Introduction to Universal Acceptance

Universal Acceptance is the ability to Accept, Store, Process and Display all Top Level Domains equally and all IDNs, IRIs and email addresses equally.

Contents

Baseline Concepts

Domain Name System (DNS)

Each resource on the Internet is assigned an address to be used by the Internet Protocol (IP). Since IP addresses are difficult to remember, servers collectively providing a public Domain Name System (DNS) exist at well-known addresses on the Internet. DNS provides the mapping between IP addresses and human-readable domain names.

Top level Domains (TLDs)

Human readable domain names are managed by companies known as registrars. When one registers a domain, it will consist of multiple text strings representing multiple domain levels, each separated by a "." character. In most scripts, the right-most domain level is the top-level domain (TLD).

Examples of well-known TLDs

- `.com`
- `.gov`
- `.info`
- `.org`

Some TLDs are assigned to specific countries, and are called Country Code TLDs.

Examples of ccTLDs

- China = `.cn`
- Russia = `.ru`
- United States = `.us`

Generic Top Level Domains (gTLDs)

Starting in 2013, ICANN (the organization responsible for the creation and maintenance of TLD assignments) has generated a large number of new TLDs. Most of these new TLDs are for brands, and others are generic. Nevertheless, all of these new TLDs are usually known as Generic Top Level Domains (gTLDs).

Examples of new gTLDs

- `.apps`
- `.lawyer`
- `.shopping`
- `.panasonic`
- `.osaka`

Character Sets and Scripts

All languages are written in characters and scripts. While these characters or scripts are recognized by humans, they are not useful to computers, which only process numbers. To resolve this, a character mapping (also called coded character set or code page) can be created to associate characters with specific numbers. Many different code pages have been created over time for different purposes, but for this topic we will focus on only two.

ASCII & Unicode

In the examples above, all of the text strings are represented using the English character set. This character set is included in American Standard Code for Information Interchange (ASCII, or US-ASCII) character-encoding scheme. ASCII is an older encoding scheme and was based on the English language. For historical reasons it became the standard character encoding scheme on the Internet. ASCII uses only 7 bits per character, which limits the set to 128 characters.

Because most languages do not use the English character set, alternate encodings have subsequently been adopted. Unicode (also known as the Universal Coded Character Set, or UCS) is capable of encoding more than 1M items. Each of these Unicode items is a called a code point. The most common form of Unicode is called Universal Coded Character Set Transform Format 8-bit (UTF-8).

Internationalized Domain Names (IDNs) & Punycode

By using Unicode instead of ASCII, domain names can contain non-English characters. Domain names using any non-ASCII characters are called Internationalized Domain Names (IDN). Since the DNS itself previously only used ASCII (see http://tools.ietf.org/html/rfc6055#section-3 for current status), an additional encoding was created to allow Unicode code points to be converted into ASCII strings. The resulting strings can be quite large due to the number of valid Unicode code points.

The algorithm which implements this Unicode-to-ASCII encoding is called Punycode, and the output strings are called A-Labels. A-Labels can be distinguished from an ordinary ASCII label because they always start with the characters "xn--" (which is called the ACE prefix). The Punycode transformation is bi-directional. It can transform from Unicode to an A-Label and also from an A-label back to Unicode.

The only valid use of the Punycode algorithm is for communicating with DNS. This has not stopped some developers from applying it to other scenarios rather than implementing Unicode. See Best Practices for more information.

The internationalized portion of an IDN can be in any level, not just the top level. Some gTLDs are IDNs, but not all IDNs are TLDs.

Examples of (imaginary) IDNs

- everyone.みんな (Punycode encoding = everyone.xn–q9jyb4c)
- 大坂.info (Punycode encoding = xn–uesx7b.info)
- みんな.大坂 (Punycode encoding = xn–q9jyb4c.xn–uesx7b)

Email Addresses using gTLDs and IDNs & Email Address Internationalization (EAI)

Email addresses contain contains two parts: a local portion (the user name, to the left of the @ character) and a domain (to the right of the @ character). The domain portion can contain any TLD, including a gTLD. Both portions may be internationalized (an IDN).

