A Précis of PRECIS: next-generation internationalization considerations

Chairs’ lunch session
IETF 92 — Dallas, TX, USA

IAB Internationalization Program
Special thanks to Peter Saint-André for use of his slides
Plan for today

• As little background as I can get away with
• Why previous attempts didn’t work
• What PRECIS does & why
• How this can and should be used by other protocols
• Some special cases
• Discuss
Why might you care?

• Any time you have any protocol element that is exposed to humans, there is a usability issue.

• Humans, alas, use languages and writing.
Background

- RFC 2277 requires “internationalization considerations”
- Also suggests “just use UTF-8”
- The first was widely ignored. The second is wrong.
- We spell internationalization “i18n”
ASCII for everyone!

Seems at least naïve
So, Unicode

- Coded character set (RFC 6365)
- It aspires to have every character
  - That humans care about
  - That have been added
Versions!

• New characters get added, and then you have a new version of Unicode

• Characters don’t go away (“stability guarantees”)
  • Also means that changing one’s mind is hard
Not developed for comparison

• Heritage is print and display of characters

• More than one way to print? Not a problem!

• Still displays the same
Jargon

- Each coded character has a number (hex)
- We mostly use U+xxxx convention (RFC 5137)
- Each coded character has many properties (e.g. letter vs number vs symbol, script, directionality)
Abstract character

• The thing you as ordinary writing-system user might think of when you think of “character”

• Might have more than one representation
  • ŵ + e or é
  • Å or Å
Cases

- Easy, right? i/l, a/A, and so on
- Not so fast. Not all writing systems have case
- Case isn’t stable for everyone (even in one script)
  - e.g. Turkic writing folds i to İ, and l to ł
  - might have round-trip issues (ß to SS to ss, é to E to e)
Comparing in Unicode

- Two kinds of equivalence
  - Canonical
  - Compatibility ("Kompatibility")
Comparing (2)

- Canonical equivalence means they’re really at bottom the same character

- Compatibility equivalence is usually to increase the probability of matches
  - Backward compatibility often important here
  - Lots of false-positive matches
  - Interactions with CaseFold
  - Often loses information in round trips
Normalization

• We need to normalize strings before comparing them so that they are arranged the same way

• Decomposition
  • take things apart

• decomposition and reComposition
  • take them apart and put them back together
Normalization (2)

<table>
<thead>
<tr>
<th>Canonical</th>
<th>Kompatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompose</td>
<td>NFD</td>
</tr>
<tr>
<td>reCompose</td>
<td>NFC</td>
</tr>
</tbody>
</table>
Normalization (2)

Canonical  Kompatible

Decompose  NFD  NFKD
reCompose  NFC  NFKC
Encoding

• These are just spaces in code tables
• Need a way to represent it to computers
• It’s like Perl! More than one way!
• Generally UTF-8 on the wire for IETF protocols
  • You remember RFC 2277, right?
Stuff we tried

• Use UTF-8 (RFC 2277, RFC 3629)
  • Other things ok, but you need to negotiate
• Ok, in NFC (RFC 5198)
• Stringprep (RFC 3454)
Stringprep

• Take your string
• Run it through this fancy procedure
• Now you’re golden
The Stringprep Way

- Choose a Unicode version (oops, 3.2 only!)
- Choose a normalization form (NFKC or nothing?!)
- Specify how to handle whitespace
- Specify whether to use case folding (someone loses?)
- Specify bidirectional handling
- Specify prohibited characters
Stringprep facilities

• Tables!
  • Mappings
    • whitespace, folding
  • Prohibited characters
  • Bidirectionalality
• Each protocol does it for itself
  • For every version of Stringprep (or Unicode)
Stringprep issues

• Version agility is important

• NFKC yields some surprises

• Enormous repertoire very hard to comprehend (false positives and negatives)

