Internationalization in Names and Other Identifiers

IAB
Goals

• The plenary’s goal is to inform the community
  – Internationalization is often understood by a relatively small number of experts, but affects a large number of protocols

• IAB draft contains some recommendations regarding choice of encodings
  – draft-iab-idn-encoding-01.txt still in progress

• More work is needed and should continue
Why is Internationalization Important and Timely?
Introduction

• Names can have non-ASCII characters and are embedded in various ways:
  – Hostname / IDN: café.com
    (Internationalized Domain Name)
  – Email: 例え@テスト.com
  – URL (actually IRI): http://مثال.اختبار/
    • (Internationalized Resource Identifier)
  – UNC path: \例えテスト\public\file.doc
    (Universal Naming Convention =
     file paths common in Windows-based environments)

• Users want to browse the web, etc. in their own language
  – Imagine typing in a name in a script & language you don’t know
Situation Today/Soon

• China uses IDNs for all govt. sites and has IDN TLDs (中国, 公司 and 网络)
  – But are not in the public root today
• 35.2% of Taiwan domains are IDNs
• 13.7% of Korean domains are IDNs
• Vocal demand from the Arabic-script world
• ICANN is expected to start issuing IDN country-code Top Level Domains soon
Introduction and Terminology
Some Unicode Terminology

- **Unicode**: A set of integer code points in the range 1 – 1,114,111 (1 – 0x10FFFF) where each code point represents (with some exceptions) a human-meaningful visual “character”
- **UTF-32**: Each Unicode integer code point stored using a single 32-bit integer (so endianness matters)
- **UTF-16**: Each Unicode integer code point encoded using one or two 16-bit integers (so endianness matters)
- **UTF-8**: Each Unicode integer code point encoded using one to four 8-bit integers (so no endianness problems)
RFC 2277: IETF Policy on Character Sets and Languages

January 1998

Protocols MUST be able to use the UTF-8 charset
UTF-8

• Code points 0x00 – 0x7F same as ASCII
  – Code points 0x00 – 0x7F encoded using octet values 0x00 – 0x7F
  – So all current 7-bit ASCII files are also valid UTF-8
    • with the same meaning
    – Existing files already assigning other meanings to octet values 0x80 - 0xFF (e.g. ISO 8859-1) are not automatically compatible

• Higher code points use multi-octet sequences
  – Multi-octet sequences use octet values 0x80 – 0xF4
UTF-8 Multi-Octet Sequences

- Single octet ASCII character (Code points 1-127)
  - 0XXXXXXX

- First octet of 2, 3, 4-octet sequences
  - 110XXXXX
  - 1110XXXX
  - 11110XXX

- Continuation octets of multi-octet sequences
  - 10XXXXXX

IETF 76 Technical Plenary
UTF-8 Multi-Octet Sequences

00000 – 0007F

00080 – 007FF

00800 – 0FFF

10000 –
UTF-8 Properties

• No mid-string zero octets
• Stateless character boundary detection
  – Robust to insertions, deletions, errors, etc.
• Strong heuristic detection
  – E.g. Any lone octet with top bit set signals text as not valid UTF-8
• Byte-wise, sorts same order as raw Unicode
Compactness: How many octets does it take to represent a string?

• Everyone creating their own ‘optimal’ solution (optimal in some specific context) comes at a high price in terms of interoperability

• Relative compactness for different encodings is not nearly as important on today's systems as in the past
  – Text is tiny compared to today’s other data:
    • Images, Audio, Video

• Even international text often contains ASCII markup
  – E.g. HTML tags in otherwise international text file
Case Study:
Localization Strings in Apple’s Mail.app

- /Applications/Mail.app/Contents/Resources/Japanese.lproj/Localizable.strings
- UTF-16: 117,624 bytes
- UTF-8: 68,693 bytes

"UNDO_MARK_READ" = "開封済みにする";
Punycode

• Used for Internationalized Domain Names
• A method of encoding a string of Unicode integer code points using only the following octet values:
  ➢ 0x2D
  ➢ 0x30 – 0x39
  ➢ 0x61 – 0x7A
• i.e. octet values that, if (mis)interpreted as US ASCII, correspond to the following US ASCII characters:
  ➢ Hyphen
  ➢ Digits 0 – 9
  ➢ Letters a – z
A Question of Interpretation: ASCII or not?
A Question of Interpretation: ASCII or not?
A Question of Interpretation: ASCII or not?
Today’s Text Chaos

**Technical Details**

- Stackable & Easily stacks with the Apple Mac mini and Airport Extreme or additional Iomega MiniMax hard drives
- Secure & Micro security slot designed to allow drive to be anchored to a desk
- Convenient & FireWire 1394a 3-port repeater/hub
Today’s Text Chaos
IDNs in Other Identifiers

There can be many ways to get to the same file. For example...