Examples of Email Addresses using gTLDs and IDNs

- user@everyone.lawyer (uses new gTLD)
- user@everyone.みんな (uses international TLD)
- user@大坂.info (uses international 2nd level domain)
- 用戶@everyone.lawyer (uses international user name and new gTLD)

Email Address Internationalization (EAI) requires the use of Unicode in all portions of the email address. Each of the 4 examples above could be expressed as EAI (even user@everyone.lawyer which contains only Latin characters), and this is the preferred format.

NOTE: An additional format, IDN-Style Email Addresses, will be discussed below.

Universal Acceptance in Action

Definitions

As mentioned above, Universal Acceptance is

> *The ability to Accept, Store, Process and Display all Top Level Domains equally and all IDNs, IRIs and email addresses equally*

These 4 criteria are described below.

- *Accept* – Applications and services may allow domain names and email addresses to be entered into user interfaces and/or received from other applications and services via APIs. Usually this process includes a validation to verify that the data has been presented in a valid format. Meeting this criterion depends on the application or service being aware of current valid formats, which include different lengths and character sets than was the case previously.
- *Store* – Applications and services may require long-term and/or transient storage of domain names and email addresses. Regardless of the lifetime of the data, it must be stored either in RFC-defined formats, or in proprietary formats which can be easily translated to and from RFC-defined formats (the latter is much less desirable). Although RFCs require the use of UTF-8, other formats may be encountered in legacy code. See Best Practices, below.
- *Process* – Processing means using domain names and email strings in a feature. There is no limit to the number of ways that domain names and email addresses could be processed (e.g. "identify all the people in New Zealand because they have a name with a .nz ccTLD", "identify all the pharmacists because they have an @pharmacist email address", or firewalls that might filter DNS requests that don't apply to their conventions.)
- *Display* – Displaying domain names and email addresses will be straightforward when the scripts used are supported in the underlying OS and when the strings are stored in Unicode; if these conditions are not met, application-specific transformations may be required.

User Scenarios

The examples and definitions above may give the impression that Universal Acceptance is only about computer systems and online services. It's actually about real people using those systems and services.

Here are some examples of activities which may require Universal Acceptance.

- *Registering a gTLD* – A user wants to use a browser to view the website of a retailer. The retailer previously had a .com domain, and has encountered spoofing in the past (people who register a similar .com name and hope to fool users); now the company has acquired a gTLD for their brand. Not only does the gTLD reinforce their brand identity, they own the gTLD; no one else can use it and users can be confident that they are accessing the correct site.
- *Accessing a gTLD* – A user accesses a web site which has a gTLD. Even though the gTLD is new, any browser the user wishes to use will display the web address correctly and access the site correctly.
- *Accessing an IDN* - A user types an address or clicks a link pointing to a web site which has an IDN. Even if the IDN uses characters different than the language settings on the user's computer, any browser the user wishes to use will display the web address correctly and access the site correctly.
- *Using an international email address* – A user has acquired multiple email addresses and associates them as aliases to a common email inbox. Even though some of the email addresses are EAI, the user can send and receive email with them regardless who they communicate with or what email service they use.

Technical Requirements for Internet Internationalization

High level requirements

An application or service which supports universal acceptance:

- Supports all domain names regardless of length or character set (except where specific domains are explicitly blocked by IT policies)
- Allows entry of international characters (i.e., all Unicode code points) into all UI inputs
- Can correctly render all code points in Unicode strings
- Can correctly render right-to-left strings such as those in Arabic and Hebrew
- Can communicate data between applications and services in formats which support Unicode and are convertible to/from UTF-8
- Offers public APIs which support Unicode & UTF-8
- Offers private APIs which support Unicode & UTF-8 (these private APIs apply only to inter-service calls by the same vendor)
- Stores user data in formats which support Unicode and is convertible to/from UTF-8 (such conversions would be visible only to the product/service owner)
- Supports all domain name strings in the authoritative ICANN TLD list and the community-served Public Suffix List regardless of length or character set
- Can send email to recipients regardless of domain or character set
- Can receive email from senders regardless of domain or character set
- Supports accounts which are associated with both an ASCII and Unicode email address aliases

<NOTE: The requirements and scenarios should probably contain some cross-references to the RFCs. Not to overwhelm the reader, just to add some context.>

Developer Considerations

Since many existing software systems contain hardcoded assumptions about domains and email addresses, code changes may be required.

