Introduction to Universal Acceptance

# About this document

The Internet’s technologies, including its naming components, continually evolve and change. In recent years, new top-level domains (TLDs), some with traditional ASCII characters and some with non-ASCII characters (Internationalised Domain Names, or IDNs), have been approved by ICANN. Examples include .nyc, .संगठन, .eco, and .католик. However, many applications and services have not been updated to manage this new full range of TLDs. Also, Internet email standards now allow non-ASCII characters in email addresses. Until it is upgraded, software will not properly handle these domains and addresses. This affects the user experience. For example:

* Valid email addresses are not recognized or accepted.
* Domain names are mistakenly treated as search terms in the address bar of the browser.

Until software recognizes and can process the new domains and addresses—a state known as **Universal Acceptance**—it will not be possible to provide a consistent and positive experience for Internet users. This document provides a broad introduction to Universal Acceptance to assist in the development of Universal Acceptance-ready software.

## Target Audience

This document is intended to introduce Universal Acceptance to a technical audience (developers, managers, and operators) that may be familiar with some aspects of Internet technology but not necessarily the details of how IDNs and other comparatively recent changes to domain names and email addresses affect the way in which they should be accepted, validated, stored, processed, and displayed. It represents a starting point for people with diverse technical backgrounds to begin their exploration of Universal Acceptance.

## Document Structure

**Part 1**: Background concepts such as what is a Domain Name, ASCII and Unicode, IDN formats, Email Address Internationalization, and other basic concepts.

**Part 2:** The five criteria of Universal Acceptance as well as the good practices for each of these criteria. Also contains user scenarios and nonconformance practices to Universal Acceptance, technical requirements, and current challenges.

**Part 3**: Advanced topics such as right-to-left scripts, the Bidi algorithm, Normalization, and Case Folding.

**Part 4:** Glossary and online resources.

**Need more information?**

The Universal Acceptance Steering Group (UASG) and community are available to provide advice to software developers and implementers.

**👍 Contact us** to share your ideas and suggestions on the topic at info@uasg.tech

**👍 Join the Universal Acceptance** discussion list at<http://tinyurl.com/ua-discuss>

**👍** To **learn more** about the effort, visit<http://www.icann.org/universalacceptance>

# Part 1: Background Concepts

## Domain Names

A domain name is a human-friendly identifier for computers and networks on the Internet. It is customarily represented as a sequence of text labels separated by “dots” (the period or full-stop punctuation mark), for example www.example.tld. Each label represents a level in the Domain Name System (DNS) hierarchy.

At the highest level or “root” of the hierarchy are Top-Level Domain (TLD) labels such as com, jp, and বাংলা, which appear at the end of a domain name. Because they appear at the end, TLDs are sometimes called “suffixes.”

Proceeding down the DNS hierarchy from the root, the next label identifies a subdomain of the TLD, commonly called a second-level domain; the label after that identifies a subdomain of the second-level domain, commonly called a third-level domain; and so on, with each of the labels separated from its neighbor by a dot. For example, a domain name with three levels might look like this:



### Country code TLDs (ccTLDs)

Some TLDs are delegated to specific countries or territories. These are called Country Code TLDs (ccTLDs). In the past all ccTLDs were two letters, matching the ISO 3166 two-letter code assigned to the country or territory by the International Organization for Standardization. Since 2010 there have also been internationalized ccTLDs that represent the name of a country or territory in that country or territory’s own script.

### Generic TLDs (gTLDs)

Most TLDs that are not ccTLDs are so-called generic top level domains (gTLDs) open to anyone, or open to members of a defined community. These include the familiar com, net, and org as well as more recent additions.

Starting in 2006, the Internet Corporation for Assigned Names and Numbers (ICANN), the organization responsible for the creation and maintenance of TLD assignments, has approved the creation of a large number of new gTLDs. These new TLDs can represent brands, communities of interest, geographic communities (cities, regions), and more generic concepts.

## Domain Name Internationalization

Domain names were originally limited to a subset of ASCII characters (letters a-z, digits 0-9, and the hyphen “-“). Since the earliest .com registration, symbolics.com, in 1985, the number and characteristics of domain names have expanded to reflect the needs of the ever-increasing global use of the Internet as a communal resource. Today, the majority of Internet users are non-English speakers. However, the dominant language used on the Internet is English. To help with the internationalization of the Internet, in 2003 the Internet Engineering Task Force (IETF) started releasing standards providing technical guidelines for the deployment of Internationalized Domain Names (IDNs) through a translation mechanism to support non-ASCII representations of domain names in any Unicode-supported script (e.g., 普遍接受-测试.世界, ua-test.كاثوليك,etc.).

