.g., $\frac{3}{4}$ (U+00BE)
- and more!
Topic #4: Decomposition
two kinds of decomposition...
canonical decomposition
compatibility decomposition
decomposition can take more than one step...
\[ \hat{A} \equiv \breve{A} \equiv A + ^\circ \]

\[ [U+212B] = [U+00C5] = [U+0041] + [U+030A] \]
in decomposition, order matters!
\[\overset{\sim}{\omega} \equiv \overset{\sim}{\omega} + \overset{\sim}{\nu}\]

\([U+1FA7] = [U+1F67] + [U+0345]\)
\[ \bar{\omega} \equiv \bar{\omega} + \sim \]

\[ \text{[U+1F67]} = [\text{U+1F61}] + [\text{U+0342}] \]
\[ \omega \equiv \omega + \prime \]

\[ \text{[U+1F61]} = \text{[U+03C9]} + \text{[U+0314]} \]
\[ ω ≡ \cdot + ' + \sim + . \]

\[ [U+1FA7] ≡ [U+03C9] + [U+0314] + [U+0342] + [U+0345] \]
full decomposition can have both canonical and compatibility steps...
how “aggressive” do we want to be?
\[ \int \equiv \int + \cdot \]

\[
[U+1E9B] = [U+017F] + [U+0307]
\]

(this is canonical equivalence)
\[ f \approx s \]

[U+017F] \approx [U+0073]

(this is compatibility equivalence)
\[ \vec{F} \approx s + v \cdot \]

\([U+2163] \approx [U+0073] + [U+0307]\)

(full decomposition leads to a strange result)
some characters don’t decompose as we might expect...
æ \neq \text{ae} \\
[U+00E6] \neq [U+0061][U+0065] \\
( is this purely an æsthetic issue? ;-)
Topic #5: (Re-)Composition
after a character is decomposed, we can put it back together
recomposition returns a
composite character
(well, usually)
output depends on which decomposition we used
i.e., canonical, compatibility, or both?
how “aggressive” were we in decomposition?
$\text{IV} \Rightarrow \text{IV}$

$\text{[U+2163]} \Rightarrow \text{[U+0049] + [U+0056]}$

(roman numeral four) $\Rightarrow$ (uppercase “i”) + (uppercase “v”)

e.g., is Henry the Fourth the same as Henry Eye Vee?
\[ \text{fi} \Rightarrow f + i \]

\[ \text{[U+FB01]} \Rightarrow \text{[U+0066]} + \text{[U+0069]} \]

(ligature “fi”) \Rightarrow (lowercase “f”) (lowercase “i”)
\[ f \Rightarrow \dot{s} \]

\[ [U+1E9B] \Rightarrow [U+1E61] \]
\[ \frac{3}{4} \implies 3/4 \]

\[ \text{[U+00BE]} \implies \text{[U+0033] [U+2044] [U+0034]} \]

(note: \text{U+2044} is “fraction slash”, not \text{solidus}!)
8 => 8

[U+2467] => [U+0038]
Topic #6: Normalization Forms
process for determining equivalence
there are 4 forms of normalization
Normalization Forms
D, C, KD, and KC
a.k.a. NFD, NFC, NFKD, NFKC
2 perform only decomposition (NFD and NFKD)
2 perform decomposition and recomposition (NFC and NFKC)
Normalization Forms

• NFD = canonical decomposition

• NFKD = canonical and compatibility decomposition ("K" is for compatibility!)

• NFC = canonical decomposition, then recomposition

• NFKC = canonical and compatibility decomposition, then recomposition
NFD

- Applies canonical equivalence rules only
- Performs decomposition only
- Does not return a composite character (usually)
- Can result in faster processing (no compatibility, no recomposition)
- The simplest of the normalization forms
NFKD

- Applies canonical equivalence rules and compatibility equivalence rules
- Performs decomposition only
- Does not return a composite character
- Can result in slower processing than NFD (compatibility, but still no recomposition)
- More Clever™ than NFD
# NFD vs. NFKD

<table>
<thead>
<tr>
<th>input</th>
<th>NFD</th>
<th>NFKD</th>
</tr>
</thead>
<tbody>
<tr>
<td>fi</td>
<td>fi</td>
<td>f + i</td>
</tr>
<tr>
<td>ñ̟</td>
<td>ñ̟</td>
<td>s + ̅</td>
</tr>
<tr>
<td>Ⅳ</td>
<td>Ⅳ</td>
<td>1 + ⅵ</td>
</tr>
<tr>
<td>¾</td>
<td>¾</td>
<td>3 + / + 4</td>
</tr>
<tr>
<td>８</td>
<td>８</td>
<td>8</td>
</tr>
<tr>
<td>２⁵</td>
<td>２ + ⁵</td>
<td>2 + 5</td>
</tr>
<tr>
<td>ω̱</td>
<td>ω + ́ + ́ + ́</td>
<td>ω + ́ + ́ + ́</td>
</tr>
</tbody>
</table>
NFC