Using HTTP:
1. http://dthαler/Public/test.htm
3. http://dth%CE%B1ler/Public/test.htm
4. http://dth&#x3b1;ler/Public/test.htm

Using CIFS/SMB (file system protocols using UNC):
1. file://dthαler/Public/test.htm
2. file://xn--dthler-rxe/Public/test.htm
3. file://dth%CE%B1ler/Public/test.htm
4. file://dth&#x3b1;ler/Public/test.htm
5. \dthαler\Public\test.htm
6. \xn--dthler-rxe\Public\test.htm
Plenary Announcement
to ietf-announce@ietf.org

smooth and interoperable functioning\^\text{\#8232;} of the Internet depends on text strings\^\text{\#8232;} being interpreted in the same way\^\text{\#8232;} by all systems connected to it.
How Many Layers of Encoding?

• How do we encode:
   A domain name...
   in an email address...
   in a “mailto” URL...
   in a web page?

• Do we use:
  – Punycode ("xn--...") encoding for the domain name?
  – Email Quoted-Printable ("=XX") encoding?
  – URL percent ("%XX") escaping?
  – HTML ampersand ("&#xxxx;") codes?

• All of the above?
IDNs in Email

• Two test emails
• From: user@chεshírε.stuartcheshire.org
• In each email, address appears in two places:
  – in “From” line
  – and in body text
• First email encoded IDN using Punycode:
  \texttt{xn--chshr-38d3be}
• Second email encoded IDN using direct UTF-8
  \texttt{chεshírε}
Punycode Email

Subject: Punycode
From: user@xn--chshr-38d3be.stuartcheshire.org

The "From" address for this email was "user@xn--chshr-38d3be.stuartcheshire.org" (i.e. using punycode encoding)
UTF-8 Email

Subject: UTF-8
From: user@chêsšíre.stuartcheshire.org

The "From" address for this email was "user@chêsšíre.stuartcheshire.org" (i.e. using direct UTF-8 encoding)
## Punycode Email: xn--chshr-38d3be

<table>
<thead>
<tr>
<th>Client</th>
<th>From (Punycode)</th>
<th>Body (Punycode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gmail / IE</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Gmail / Firefox 3</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Apple Mail</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Penelope</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Mulberry 4.08</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Thunderbird 2.0.0.16</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Eudora 6 on Mac OS X</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Lotus Notes 7.03 &amp; 8.01</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Outlook 2007</td>
<td>chεshíε</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Outlook E-Mail (WM6)</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
<tr>
<td>Outlook Web Access / IE</td>
<td>xn--chshr-38d3be</td>
<td>xn--chshr-38d3be</td>
</tr>
</tbody>
</table>
## UTF-8 Email: chẹshírẹ

<table>
<thead>
<tr>
<th>Client</th>
<th>From (UTF-8)</th>
<th>Body (UTF-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gmail / IE</td>
<td>xn--chshr-38d3be</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Gmail / Firefox 3</td>
<td>xn--chshr-38d3be</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Apple Mail</td>
<td>chẹshírẹ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Penelope</td>
<td>chẹshírẹ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Mulberry 4.08</td>
<td>chẹshírẹ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Thunderbird 2.0.0.16</td>
<td>chẹshírẹ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Eudora 6 on Mac OS X</td>
<td>chÎµsh¯ rÎµ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Lotus Notes 7.03 &amp; 8.01</td>
<td>chÎµsh¯ rÎµ</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Outlook 2007</td>
<td>ch??sh??r??</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Outlook E-Mail (WM6)</td>
<td>ch??sh??r??</td>
<td>chẹshírẹ</td>
</tr>
<tr>
<td>Outlook Web Access / IE</td>
<td>ch??sh??r??</td>
<td>chẹshírẹ</td>
</tr>
</tbody>
</table>
More Terminology in this Presentation

• **Mapping**: converting one string to another “equivalent” string
  - “CONTOSO.com” ⇒ “contoso.com”

• **Matching**: checking two strings for equivalence
  - “CONTOSO.com” ~ “contoso.COM”
  - “möhringen.de” ≠ “moehringen.de”

• **Sorting**: determining which string comes first
  - “contoso.com” < “Microsoft.com”

• **Encoding**: same string can be encoded in different ways
  - including issues of combining characters: é vs e + ´
IDN Identifier Space

• **IDNA-valid string:** no invalid characters, legal length, etc.
• **U-label:** a Unicode IDNA-valid string
• **A-label:** “xn––” followed by Punycode-encoded IDNA-valid string
More Variety Brings More Ambiguity

- Computer Systems: 2 (binary)
- Telephone Numbers: 10 (0-9)
- ASCII Domain Names: 37 (A-Z, 0-9, -)
- International Domain Names: Tens of thousands
Matching
Confusable Strings (1/4)

