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 of Universal Acceptance

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.

Define also?

Hostnames, directionality, structured text, IRI

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). Sometimes new gTLDs are known as nTLDs.

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 RFC-defined 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).

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

Examples of (imaginary) 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 (i.e. even user@everyone.lawyer which contains only Latin characters), and this is the preferred format.

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

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.

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, some software will treat these two "IDN-Style email" addresses equivalently for all purposes (sending, receiving, and searching):

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

Unfortunately, some software will not robustly treat these addresses as equivalent, which can result in unpredictable user experience as messages are replied-to or forwarded.

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.

Dynamic IRI generation ("Linkification")

Modern software sometimes allows a user to create an IRI simply by typing in a string that looks like a web address, email name or network path. For example, typing "www.icann.org" into an email message may result in a clickable link to http://www.icann.org being automatically created if the app treats "www." as a special prefix or ".org" as a special suffix.

Linkification should work consistently for all well-formed web addresses, email names or network paths.

(More needed?)

Universal Acceptance in Action

Four Criteria of Universal Acceptance

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 allow domain names and email addresses to be entered into user interfaces and/or received from other applications and services via APIs. If this process includes a validation to verify that the data has been presented in a valid format, meeting this criterion will depend 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 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. Additional validation may occur during processing. 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", or "identify all the pharmacists because they have a `user@*.pharmacist` email address", or firewalls that might filter DNS requests that don't apply to their conventions.)

Display

Displaying domain names and email addresses are usually 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 require Universal Acceptance.

Registering a new gTLD

An online retailer previously had a `.com` domain, and encountered spoofing in the past (by malicious people who registered a similar `.com` name and hoped 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 new gTLD, by typing an address into a browser or clicking a link in a document. Even though the gTLD is new, any browser the user wishes to use will display the web address correctly and access the site correctly.

Using an email address containing a new gTLD as an online identity

A user acquires an email address with the domain portion using a new gTLD, and uses this email address as their identity for accessing their bank and airline loyalty accounts. Even though the domain used in the email address is new, the site accepts the address exactly as if it were used an older domain such as `.com` or `.org`.

Accessing an IDN

A user accesses a web site which has an IDN, by typing an address into a browser or clicking a link in a document. 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 for mail

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.

Using an international email address as an online identity

A user acquires an EAI email, and uses this email address as their identity for accessing their bank and airline loyalty accounts. Even though the script used in the email address is different than the language settings of both their operating system and their browser, the site accepts the EAI identity exactly as if it were a non-EAI identity.

Dynamically creating an IRI in a document

A user types a web address into a document or email message. The rules used by the app to automatically generate an IRI based on this web address string are the same even if the address is an IDN or contains a new gTLD.

Developing an App

A developer writes an app that accesses web resources. The toolchain used by the developer includes libraries which enable Universal Acceptance by supporting Unicode, IDNs and EAI.

Examples of Non-Universal Acceptance

- Displaying Punycoded text to the user without a corresponding user benefit (i.e. to highlight a potential security risk)
- Requiring a user to enter Punycoded text when signing up for a new email address
- Requiring a user to enter Punycoded text when signing up for a new hosted domain
- Using heuristics to validate domain names rather than using an authoritative online domain name resource
- Caching or hard-coding valid domain names but infrequently updating the cached or hard-coded domain name data
- Exposing internal use of non-Unicode to users (e.g. converting from EAI to an IDN-style email address when replying to an EAI user)

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.

Robustness (Postel's law)

In RFC 793, Jon Postel formulated the Robustness Principle, now known as Postel's Law, as an implementation guideline for the then-new TCP:

"Be conservative in what you do, be liberal in what you accept from others."

This is also a good approach when dealing with the vagaries of Universal Acceptance currently implemented in the ecosystem.

Best Practices for developing and updating Universal Acceptance Software

Accept

- Don't ever require the user to enter data as Punycoded text.
- Do allow users to enter Punycoded text in place of its Unicode equivalent. If a valid Punycoded text string is input, the application should convert it and display it as Unicode.
- Don't generate IDN-Style email addresses, but be able to handle them if presented by someone else's software.
- Do allow up labels up to 63 characters.
-

Store

- Do convert data to Unicode before storing them in databases and files. Multiple character sets in databases is a common but 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.
- Don't normalize by converting to uppercase, or ignoring nonspacing characters, because this may also make 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 (e.g. a search for `everyone.みんな` should also find `everyone.xn--q9jyb4c`.)

Process

- Do convert non-Unicode data to Unicode before data transactions (passing data from file to file, system to system, application to application) occur. Data in one non-Unicode code page may not be represented in another non-Unicode code page leading to possible data loss or corruption.