• Applies canonical equivalence rules only (first decomposition, then recomposition)

• Compared to NFKC:
  • Produces more matches during comparison operations
  • Requires less time and processing
  • Less Clever™ (but smarter than NFD)
NFKC

- Applies canonical equivalence rules and then compatibility equivalence rules (first decomposition, then recomposition)

- Compared to NFC:
  - Produces more false negatives
  - Requires more time and processing
  - It's Really Clever™
## NFC vs. NFKC

<table>
<thead>
<tr>
<th>input</th>
<th>NFC</th>
<th>NFKC</th>
</tr>
</thead>
<tbody>
<tr>
<td>fi</td>
<td>fi</td>
<td>fi</td>
</tr>
<tr>
<td>ĕ</td>
<td>ĕ</td>
<td>š</td>
</tr>
<tr>
<td>IV</td>
<td>IV</td>
<td>IV</td>
</tr>
<tr>
<td>3/4</td>
<td>3/4</td>
<td>3 / 4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>(2^5)</td>
<td>(2^5)</td>
<td>2 5</td>
</tr>
<tr>
<td>ω</td>
<td>ω</td>
<td>ω</td>
</tr>
</tbody>
</table>
which normalization form is best?
it depends on what you want to accomplish :)
NFC is generally recommended

[ RFC 5198 ]
think long and hard about using something other than NFC
Topic #7: Encoding
Unicode is not an Internet technology
Unicode is not even a computing technology
nothing we’ve talked about yet has anything to do with computers
a code point just identifies a character
a character could be written, spoken, etc.
computers need characters to be encoded as bits and bytes
e.g., ASCII was originally a 7-bit system \((2^7\text{ gives us 128 code points})\)
8-bit ASCII gives us
$2^8 = 256$ code points
Unicode has many more code points, we need fancier encodings
UTF-8,
UTF-16,
UTF-32
UTF-8 (RFC 3629) is the IETF-preferred encoding (RFC 2277 / RFC 5198)
each character is encoded using 1-4 8-bit “octets”
for the ASCII range, UTF-8 preserves the old 7-bit assignments
e.g., $P =$

ASCII decimal code 80
(i.e., UTF-8 hex code 50)
for characters above
decimal code 128,
we need 2+ 8-bit “octets”
most modern characters are encoded with two or three octets (up to U+FFFF)
a.k.a. the
“Basic Multilingual Plane” (BMP)
higher planes are available (a.k.a. the “astral planes”)

e.g., \(\underline{\text{#}} = \text{U+10133}\)

[i.e., AEGEAN NUMBER NINETY THOUSAND]
however, these are unlikely to be used on the Internet
although UTF-8 is very common, there are exceptions...
especially the internal data representation in Java, JavaScript, and Windows
[note: some systems insert a “byte order mark” (BOM) at the start of UTF-8 data]
think long and hard before using something other than UTF-8
Topic #8:
String Comparison
some strings are special
e.g., addresses and other identifiers
many reasons to compare strings...
authentication
authorization
registration
data storage
and many other operations
first attempt:
stringprep (~2002)
designed for domain names ("IDNA")
applied to many other identifier types
addresses, usernames, passwords, file paths, nicknames, etc....
each has different uses, needs, and structure
Stringprep Basics (1)

- Choose a Unicode version (oops, 3.2 only!)
- Choose a normalization form (NFKC?!)  
- Specify how to handle whitespace
- Specify whether to use case folding
- Specify bidirectional handling
- Specify prohibited characters
Stringprep Basics (2)

• Specify handling via comprehensive tables that capture:
  • Mappings (e.g., whitespace, case folding)
  • Prohibited characters (e.g., controls, spaces, symbols)
  • Bidirectionality
Why Not Stringprep?

- Version agility is important (latest = 6.3)
- NFKC can lead to unintuitive results, as we’ve seen (e.g., ſ ≈ s)
- Accepting registration of all characters and scripts can cause problems (e.g., phishing)
- Big tables are hard to maintain and update
- See RFC 4690 for details
• No more stringprep

• Decisions based on properties of Unicode characters

• Algorithms, not huge tables

• Version agility
IDNA2008 (2)

- Four "buckets" based on properties:
  - PROTOCOL-VALID
  - CONTEXT RULE REQUIRED
  - DISALLOWED
  - UNASSIGNED
• Basically, PVALID = "letter-digit-hyphen"

• The "inclusion approach" of IDNA2008 works because domain names have always traditionally been "letter-digit-hyphen", not just any random symbols, punctuation, etc.

