I18N in RFC3530bis

Dave Noveck

July 28, 2010

Overview

- I18N in RFC3530
  - How it was handled
  - Lack of correspondence with implementations
    - Or what might reasonably be expected
- What to do about it?
  - Change all the clients and all the servers?
  - Fix the internationalization chapter? **BINGO!**
- What has been done in latest draft given latter choice

In RFC3530

- Used stringprep
  - Group didn't have much choice
  - Back then, stringprep was the wave of the future and would solve i18n problems
  - If we didn't think so, we were trying to handle internationalization ourselves (and that was *bad*)
- So some stringprep verbiage was put in
  - Implementers probably didn't read it
  - And if they did, they probably didn't understand it
  - And if they did, they probably didn't implement it
  - And now we have to figure out what to do

Since Then

- Stringprep is no longer so wonderful
  - Issues have been found with German eszett, word delimiter characters in Farsi, etc.
  - Newprep will solve all the problems when it exists
- Had some free time to address this
  - I actually read the chapter and stringprep RFC and tried to understand what it meant.
  - Decided that implementations I knew didn't match it
  - Wondered if there were any that did
  - Guessed not

So what is so bad?

- Every string type assigned to stringprep profile
  - Wasn't done too well
    - E.g. tag strings, names, and link texts all have same profile
    - So they are all supposed to be handled *exactly* the same
- So let's focus on "exactly the same"
  - Stringprep really means it
  - Fixed character repertoire based on Unicode 3.2
    - Then lots of characters excluded
    - File systems and existing file names not considered
      - May not add characters
    - But RFC3530 allows them to excluded (BADCHAR)

So as written it says

- You MUST treat as errors tags that:
  - Aren't UTF-8
  - Are UTF-8 but contain characters that are not in Unicode 3.2
  - Or have one of several thousand characters that the stringprep profile excludes (for a range of reasons)
- Similarly for name components
  - Plus you must edit names that contain certain characters (deleting certain characters)
  - Including some that affect string meaning (e.g. word break characters in Farsi)
- Usable with very limited set of FS's
  - But client cannot depend on character set (BADCHAR)
  - RFC3530 suggest the server may pick normalization form and reject names sent by client in a different form
  - See slide 13 in appendix for details on these issues.

Interoperability choices for names

- The stringprep sort of choice
  - Every detail of name processing is specified
  - If you have clients and server+FS's that obey, can interoperate.
    - If that's the null set, too bad.
- The current RFC3530bis choice
  - Main goal is to allow use with all FS's
  - Clients should not be unduly constrained
  - Maximum interoperability within that framework
- The RFC3530 choice
  - Specify almost all details but not all
  - Enough to limit the set of server+FS's, so protocol not useful
  - But provide enough flexibility so there is no possibility of reliable client-server+FS interoperability
  - Worst choice of all

So let's start again

- Divide strings into two classes
  - Stuff for which there is an internet standard
    - Domain names, server names
    - We're letting IDNA handle it
  - Stuff which is up to implementation
    - Names up to FS's and not for the IETF to legislate
    - Sets of principal names are up to client-server implementation agreement
    - Protocol's job is to help communicate them and so there are rules to help interoperability
    - Not rules to legislate every detail
- For details see slides 14 and 15

Name components (1)

- As an example, old/new treatments of name components
- Character repertoire:
  - WAS: Unicode 3.2 must be used
  - NEW: Should use recent of version of Unicode; May treat as string of bytes with no UTF-8 checking
- Character mapping:
  - WAS: Must use stringprep tables with known problems
  - NEW: Should not use anything outside those tables, but should avoid known problems
- Normalization:
  - WAS: Didn’t say any anything, but talked of error on normalization mismatch
  - NEW: Allows server many choices but server must not reject strings based on normalization form

Name Components (2)

- Prohibited Characters
  - OLD:
    - Big long stringprep list.
    - Implied rejection of anything outside Unicode 3.2 under repertoire.
    - Server/FS may reject other things with BADCHAR
  - NEW: Server/FS may reject characters with BADCHAR
- Bidirectional strings
  - OLD: No provision (rejection possibly allowed under BADNAME).
  - NEW: Explicitly up to Server/FS and BADNAME to be returned if rejected

Big controversies

- IDNA format (A-label/U-label)
  - Allow A-label or UTF-8 string
- Name normalization
  - Allow many choices
    - Normalization NFC or NFD (and server converts)
    - Normalization-sensitive
    - Normalization-insensitive but preserving
- The big one. Two choices:
  - Specifying interoperable v4 service including FS for which interoperating with existing FS's is not a concern
  - NFSv4 specifies a protocol for the client to interact with existing and future FS's which becomes "They're trying to handle I18N themselves" which is true because that is the not the problem they're dealing with in the xyz-prep series.

Additional Slides

Interoperability considerations

- And stringprep's need to control
  - Eliminating (some) characters that might be confused with others
- Could this be justified by interoperability?
  - But with filesystems, we want to be interoperable with others storing file names in those FS's
  - If local file system can store name, we want to access it
  - Also, FS free to exclude additional characters as BADCHAR
  - But it can't add accept additional ones? This is a pretty strange interoperability story.
- Worse is how RFC3530 handles normalization
  - Assumes server may pick normalization form
  - And talks about error to return if the client uses a different normalization form

What about stringprep?

- Still use the stringprep format:
  - Character set, mapping, normalization, excluded chars, bidirectional string handling
  - Instead of saying exactly what must be done, says what may, should, must be done
  - In a few places refer to stringprep tables
- Still a few normative references to stringprep
  - A few table references (but more liberal: any subset OK)
  - Could eliminate but chose not to
    - Seemed like kicking an RFC when it was down

New typedef structure

- Old: utf8str_cs, utf8str_cis, utf8str_mixed
  - Put tags and filename components together
- New:
  - **utf8_should:** Should be utf8 but no checking
  - **utf8val_should:** Should be utf8 and checking generally done but MAY be inhibited
  - **utf8val_must:** Must be utf8 and must be validated as utf8
  - **ascii_must:** Must be ASCII and check although not necessarily in a validation step
- Works out a whole lot better