The ICANN Board of Directors approved the process to introduce new IDN Country Code Top Domain Names (ccTLDs) in October 2009, with the first IDN ccTLDs added to the root zone in May 2010. In June 2011, the Board approved and authorized the launch of the new Generic Top Level Domain (gTLD) program, which included new ASCII as well as IDN TLDs. The first batch of TLDs from this program was added to the root zone in 2013.

## The Need for Universal Acceptance

A decade after the IETF released its IDN-related guidelines, and due to the ICANN New TLD Program, more than one thousand new TLDs are now active. However, much software and many applications are still not updated to handle these new TLDs—they are not **Universal Acceptance-ready**. This causes problems for Internet users, including but not limited to those whose languages are written in scripts that include non-ASCII characters.

To keep pace with this new TLD world, new software must be built and old software and applications must be updated. The state of successfully complying with this new world of TLDs is called **Universal Acceptance**.

Universal Acceptance is the state in which all valid domain names and email addresses are **accepted**, **validated**, **stored**, **processed**, and **displayed** correctly and consistently by applications, devices, and systems. For example, every valid web address resolves to the expected resource on the correct website and every valid email address results in mail delivery to the expected recipient. Due to the rapidly changing domain name landscape, many systems do not recognize or appropriately process new domain names—because they may be IDNs, because the software is not aware of the newly released TLD, or because the TLD name is longer than older TLDs. The same is true for email addresses that incorporate these new TLDs.

The **Universal Acceptance Steering Group (UASG),** a community-led, industry-wide initiative that is supported by ICANN, is working on creating awareness, identifying and resolving problems associated with Universal Acceptance of domain names to help ensure a consistent and positive experience for Internet users globally.

## U-labels and A-labels

Domain names that use non-ASCII characters are called Internationalized Domain Names (IDNs). The internationalized portion of a domain name can be in any label – not just the TLD but also the other labels.

Since the DNS itself previously only used ASCII, it was necessary to create an additional encoding to allow non-ASCII Unicode code points to be represented as ASCII strings. The algorithm that implements this Unicode-to-ASCII encoding is called **Punycode;** the output strings are called **A-Labels**. A-Labels can be distinguished from an ordinary ASCII label because they always start with the following four characters:

**xn**--

These characters are called the **ACE prefix**.[[1]](#footnote-0)

The Punycode transformation is reversible: it can transform from Unicode to an A-Label and also from an A-label back to a string of Unicode characters (known as a U-Label).

The only standard use of the Punycode algorithm is for expressing internationalized domains. While one could hypothetically encode other UTF-8 strings using Punycode, that would be non-standard and would not interoperate with other systems.

**Examples of (imaginary) IDNs**

|  |  |
| --- | --- |
| U-label version | A-label version |
| example.みんな | example.xn--q9jyb4c |
| 大坂.info | xn--uesx7b.info |
| みんな.大坂 | xn--q9jyb4c.xn--uesx7b |

## Email Address Internationalization

Email addresses contain two parts:

1. A local part (before the “@” character)

2. A domain part (after the “@” character)

In an internationalized email address, the domain part can contain any domain name, including those with new TLDs, and can contain Unicode U-labels. The local part is not a domain name and can in principle contain nearly any Unicode character, although in practice mail systems will limit the characters used in their mailbox names.

The abbreviation EAI is often used to describe any mail that uses internationalized addresses, or the internationalized addresses themselves.

|  |  |  |
| --- | --- | --- |
| **Examples of (imaginary) EAI Addresses:**

|  |  |
| --- | --- |
| user@example.みんなuser@大坂.info**用戶**@example.lawyer | Uses IDN TLDUses IDN 2nd level domainUses Unicode local part and new gTLD |