- Two strings that are easily confused by a human
  ETHIOPIA.com ↔ ETHIOPIA.corn
  Greek alphabet! Plain “ASCII” confusion

- Lower-casing will reveal the ETHIOPIA issue
  - ετηιορια is fairly distinctive
    - current trend is to deprecate upper-case
      and other mapping-required forms in IRIs etc.
    - IDNA2008 treats these characters as DISALLOWED
Confusable Strings (2/4)

• Another example:

  jessica.py ↔ jessica.py

  .py = Paraguay

  Cyrillic alphabet

  .ru = Russia

• “jessica” actually uses Cyrillic characters from two separate languages
  – A registry may restrict registrations to only characters in their language

• Other examples exist without mixing languages
  • epoxy.py ↔ epoxy.py
Confusable Strings (3/4)

• People see what they expect to see
  – Russian restaurant: “ресторан”
  – Non-Russians might read “pectopah”

• Given sufficiently creative use of fonts forced by style sheets etc., confusion can be easy
It Depends on What You Know – and Expect

Is the second character “A”? If you were not familiar with Latin script, or didn’t know what to expect, would you be sure? Could it be a star of some sort? Are you sure that the first character is an ASCII dot (the DNS cares – a lot)?

Are these two strings identical? Are you sure? Would you be sure if you didn’t know Latin script or the organization involved?
Confusable Strings (4/4)

• Other kinds of “equivalence” — equality in some contexts

• 中国 and 中國
  Two code points, same concept

• السعودية and السعودية
  Two code points for same letter (more or less)
Are the Following Equivalent?

<table>
<thead>
<tr>
<th>Script</th>
<th>0 1 2 3 4 5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic-Indic</td>
<td>۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹</td>
</tr>
<tr>
<td>Eastern Arabic-Indic</td>
<td>۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹</td>
</tr>
<tr>
<td>Chinese Suzhou</td>
<td>一 二 三 亖 <em>SINGLE</em></td>
</tr>
<tr>
<td>European (ASCII)</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>Devanagari (Hindi)</td>
<td>० १ २ ३ ४ ५ ६ ७ ८ ९</td>
</tr>
<tr>
<td>Tibetan</td>
<td>༠ ༡ ༢ ༣ ༤ ༥ ༦ ༧ ༨</td>
</tr>
</tbody>
</table>
| Tamil                 | எ ஒ ஐ ஑ ஓ ஔ க க க
“Suspect” Names

• Potential for phishing attacks
  – but could be innocent or accidental
• Names with scripts not used by the user’s locale
• Names with mixed scripts (e.g. Cyrillic + Latin)
• UI might want to warn the user when displaying any of these from an untrusted source
  – Some browsers display A-labels (“xn--...”) in address bar, but that confuses humans
Universal Confusable String

• Few user systems have all possible characters and display fonts for them installed.
• If character cannot be represented locally, a six-character string might appear as
  – ❓❓❓❓❓❓ or
  – ☐☐☐☐☐☐
• This should be a warning (but remember what users do when they see a warning they don’t understand)
Mapping
Why Mapping?

• Instead of intelligent matching algorithm:
  – Map each string to a defined canonical form
  – Simple test if canonical forms are bitwise identical
  – Does not permit “close enough” or other fuzzy matching

• Conversion of one visual form to another that is more locally understandable
  – E.g. Traditional Chinese (中 國) to Simplified (中 国)
Mapping

• Mapping inherently loses information
  – Case conversion, half/full width, NFC/NFKC/etc
• Upper/lower casing differs by language
  – Latin alphabet: $l \leftrightarrow i$
  – Turkish: $l \leftrightarrow ı$ $\hat{l} \leftrightarrow i$
• tolower(‘l’) = ???
• toupper(‘i’) = ???
• tolower(toupper(‘i’)) ≠ ‘i’
• Turks aren’t too happy about this…
Mapping

• Summary:
  – Never roll your own mapping
  – Correct mapping for user depends on language context, which we often don’t know
Encodings
draft-iab-idn-encoding-01.txt
Problem #1: DNS isn’t the only name resolution protocol

Different protocols use different encodings today

Problem #2: The public Internet name space isn’t the only name space

Different name spaces use different encodings today
Realistic Network Architecture

- Host
  - Application
  - Name Resolution Library
    - DNS
    - mDNS
    - LLMNR
    - NetBIOS over TCP
    - hosts file
    - NIS
    - Implementation-specific encoding (e.g., UTF-8 or UTF-16)

- Enterprise network
- Local LAN
- Internet
- ...
Other Name Resolution Protocols

• Many defined to use \textit{the same syntax}
  – Hosts file, DNS, mDNS, NetBIOS-over-TCP, etc.