- Do ensure all server responses should have the Unicode specified in the content type.
- Do specify Unicode 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.
- Do convert all data to Unicode before data processing (retrieval, alteration, and manipulation) begins.
- Do 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.
- Do use MIME for email encoding.
- Don't use URL-encoding for domain names (e.g. `domaintest.みんな is correct, but domaintest.%E3%81%BF%E3%82%93%E3%81%AA is not correct).`
-

Display

- Do convert non-Unicode data to Unicode before they are displayed.
- Don't display Punycoded text to the user unless it benefits the user in some way.

Unicode

- Do 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
- Don't ever use UTF-7 or UTF-32.
- Do 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'
- Do use Unicode in cookies so they can be read correctly by applications.
- Do use IDNA 2008 Protocol [RFC5891] and Tables [RFC5892] documents. Don't use IDNA 2003.
- Do not assume that external APIs can consume data which has been NFKC converted.
- Do maintain IDNA and Unicode tables that are consistent with regard to versions, i.e., unless the application actually executes the classification rules in the Tables document [RFC5892], its IDNA tables must be derived from the version of Unicode that is supported more generally on the system. As with registration, the tables need not reflect the latest version of Unicode, but they must be consistent.
- Do validate the characters in labels only to the extent of determining that the U-label does not contain "DISALLOWED" code points or code points that are unassigned in its version of Unicode.
- Do limit validation of labels itself to a small number of whole-label rules.
  - No leading combining marks
  - Bidirectional conditions are met if right-to-left characters appear
  - Any contextual rules that are associated with joiner characters (and CONTEXTJ characters more generally) are tested.
- Don't use 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.

Linkification

- If a string resembling a domain name contains the ideographic full stop character "。" (U+3002), do accept it and transform it to ".".

General

- Do 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."
- Do ensure that the product or feature handles numbers correctly. For example, ASCII numerals and Asian ideographic number representations should all be treated as numbers.
- Do 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.
- Do 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
- Do look for potential IRI paths in unexpected places
  - Single-label machine names regardless of loaded system codepage
  - Fully-qualified machine names regardless of loaded system codepage
- Do use GB18030 for Chinese language support.
- Do restrict the code points allowed when generating new domain names and email addresses.
  - All products that use email addresses must accept internationalized email addresses, allowing characters > U+007f. That is, no characters > U+007f are disallowed.
  - However, an app or service need not allow all of these characters when a user creates a new IDN or email address. Use only this allowed list of characters for IDN: http://unicode.org/reports/tr36/idn-chars.txt
  - Some likely security and accessibility concerns can be mitigated by preventing certain IDNs or email addresses from being created in the first place. (NOTE: Postel's Law would still require software to accept such strings if presented.)
- It is important to note that Universal Acceptance cannot always be measured through automated test cases alone. For example, testing how an app or protocol handles network resource may not always be possible and sometimes it is best to verify the compliance through functional spec review and design review.
- Don't automatically assume that because a component does not directly call name-resolution APIs, or directly use email addresses, it does not mean that it is not affected the by them.
  - Understand how network names are obtained by the component; it is not always through user interaction. Following are some examples on how the component can get a network name:
    - Group Policy
    - LDAP query
    - Configuration files
    - Registry
    - Transferred to/from another component/feature.
- Do perform code reviews to avoid buffer overflow attacks.
  - In Unicode, strings may expand in casing: Fluß → FLUSS → fluss. When doing character conversion, text may grow or shrink substantially.

Authoritative sources for domain names – TO DO

There are a few options for the authoritative list of TLDs. The first option would be the DNS root zone itself. It is DNSSEC signed, so the list is properly authenticated. You can obtain the root zone at http://www.internic.net/domain/root.zone or from https://www.dns.icann.org/index.html%3Fp=196.html.

Public suffix list

-

Other Challenges

General

- In some applications IDNs are encoded in Punycode as per IDNA if the name is identified as an Internet name, but UTF-8 is used if the name is identified as an intranet name.
- 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).
- Some applications that do IDNA (e.g., IE7+) break for non-DNS protocols, and could affect accessing resources using non-DNS protocols.
- When allowing a user to generate a domain name or email address, consider avoid using visually confusing characters to prevent homograph attacks. 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.