**Example of Right to Left text in an EAI address:** **LTR RTL**Dynamic Link Generation (Linkification)Modern software, such as popular word processing or spreadsheet applications, sometimes allows a user to create a hyperlink simply by typing in a string that looks like a web address, email address 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 addresses or network paths, and not only some. Accurate linkification is difficult and depends on the context of the text (for example, in some languages “www” doesn’t indicate a web address) so it is not further addressed here.The Dynamic Nature of the Root Zone RegistryThe DNS is a large distributed database divided into sections called *zones*. The section that contains all of the TLDs is called the *root zone* because it is conceptually at the root of the tree of DNS names. All DNS zones, including the root zone, are updated as required. As new TLDs are added or old TLDs are retired, their names are added to and removed from the root zone.This means that any fixed list of TLDs, such as a list stored in an application or in a file, will eventually and inevitably become out of date. To reliably validate the TLD in a domain name, either software can check it on-line with a DNS query, or if it uses a file, it needs to refresh the file on a regular basis. These are both described in more detail later. |

# Part 2: Universal Acceptance in Action

## Five Criteria of Universal Acceptance

As described in the section, Universal Acceptance is the state where all valid domain names and email addresses are **accepted, validated, stored,****processed** and **displayed** correctly and consistently by all applications, devices and systems*.* These five criteria are described below.

|  |  |
| --- | --- |
| **1. Accept[[2]](#footnote-1)**  | ***Accepting* occurs whenever an email address or a domain name is received as a string of characters from a user interface, from a file, or from an API used by a software application or online service.**Applications and services allow domain names and email addresses to be:* Entered via user interfaces, or
* Received from other applications and services via APIs.
 |
| **2. Validate[[3]](#footnote-2)**  | ***Validation* may occur in many places whenever an email address or a domain name is either received or emitted as a string of characters by an application or online service.** Validation is intended to ensure that the entered information is either valid or at least definitely not invalid. Validation ensures the information has the correct syntax, and may make other checks. For domain names and email addresses, many programmers have been using some heuristics (for example, checking that a TLD is within length limits, or that the characters are from the ASCII character set). However, these heuristics are no longer applicable because the Internet is changing:* Domain names and email addresses can now include non-ASCII Unicode characters
* The list of TLDs is changing
* Any label in a domain name, including the TLD label, can be up to 63 characters long.[[4]](#footnote-3)

It remains possible to validate TLDs using other techniques, as described later. |
| **3. Store**  | ***Storage* occurs when an email address or a domain name is stored as a string of characters in a database or file used by a software application or online service and later retrieved by the same or other software applications.**Applications and services might require long-term and/or transient storage of domain names and email addresses. Regardless of the lifetime of the data, it must be stored in:* RFC-defined formats, OR
* Alternate formats that can be translated to and from RFC-defined formats (this is less desirable)

Although Unicode in DNS names and email addresses are stored as UTF-8, other formats may be encountered in legacy code. See the “Good Practices” section below. |
| **4. Process[[5]](#footnote-4)**  | ***Processing* occurs when an email address or a domain name is used by an application or service to perform an activity such as searching or sorting a list, or transformed into an alternate format (such as turning U-labels into A-labels).**Additional validation may occur during processing. The ways in which email addresses and domain names may be processed is limited only by the imagination of application developers, but it is important not to make assumptions (e.g., that email addressed to *pākehā@tetaurawhiri.govt.nz* is intended for a person in New Zealand) that depend on policies outside of the DNS. |
| **5. Display**  | **The *Display* action occurs when an email address or a domain name is rendered within a user interface.**Displaying domain names and email addresses is usually (but not always) straightforward when the scripts used in the name or address are supported in the underlying operating system and when the strings are stored in Unicode.[[6]](#footnote-5) If these conditions are not met, application-specific transformations may be required. Further, even if the underlying operating system does support the strings display can be complicated if for example RTL and LTR scripts are mixed, and/or if the overall directionality of text is unclear. |

## User Scenarios

The examples and definitions above may give the impression that Universal Acceptance is only about computer systems and online services. The reality, however, is that it’s also about the people using those systems and services.