• Name resolution library decides what protocols to try in what order
  – Apps cannot tell from the name what protocols will be used for resolution
  – Different libraries may use different order and hence find different name targets

• Different protocols specify use of different encodings
  – Apps cannot tell what encodings will be needed for resolution
What’s a Legal Name?

• In 1985, RFC 952 defined the format of the hosts file:
  – “Internet host/net/gateway/domain name” contains ASCII letters, digits, hyphens (LDH)

• In 1989, RFC 1035 defined DNS:
  – “Preferred name syntax”: LDH
  – But does “preferred” mean MAY/SHOULD/MUST?
Legal DNS Names

• In 1997, RFC 2181 clarified:
  – Any binary string whatever can be used as the label of any resource record
  – Any binary string can serve as the value of any record that includes a domain name
  – Applications can have restrictions imposed on what particular values are acceptable in their environment

• Same year:
  – IETF policy on character sets and languages...
IETF Policy on Character Sets and Languages (RFC 2277)

• It says:
  – Protocols MUST be able to use the UTF-8 charset
  – Protocols MAY specify, in addition, how to use other charsets or other character encoding schemes
  – Using a default other than UTF-8 is acceptable

• Silent on different forms within UTF-8 (e.g. case, encoding of combining characters, sort order)
  – Two Unicode strings often cannot be compared to yield results users expect without additional processing

• Per RFC 2181, DNS complies
Use of Different Encodings in DNS

• Starting that year (1997), some systems began using UTF-8 in DNS in private name spaces
  – Private name space here means names are not resolvable from outside the specific network

• About five years later, IDNA development (including Punycode) began for use in the public DNS name space
Length Issues

• DNS names have
  – 63 octets per label
  – 255 octets per name (not counting zero at the end)
• Most application APIs use NULL-terminated strings
• Non-ASCII characters use a variable number of octets in UTF-8, UTF-16 and Punycode
  – 256 UTF-16 octets ≠ 256 UTF-8 octets ≠ 256 A-label octets
• Some names can be represented (within length) in Punycode A-labels but not in UTF-8
• Some names can be represented (within length) in UTF-8 but not in Punycode A-labels
Let’s Recap Where We Are…

• Multiple encodings of same Unicode characters:
  – U-labels: مثال.اختبار
  – A-labels: xn--mgbh0f.xn--kgbechtv

• Different encodings used:
  – By different protocols
  – On different networks with DNS
    • Punycode A-labels used on Internet, UTF-8 in intranets
  – By different applications

• Results:
  – Failure — or worse — launching one app from another
  – Competitor switching incentives, and poor user experience when one app works and competitor doesn’t
Example “IDN-aware” app: Browser picks encoding based on intranet vs. Internet

Problem 1:
Host/app B uses A-labels in intranet

Problem 2:
Local name resolution uses UTF-8

Intranet name?

Internet name?

Name Resolution API

example.com (UTF-8)

xn--mgbh0fb.example.com

xn--mgbh0fb

Intranet

Internet

DNS

DNS

LLMNR/mDNS

IETF 76 Technical Plenary
Inconsistent Experience Across Applications

IDN Aware App

Non-IDN Aware App

Phishing attacks possible
Other IDN-Aware Apps
Lack consistency, causing non-deterministic experience
Basic Principle

• Conversion to A-labels, UTF-8, or whatever other encoding, can be done only by an entity that knows which protocol and name space will be used
Hard Issues 1 of 2

• Client has to guess or learn what encoding a {HTTP, DNS, SMTP, ...} server expects for an identifier

• Names appear inside many other types of identifiers, e.g. email address, URLs, UNC paths, network access identifiers (NAIs)
  – Each identifier type has its own encoding conventions
  – Today, apps that extract host names need to convert encodings
Hard Issues 2 of 2

• Use of a single encoding is the easy part
  – Sufficient only if the only intent is to display
  – Comparison, matching, lookup, sorting, etc., all require more work.
  – Just as RFC 952 defined an ASCII subset for “hostname” identifiers, we need to define Unicode subsets for other types of identifiers.

• Optimal subset for one protocol may not be optimal for another.

• Interpretation and display of some strings may differ by operating systems – usually a bug, but sometimes no agreement as to which variation is the bug.
Conclusions

• Smooth and interoperable functioning of the Internet depends on text strings being interpreted in the same way by all systems connected to it

• The IETF has recognized this since RFC 20 specified ASCII for use in interchange in 1969 — the suggestions in this presentation extend and update that understanding as well as the understanding in RFCs 2277 and 5198
Conclusions

• To avoid confusion and ambiguity, it is not enough merely to support UTF-8 as *one* of the text encoding options
  – Text in protocols on the wire should be in UTF-8 and *only in* UTF-8

• For user-visible text in protocols:
  – If you don’t use UTF-8, why not?

• For protocol identifiers not seen by users:
  – If you do allow the full Unicode character range, why?