Linkification challenges

Even when applications fully support new gTLDs, IDN and EAI linkification may not happen as expected by a user. In some cases invalid links may even be created. Here are some examples of typical linkification in existing applications:

Example string	**Likely result**
apple.com	No change
www.apple.com	Link to www.apple.com
http://apple.com	Link to http://apple.com
http:apple.com	No change
http://.com	Link to http://.com
apple.news	No change
www.apple.news	Link to www.apple.news
http://apple.news	Link to http://apple.news
apple.photography	No change
www.apple.photography	Link to www.apple.photography
http://apple.photography	Link to http://apple.photography
http://.photography	Link to http://.photography
田中.com	No change
www.田中.com	Link to www.田中.com
http://田中.com	Link to http://田中.com
田中。com	No change
http://田中。com	No change
español.com	No change

www.español.com	Link to www.español.com
http://español.com	Link to http://xn--espaol-zwa.com/
http:///español.com	Link to http://xn--espaol-zwa.com/

Complex Scripts – TO DO

Right to left languages and Unicode conformance

-

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

-

Topics for potential proposals to ecosystem, ICANN, IETF – TO DO

- 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 encourage "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)?
- Define IDN-style email
- UTR#36 – does not discuss structured text?
- Structural separators like – and numerals – define behavior esp. for bidirectional
- Do tooltips correctly show the TLD for mixed RTL/LTR? (show visual example) – best practices?

Glossary and other resources

Cross-link to RFCs – TO DO

- 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
- .NET Framework 4.5 and higher
- URIs - http://tools.ietf.org/html/rfc3986
- http://www.internic.net/faqs/authoritative-dns.html

Acknowledgements

Scratchpad

A domain name is a dotted text string used as a human-friendly technical identifier for computers and networks on the Internet, e.g. “www.domain.tld”. Each dot represents a “level” in the Domain Name System (DNS) hierarchy. A Top-Level Domain (TLD) is often called the “suffix” at the end of a domain name (“.tld” in the above example).

The set of TLDs change over time:

The characters allowed expands

- ✓ **DO ask and avoid over simplification of assumptions.** ICANN and the community is happy to help provide advice to software developers and implementers on what is needed. Contact us at: tld-acceptance@icann.org and/or join the Universal Acceptance discussion at [UASG link].
- ✓ The primary method of correctly checking if a domain is valid is to use the DNS. If the application has access to the Internet (most applications do), the best way is simply to perform a DNS query. This ensures the most accurate and up-to-date data is returned from the most authoritative source – the DNS itself.
- ✓ **More information**
- ✓
- ✓ To learn more about the effort, visit us at: http://www.icann.org/universalacceptance
- ✓
- ✓ To share your ideas and suggestions on the topic email us at: tld-acceptance@icann.org.

“Domain name slot” is defined (RFC5890) to be a protocol element or function argument or a return value (etc.) explicitly defined for a domain name

- QNAME of a DNS query
- Name argument of gethostname(), getaddrinfo() standard C libraries
- Part of an email addr after the “@” symbol
- SMTP parameter to the MAIL or RCPT commands
- Hoist portion of URI/IRI in the “src” attribute of an HTML “<IMG> tag

Labels are considered, and characters ordered, in the order which they are transmitted “on the wire”

U-Labels can be 252 chars – buffer overflow

It is suggested that UI indicate when mixed scripts contain confusable characters

RFC5890/4.6 ZWJ and ZWNJ – careful

IDNA applies only to domain names in the NAME and RDATA fields of DNS resource records whose class is IN – see RFC 1035

Normalization for C (NFC)

Must not contain characters that appear in the DISALLOWED and UNASSIGNED lists (RFC5892)

Bidi RFC5993

5893 – if a string contains a RandALCat character, it MUST be the first char and a RandALCat must also be the last char in the string

Directionality is by context: "ABC.abc is displayed as CBA.abc" – network order and display order.

"Bidi domain name" contains at least one RTL label.

RFC5893 /2. The "bidi rule"

DNS clients and registries are subject to differences – registries urged to register only exact valid A-Labels, whereas clients may transform a string from input → valid a-label

Not all apps are required to support IDNA

App-specific mapping, folding, display conversions will result in CPE during input, display and APIs between apps

Verify Latin-based mathematical character disallowed – to-do

Only strings NOT transformed by nfkc are valid

What is "case folding"

Context – ligatures may make sense in one language versus another (e.g. ae ligature in Norwegians and Swedish

Latin numerals in 2008 versus 2003

DNS lookups as A labels is probably less ambiguous

Idna 2003 expects multiple mapping (input method and nameprep) could be confused in APIs

Convert inputs to utf8 codepoints ASAP

Registry tables not guaranteed to be updated to latest Unicode

Due to 2008 which now supports ZWJ and ZWNJ (and 2003 ignored them) so same name maps to different a-label

What is "IESG statement"?

Context is applied in the app, but never in the DNS

"IDNA language character registry" for context – resource where? "IANA Script Registry"

Don't mix languages within a script – keyboard and user confusion – may be allowed by user generation rules, but not a best practice?

http://www.w3.org/International/wiki/Case_folding