Below are some examples of activities that require Universal Acceptance:

|  |  |
| --- | --- |
| **Registering a new TLD** | An organization adopts a “brand” TLD to offer its customers a differentiated customer experience by providing email addresses in the format, customername @example.brand.Universal Acceptance means:* Web sites and applications accept these “@example.brand” email addresses as they would with older TLDs such as .com, .net, .org.
 |
| **Accessing a gTLD** | A user accesses a website, whose domain name contains a new TLD, by typing an address into a browser or clicking a link in a document.Universal Acceptance means:* Even though the TLD is new, the user’s browser displays the web address in its native form and accesses the site as the user expects. The browser does not display domain names as A-labels to the user unless it benefits the user in some way.
 |
| **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.Universal Acceptance means:* Even though the domain used in the email address is new, the bank or airline site accepts the address the same way in which it accepts an address in an older TLD such as .biz or .eu.
 |
| **Accessing an IDN** | A user accesses an IDN URL, by typing the URL into a browser or clicking a link in a document.Universal Acceptance means:· Even if the domain name contains characters different than the language settings on the user’s computer, any browser the user wishes to use will display the web address as expected and access the site successfully. |
| **Using an internationalized email address for email** | A user has acquired a new email address which includes non-ASCII characters in the domain name (e.g. īnfo@普遍接受-测试.世界).Universal Acceptance means:* The user can send to and receive from any email address and using any email client.
 |
| **Using an internationalized email address as an online identity** | A user acquires a non-ASCII email address, and uses this email address as their identity for accessing their bank and airline loyalty accounts.Universal Acceptance means:* The bank or airline site accepts the new identity exactly as if it were any other email identity.
 |
| **Dynamically creating a Hyperlink in an Application** | A user types a web address into a document or email message.Universal Acceptance means:* The rules used by the application to automatically generate a hyperlink are the same if the address is non-ASCII or contains a new TLD.
 |
| **Developing an Application** | A developer writes an app that accesses web resources.Universal Acceptance means:* The tools used by the developers include libraries that enable Universal Acceptance by supporting new TLDs and IDNs.
 |

## Nonconformance to Universal Acceptance Practices

The following are considered to be **poor practice**:

|  |  |
| --- | --- |
| **✖** | Displaying A-labels to the user without a corresponding user benefit, such as to show the mapping between a U-label and a A-label. |
| **✖** | Requiring a user to enter A-labels when signing up for a new email address or requiring a user to enter A-labels when signing up for a new hosted domain. |
| **✖** | Validating the syntax of domain name or email address using out of date criteria or non-authoritative online domain name resources. |
| **✖** | Using an outdated list of TLDs even though new TLDs are regularly being added and deleted. |
| **✖** | Exposing internal use of A-labels to users.For example, converting domains in EAI addresses to A-labels when replying to an EAI user.  |
| **✖** | Treating some domain names as search terms rather than as domain names because the application does not recognize them as such. |
| **✖** | Setting spam blockers to automatically block entire (new) TLDs without evidence of abuse. |

## Technical Requirements for UA Readiness

For an application or web site to be UA ready, it must meet a variety of requirements.

## High level Requirements

An application or service that supports universal acceptance (UA):

**1.** **Supports all domain names regardless of length or character set.**

See [RFC 5892](https://tools.ietf.org/html/rfc5892).

**2.** **Allows all of the character sets that are valid for domain names and email addresses.**

That is, accept Unicode code points as well as ASCII.

**3.** **Can correctly render all code points in Unicode strings.**

See [RFC 3490](https://www.ietf.org/rfc/rfc3490.txt). Note that Unicode regularly adds new code points, so this is a moving target.

**4.** **Can correctly render right-to-left (RTL) strings such as those in Arabic and Hebrew.**

For information about RTL scripts, see [RFC 5893](https://tools.ietf.org/html/rfc5893).

**5.** **Can communicate data between applications and services in UTF-8 and, where needed, other encodings that can be converted to and from UTF-8.**

For information about UTF-8, see [RFC 3629](https://tools.ietf.org/html/rfc3629).

**6.** **Offers public and private APIs that support UTF-8.**

Private APIs apply only to inter-service calls by the same vendor.

**7.** **Treats EAI addresses correctly.**

In particular, doesn’t convert IDNs in addresses to A-labels.

**8.** **Can send email to and receive from recipients regardless of domain name or character set.**

 See [RFC 6530](https://tools.ietf.org/html/rfc6530).

**9.** **Stores user data in formats that support Unicode and is convertible to and from UTF-8.**

 Such conversions would be visible only to the operator of the product or service.

**10.** **Supports all top-level domain names in the authoritative ICANN TLD list regardless of length or character set.**

See the authoritative list at <https://data.iana.org/TLD/>.

## Developer Considerations

Since many existing software systems contain hardcoded assumptions about domains and email addresses, code changes may be required to recognize IDNs, new TLDs, and EAI mail addresses. This section discusses how developers can incorporate code changes that will enable Universal Acceptance.