Best Practices for developing and updating Universal Acceptance Software

- Use supported Unicode-enabled APIs - don't spin your own.
    - String format conversions
    - Determining which script comprises a string
    - Determining if a string contains a mix of scripts
    - Unicode normalization / decomposition
- Non Unicode data should be converted to Unicode before data transactions (passing data from file to file, system to system, application to application) occur. This is because data in one non-Unicode code page may not be represented in another non-Unicode code page leading to possible data loss or corruption.
- All server responses should have the Unicode specified in the content type.
- All data should be converted to Unicode before data processing (retrieval, alteration, manipulation) begins.
- Non Unicode data must be converted to Unicode before they are displayed.
- Don't display punycode to the user. Don't ever require the user to enter data in punycode.
- Applications should convert data to Unicode before storing them in databases and files. Multiple character sets in databases is a typically common and severe problem that is expensive to resolve. It makes sorting, data copy, data import and export, data retrieval by client applications rather difficult and may result in data loss or corruption.
    - If you *don't* store in Unicode, must be able to match strings in multiple formats (a search for `everyone.みんな` should also find `everyone.xn–q9jyb4c`.)
- Ensure that the product or feature handles sort order, searches, and collation according to locale/language specifications, and that it addresses multilanguage searching and sorting.
- Use authoritative resources to validate domain names. Do not make heuristic assumptions, such as "all TLDs are 2, 3, 4, or 6 characters in length."
- Recognize that mixed-script is going to become more common. Don't assume that mixed-script strings are intended for malicious purposes, such as phishing, and if your user interface calls such strings to the user's attention, be sure that it does so in a way which is not prejudicial to users of non-Latin scripts.
- Don't generate IDN-Style email addresses, but be able to handle them if presented by someone else's software.
- Never use UTF-7 or UTF-32.
- Avoid UTF-16 except where it is explicitly required (as in certain Windows APIs)
    - When using UTF-16, note that 16 bits can only contain the range of characters from 0x0 to 0xFFFF, and additional complexity is used to store values above this range (0x10000 to 0x10FFFF). This is done using pairs of code units known as surrogates.
    - If handling of surrogate pairs is not thoroughly tested, it may lead to tricky bugs and potential security holes.
- Ensure that the product or feature handles numbers correctly. For example, Roman numerals and Asian ideographic number representations should all be treated as numbers.
- Upgrade your app and server/service together. If the server is Unicode and client is non-Unicode, or vice versa, data needs to be converted to each code page every time the data travels from server to client or vice versa.
- Use MIME for email encoding.
- Use GB18030 for Chinese language support.
- Preventing users from typing in Punycode-encoding names is not required. If a valid Punycode-encoded name is input, the application should convert it and display it as Unicode.

- Unicode should be specified in the web server http header and directly in a web file. Every web file should include the UTF-8 charset. It is important to ensure that the encoding is specified on every response.
- Cookies should use Unicode so it can be read correctly by applications.
- Look for mail addresses in unexpected places:
    - Artist/Author/Photographer/Copyright metadata
    - Font metadata
    - DNS contact records
    - Binary version information
    - Support information
    - OEM contact information
    - Registration, Feedback, and other forms

"IDN-Style email", and why it is not the same as EAI

Because IDNs can be Punycode encoded, some existing software allows the IDN portion of an email address to be represented in ASCII or Unicode. For example, in this "IDN-Style email" method these addresses are equivalent for all purposes (sending, receiving, and searching):

- user@everyone.みんな
- user@everyone.xn--q9jyb4c

EAI is defined as using Unicode only; A-Labels are not allowed. Nevertheless developers have sometimes adapted email software and services to handle IDN-Style email addresses rather than make a full conversion to Unicode.