• Big maintenance headache

• See RFC 4690 for more details
Try again

- IDNA2003 used Stringprep, so try again
- IDNA2008 uses algorithms based on Unicode properties
- Default is “not included”, rather than “included”
- *Should* be Unicode version agnostic
Four buckets

- PROTOCOL-VALID ("PVALID")
- DISALLOWED ("default" state before properties)
- Context rule required (CONTEXTO and CONTEXTJ)
- UNASSIGNED
Mostly works

• Goal is to internationalize old letter, digit, hyphen rule, not write literature

• Domain names not words

• Domain names have a registration authority
  • One per zone

• Maybe a useful pattern for other protocols
Some challenges

• We’re encroaching on user-interface space

• No mappings require some tricks in protocol deployment

• Depends on registration authority to do the right thing

• Depends on user agent to do the right thing (and know what that is in the locale)
Protocols want help

• “Can’t I just hand this to an expert subsystem and know what to do?”

• That was the idea behind PRECIS
PRECIS

• Takes approximately the same strategy as IDNA2008

• Generalized

• Gives two classes: IdentifierClass and FreeformClass

• Generate profiles for particular protocols
How to use it

• Figure out which of your protocol elements are user-facing. Those are the only ones you should internationalize.

• Figure out which of those elements are identifiers
  • Hint: if you need to match them ever, then they probably are
    • Use IdentifierClass
    • Otherwise, use FreeformClass

• There are some profiles, because the Classes are too big
Profiling

- Easy way: take an existing profile, and use that
- Hard way: take a profile, and try to modify it to accommodate your use case
- Hardest way: start with FreeformClass and start whittling away
Use the easy way

• These are really pretty comprehensive

• Yes, there’ll be some favourite character someone has that they really really want

• Is the case so compelling that it really makes things impossible to use, or is this a nice to have?

• Your WG can spend a million years in nice-to-have-land

• Safest to pick the smallest class you can get away with, not the largest set of characters anyone wants
If you must make it harder

- You *really* have to understand the particular characters, the cases in which they’re used, and so on.
- Much safer if you have a protocol with registration authority that can make decisions.
- Much safer if you’re absolutely sure no protocol element will leak off the LAN (on the grounds that a LAN is usually confined to a well-defined user population.
- You need an extensive bit of documentation in your Internationalization Considerations.
- If you’re different than other protocols, users may be surprised.
Windmill-tilting (rolling your own)

- If you really think the right thing to do is start with Freeform and whittle it down, try hard to think again

- You will need to understand all of Unicode.
  - This is the step you were trying to avoid, remember?

- There are nasty corner cases we’re just learning about (more in a moment)

- Your goal is to ship a working protocol before heat death of the universe, not write long Internationalization Considerations

- If you’re really different than other protocols, users will think your protocol is awful
Late-breaking challenges

• Unicode 7.0.0 introduced a character called ARABIC LETTER BEH WITH HAMZA ABOVE (U+08A1)

• Earlier versions had ARABIC LETTER BEH and ARABIC HAMZA ABOVE

• This led us to do some digging
Looks like this

U+08A1

U+0628  U+0654
So?

- Not canonically equivalent
- Surprising to some of us
- Turns out this is not completely strange
  - Some similar cases have been around since at least Unicode 3.2
  - Basic issue has nothing to do with Arabic script
More things for WGs

• You probably want guidance to operators

• IDNA2008 says, “If you’re going to register a character, you better understand it.” That is, “Make policies, and don’t do it blindly.”

• Other Internationalization Considerations or Security Considerations sections should probably say something similar

• There might be (a) new property(ies) coming, so the protocol could still change
What to read on PRECIS

• RFC 6885 (problem statement)
• draft-ietf-precis-framework
• draft-ietf-precis-saslprepbis
• draft-ietf-precis-nickname
Other things to see

- LUCID BoF (up next!)
- draft-sullivan-lucid-prob-stmt
- draft-klensin-idna-5892upd-unicode70