### Designing software for Compatibility and Flexibility

In [RFC 793](https://tools.ietf.org/html/rfc793), [Jon Postel](https://en.wikipedia.org/wiki/Jon_Postel) described the Robustness Principle, as an implementation guideline for the then-new TCP. In [computing](https://en.wikipedia.org/wiki/Computing), the Robustness Principle is a general design guideline for software:

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

That is, be conservative in what you send: in any area in which a specification might be ambiguous or unclear, avoid anything that could surprise others. On the other hand, when receiving, accept anything that is plausibly valid. This does *not* mean changing code to work around clear mistakes in other implementations, though, since that leads to an undocumented and undebuggable mess.

### Good Practices for Developing and Updating Software to Achieve UA-Readiness

|  |
| --- |
| **Accept** |
| **✔** | Display names as Unicode whenever possible.Users should be allowed, but not required, to enter domain names as A-labels rather than U-labels. However, U-labels should be shown by default, with A-labels shown to the user only when it provides a benefit. |
| **!** | **Don’t** generate EAI addresses with A-labels, but **do** be able to handle them if presented by someone else’s software. |
| **✔** | Any user interface element requiring a user to type a domain name or email address must accept long names. ASCII domain names can have up to 63 characters in each label, and can be in total up to 253 bytes. UTF-8 labels can be much longer than ASCII labels and the total length can be up to 670 bytes. Remember that the UTF-8 code for most Unicode code points takes more than one byte.· See [RFC 1035](https://tools.ietf.org/html/rfc1035). |

###

|  |
| --- |
| **Validate** |
| **✔** | Validate only as appropriate**.****Validate only if it is required for the operation of the application or service.** This is the most reliable way to ensure that all valid domain names are accepted into your systems.  |
| **✔** | Recognize that syntactically correct inputs may represent domain names or email addresses that are currently in use on the Internet. That may or may not be valid depending on the application. |
| **!** | When you validate, consider the following:· Verify the TLD portion of a domain name against an authoritative table. IANA publishes the list of top level domains at:* <http://data.iana.org/TLD/tlds-alpha-by-domain.txt>
* See also:<https://www.icann.org/en/system/files/files/sac-070-en.pdf>

· Query the domain name against the DNS* The GETDNS API (<http://getdnsapi.net/>) is a highly portable way to query the DNS
* Most operating systems also have a native DNS query API.

· Require repeated entry of an email address to detect typing errors· Validate the characters in labels by checking that each label follows the IDNA 2008 rules.* See [RFC 5892](https://tools.ietf.org/html/rfc5892)

· Limit validation of labels itself to a small number of whole-label rules defined in the RFCs* See [RFC 5894](https://tools.ietf.org/html/rfc5894)

· Ensure that the product or feature handles numbers correctly* For example: Arabic-Indic digit characters should be treated as numbers in numeric input fields, as well as ASCII digits.
* Note that Arabic-Indic digits are valid in U-labels, but are not considered equivalent to ASCII digits in that context.
 |

###

|  |
| --- |
| **Store** |
| **✔** | Applications and services should support the most current Unicode standards. |
| **✔** | Information should be stored in the UTF-8 format whenever possible.Some systems may require legacy support for UTF-16 as well, but generally UTF-8 is preferred. UTF-7 is obsolete, and UTF-32 is too bulky for file storage.The strings should be normalized when appropriate (some normalization can in some context result in loss of information). |
| **!** | Consider all end-to-end scenarios before converting between A-Labels and U-Labels when storing.In new applications, it is better to maintain only U-Labels in a file or database, because it simplifies searching, sorting, and presentation. However, conversion may have implications when interoperating with older, non-Unicode-enabled applications and services. |
| **✔** | Tag email addresses and domain names as such in storage for easier access.Filing email addresses and domain names in the “author” field of a document or “contact info” in a log file have led to the loss of the original address. |
| **✔** | Regardless of the way addresses and domain names are stored, you must be able to match strings in multiple formats.For example, a search for example.みんなshould also find example.xn--q9jyb4c. |