Universal Acceptance software and services should be able to handle IDN-Style email and convert it successfully to Unicode EAI format. Nevertheless, UA software should not generate IDN-Style email addresses - should support true EAI only.

Complex Scripts – TO DO

Right to left languages and Unicode conformance

-

"Joiners" - RFC 5894, 4.3. Linguistic Expectations: Ligatures and Digraphs

-

Other Complex scripts

-

Other Challenges

- Some older email applications were encoded in a local code page and they did not have a set mechanism for detecting and converting charset as needed. This was especially true for the email header (i.e. TO, CC, BCC, Subject).
- It is possible for an application to recognizing homographs (characters which appear identical or nearly-identical when rendered but which are distinct code points), but it isn't always useful to point these out to

a user. Application writers (particularly for spam filters) may elect to scrutinize mixed Latin, Cyrillic and/or Greek strings as suspicious combos or phishing attempts. This does run the risk of flagging what may become a common scenario, with negative impact to user experience and dubious protection as users learn to ignore the majority of these strings as false positives.

- When allowing a user to generate a domain name or email address, consider avoid using visually confusing characters to prevent homograph attacks.
    1. Use only this allowed list of characters for IDN: http://unicode.org/reports/tr36/idn-chars.txt
- When a user is aliasing multiple email addresses it may be tricky to manage these addresses as a single user identity. Email programs can direct traffic to such aliases to the same mailbox, but the application will still perceive these emails to pertain to different identities.
- Normalization of text, such as conversion to lowercase or uppercase, or ignoring nonspacing characters, may make the results of comparisons with the back-end data store unpredictable.
- Code reviews are recommended to avoid buffer overflow attacks.
    1. In Unicode, strings may expand in casing: Fluß → FLUSS → fluss. When doing character conversion, text may grow or shrink substantially.
-

Examples of Non-Universal Acceptance – TO DO

- *Displaying punycode to the user*
- *Exposing punycode to a domain owner*
- *Downgrading Example:*
    - EAI Clients and Servers can communicate using UTF-8 protocols, but ASCII clients are limited to existing SMTP/MIME protocols. Mail from EAI to ASCII systems cannot be sent or must be downgraded. Mail from ASCII systems to EAI systems remains in ASCII form.

Glossary and other resources

Cross-link to RFCs

- RFC 3492 (Punycode)
- RFC 5890-94 (IDN)
- RFC 6530-33 (EAI)
- ISO 10646 (Unicode)
- GB18030 (China)

Online resources – TO DO

- Windows APIs
- SharePoint APIs
- Public Suffix List
- ICANN Authoritative TLD list
- Android & iOS APIs
- Unicode Security considerations http://www.unicode.org/reports/tr36/
- Unicode security mechanisms http://www.unicode.org/reports/tr39/
- For more details on Unicode character groupings, read the following:
- Unicode planes http://en.wikipedia.org/wiki/Mapping_of_Unicode_character_planes
- Overview of GB18030
- GB18030 FAQ - Unicode normalization
- http://unicode.org/reports/tr36/#UTF-8_Exploit - does this still apply in latest version?
- Unicode.org for security exploits

Topics for potential future proposal to ICANN – TO DO

It is desirable for the presence of a brand TLD to make *users confident that they are accessing the correct site.*

Due to differences in IDNA2003 and 2008, similar strings such as `foosball.de` and `foSSball.de` may or may not resolve to the same address. They might not even belong to the same owner!

Can/should we enforce "bundling" at the registration level for compatibility? (I.e. should we require a registry to sell both to the same customer?)

Should ICANN restrict the delegation of homograph domain names (at any level, not just TLD)?