• Domain names are mnemonics, not random strings of characters
Challenges

• The dividing line between user interface and protocol has moved substantially
• Applications need to take more responsibility
• Can't just hand things off to stringprep and expect good things to happen
• Mappings are out of scope for IDNA2008
Stringprep Customers

- IDNA was the main stringprep “customer”
- Other customers: LDAP, SASL, iSCSI, XMPP, etc.
- A new approach developed in the PRECIS WG ("Preparation and Comparison of Internationalized Strings")
PREMIS

• Follow "inclusion approach" like IDNA
• Define two “string classes” (IdentifierClass, FreeformClass)
• Enable "profiling" for particular protocols (case mapping, normalization, etc.)
• draft-ietf-precis-framework etc.
Topic #9: Confusable Characters
many characters look alike ("confusables")...
A \neq A
[U+0041] \neq [U+0410]
4 \neq 4

[\text{U+13CE}] \neq [\text{U+0034}]
$\text{STPETER} \neq \text{STPETER}$

$\text{[U+13DA]} [\text{U+13A2]} [\text{U+13B5]} [\text{U+13CB]} [\text{U+13A2]} [\text{U+13CB]} [\text{U+13D2]}$

$\neq$

$\text{[U+0053]} [\text{U+0054]} [\text{U+0050]} [\text{U+0045]} [\text{U+0054]} [\text{U+0045]} [\text{U+0052}.$
these are *not* equivalents
humans usually can't distinguish
4 vs. 4 looks like a font difference
no foolproof solutions for confusables
prohibiting mixed scripts can help
see RFC 4690/5890 and draft-ietf-precis-framework
Topic #10: Rules and Responsibilities
Who's Your Registrar?

- In IDNA, domain registrars have policies
- E.g., Hungarian registrar likely won't accept characters from Korean code block
- Do providers of (say) email and IM services also need to define such policies?
Enforcement

• Who enforces the rules?
  • Server?
  • Client?
  • Any network endpoint?
• Needs to be clear for each protocol!
Topic #11: Versioning
Version Changes

• New Unicode versions can add new characters, deprecate old characters, etc.

• Character properties can change between Unicode versions (e.g., from number to letter), but this should be rare

• A character could change from PVALID to DISALLOWED (etc.)
Version Mismatches

• Possibility of problems with authentication, message delivery, etc.
• In practice, not a concern because most of the modern characters we need are mapped in a stable way
Topic #12: User Interface
Good UI is Hard

• Account for application type, string types, locale, scripts, culture, input methods, output methods, graphical capabilities, etc.

• Probably not much that protocol geeks can say in the matter :)

• We need input from UI experts
Rendering

• UTF-8 encoded Unicode characters are rendered in a UI by a rendering engine

• These have improved dramatically over time! (Fewer “renderings” via □)

• In general, support for Unicode is improving all the time
Topic #13: Directionality
LTR and RTL

• Most scripts are rendered left-to-right
• Some scripts are rendered right-to-left (e.g., Arabic and Hebrew)
• Each Unicode code point is LTR or RTL
• What if they’re mixed? Hard problem!
BiDi Policies

• If a string contains any LTR character, the entire string is left-to-right

• BiDi rule from RFC 5893

• Other rules are possible (but there be major dragons here!)
THE END
STRETER @ STRETER.im
References: Unicode

- Unicode 6.3 spec @ unicode.org
- UAX 15: Unicode Normalization Forms
- UTR 17: Unicode Character Encoding Model
- UTR 36: Unicode Security Considerations
References: General

- RFC 6365: Internationalization terminology
- RFC 2277: IETF policy on characters sets and languages
- RFC 3629: UTF-8
- RFC 5137: ASCII escaping for Unicode
- RFC 5198: Unicode format for networks
References: Stringprep & IDNA2003

- RFC 3454: Stringprep
- RFC 3490: IDNA2003
- RFC 3491: Nameprep
- RFC 3492: Punycode
- RFC 4690: IDN review by IAB
References: IDNA2008

- RFC 5890: Definitions
- RFC 5891: Protocol
- RFC 5892: Unicode and IDNA
- RFC 5893: Right-to-Left Scripts
- RFC 5894: Background
References: PRECIS

- http://datatracker.ietf.org/wg/precis/
- RFC 6885
- draft-ietf-precis-framework
- draft-ietf-precis-mappings
- PRECIS-related I-Ds on usernames, passwords, nicknames, JabberIDs, etc.
Useful Tools and Websites

• Unicode Checker (Mac OS X)
• unicode-table.com
• Wikipedia pages about Unicode, UTF-8, and related topics
Acknowledgements

• Martin Dürst
• Joe Hildebrand
• John Klensin
• PRECIS WG
• XMPP community