###

|  |
| --- |
| **Process** |
| **✔** | Ensure all web server and MIME mail responses have UTF-8 specified in the content type. |
| **✔** | Specify UTF-8 in the web server http header.· It is important to ensure that the encoding is specified on every response |
| **!** | Consider context before converting A-Labels to U-Labels and vice versa during processing.It is desirable to maintain only U-Labels in a file or database as it simplifies searching and sorting. However, conversion may have implications when interoperating with older, non-Unicode-enabled applications and services.  |
| **✔** | Ensure that the product or feature handles sort order, searches, and collation according to locale and language specifications, and that it addresses multilanguage searching and sorting. |
| **✖** | Don’t use percent encoding for labels in domain names:· example.みんな is correct· example.%E3%81%BF%E3%82%93%E3%81%AA is not correct |
| **✔** | Since the Unicode standard is continually expanding, code points not defined when the application or service was created should be checked to ensure they don’t produce erroneous or confusing outputMissing fonts in the underlying operating system may result in non-displayable characters (frequently a small box is used to represent these), but this situation should not result in a crash or error message. |
| **✔** | Use supported Unicode-enabled APIs. |
| **✔** | Use the latest Internationalized Domain Names in Applications (IDNA 2008) Protocol and Tables documents for IDNs:·  [RFC 5891](https://tools.ietf.org/html/rfc5891)·  [RFC 5892](https://tools.ietf.org/html/rfc5892) |
| **✔** | Process text in UTF-8 format wherever possible. |
| **✔** | Coordinate upgrades of applications and the services they depend on.If theserver is Unicode and client is non-Unicode, or vice versa, the data will need to be converted in every transaction, which is error prone and can be slow. |
| **✔** | When doing character transformation, text strings may grow or shrink substantially. Each UTF-8 code point may be from 1 to 4 bytes, and in some cases a single character in another encoding may correspond to several UTF-8 code points or vice versa. |

###

|  |
| --- |
| **Display** |
| **✔** | Display all Unicode code points that are supported by the underlying operating system.Modern operating systems all have Unicode support, but their rendering engines are not always correct for all scripts and languages. Provide character rendering in applications only when correct rendering is not available from the target operating system(s). |
| **✔** | When developing an app or a service consider the languages supported and make sure operating systems and applications cover those languages. |
| **✔** | Convert A-labels to U-labels before display.For example, the end user should see “example.みんな” as opposed to “example.xn--q9jyb4c”. (This conversion is an example of UA-ready processing). |
| **✔** | Display domain names as U-labels by default.Display A-labels to the user only when it provides a benefit. |
| **!** | Be aware that mixed-script domain names are possible.· Some Unicode characters may look the same to the human eye, but different to computers, for example, latin O, cyrillic O, and greek omicron O.· Mixed-script strings are common in closely related scripts (e.g., Japanese Kanji, Katakana, Hiragana, and Romaji.) Otherwise mixed scripts may be intended for malicious purposes, such as phishing. Use Unicode Technical Standard #39, “Unicode Security Mechanisms[[7]](#footnote-6) to check that the scripts in a Unicode sequence follow good practice.· If the user interface calls the strings to the user’s attention, be sure that it does so in a way which is not prejudicial to users of non-Latin scriptsLearn more about Unicode Security Considerations at:<http://unicode.org/reports/tr36> |
| **✔** | Be aware of unassigned and disallowed characters for domain names.· See [RFC 5892](https://tools.ietf.org/rfc/rfc5892.txt) |

###

|  |
| --- |
| **Unicode** |
| **✔** | Use supported Unicode-enabled APIs. |
| **✖** | Use standard well debugged APIs for:* String format conversions
* Determining which script comprises a string
* Determining if a string contains a mix of scripts
* Unicode normalization/decomposition
 |
| **✖** | Don’t use UTF-7, and limit the use of UTF-32.* UTF-7 is obsolete.
* UTF-32 uses four bytes for each code point. Since each code point takes the same amount of space and can be directly indexed in arrays, it’s convenient to use within program code, but may be too bulky for storage in files and databases.
 |
| **✖** | Don’t use UTF-16 except where it is explicitly required (as in certain Windows APIs, and Javascript applications).In UTF-16, 16 bits can only represent characters from 0x0 to 0xFFFF. Values above this range (0x10000 to 0x10FFFF) use pairs of pseudo-characters known as surrogates. If handling of surrogate pairs is not thoroughly tested, it may lead to tricky bugs and potential security holes. |
| **✔** | Use UTF-8 in cookies so they can be read correctly by applications. |
| **✔** | Use IDNA 2008 Protocol and Tables documents:· [RFC 5891](http://tools.ietf.org/html/rfc5891)·  [RFC 5892](http://tools.ietf.org/html/rfc5892) |
| **✖** | Do not use IDNA 2003 which has been superseded by IDNA 2008 and is because of this obsolete. |
| **!** | Maintain IDNA and Unicode tables that are consistent with regard to versions.For example, unless the application actually executes the classification rules in the Tables document to interpret code points as entered ([RFC 5892](https://tools.ietf.org/html/rfc5892)), its IDNA tables must be derived from the version of Unicode that is supported on the system. The tables do not need to reflect the latest version of Unicode, but they must be consistent. |
| **✔** | Validate labels using IDNA 2008 whole-label rules* In some contexts further validation may be appropriate, e.g., if the application knows what scripts are allowed in the domain names it uses.
 |

###

|  |
| --- |
| **General** |
| **✔** | Use authoritative resources to validate domain names.Do not make heuristic assumptions, such as “all TLDs are 6 characters or shorter”.  |
| **✔** | Ensure that the product or feature handles numbers correctly.For example, ASCII numerals and Asian ideographic number representations should all be treated as numbers in numeric contexts. |
| **!** | Look for mail addresses that may be EAI 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 |
| **!** | 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 most UTF-8 printing characters in the local part. However, an app or service need not allow all of these characters when a user creates a new IDN or EAI address. Preventing certain IDNs or email addresses from being created in the first place can mitigate some likely security and accessibility concerns. (NOTE: Good practice would still require software to accept such strings if presented.) |
| **!** | Be aware 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 assume that because a component does not directly call name-resolution APIs, or directly use email addresses, it does not affect it.Understand how domain names are obtained by the component; it is not always through user interaction. The following are some examples on how the component can get a domain name:· Group policy· LDAP query· Configuration files· Windows Registry· Transferred to/from another component/feature |
| **✔** | Perform code reviews to avoid buffer overflow attacks.· In Unicode, strings may expand or shrink when case folded or normalized· When doing character conversion, text may grow or shrink substantially |

##

|  |
| --- |
| **Other challenges** |
| **Mechanism to detect and convert charsets** | Some older email applications used local character encodings and did not have a way to detect and converting text to and from UTF-8 as needed. This was especially true for the email headers (TO, CC, BCC, Subject).  |
| **Managing multiple email addresses as a single user identity** | When a user has 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 applications may still treat the addresses as different identities. |

### Authoritative Sources for Domain Names: DNS Root Zone and IANA lists

There are a few options for the authoritative list of TLDs. The first option is the DNS root zone itself. It is DNSSEC-signed, so its contents can be authenticated by a DNSSEC aware name server, although its contents are fairly hard to parse as a text file. Another is the text file of TLDs that IANA publishes, one TLD per line in alphabetical order. These files are on https web servers, so it is good practice to check that the site’s TLS certificate is valid when downloading it to be sure you’re getting the right file.

You can obtain the list of TLDs from either of the following links:

* <https://www.internic.net/domain/root.zone> (root zone file)
* <https://data.iana.org/TLD/tlds-alpha-by-domain.txt> (text TLD file)

### Email with IDNs and Why It Is Not the Same as EAI

EAI mail prefers UTF-8 domain names; ASCII coded A-Labels are discouraged. Some mail systems have made partial provisions for email addresses that include IDNs rather than provide full EAI support. Because IDNs can be represented as ASCII A-labels, some existing software allows the IDNs in an email address to be represented in ASCII or Unicode. For example, some software will treat these two IDN addresses equivalently for all purposes (sending, receiving, and searching):

|  |
| --- |
| **Not all software will treat these two addresses as equivalent, even though the domains are different representations of the same name.** |

 user@example.みんな **=** user@example.xn--q9jyb4c

However, some software will not treat these addresses as equivalent, even though are both valid, because it does not convert an A-label (i.e. “xn--q9jyb4c”) into its U-label equivalent (i.e. “みんな”) before comparing. This can result in unpredictable user experience. The user experience may become especially confusing if some software converts U-labels into A-labels for “compatibility”; as messages are replied-to or forwarded, the addresses which are visibly different to a user, or which fail to search and sort as expected, may increase.

In the example below, some software may attempt to convert the local part of the email address using Punycode, i.e. same conversion as being used between A-labels and U-labels. This sort of conversion is invalid and will create invalid, undeliverable addresses.

|  |
| --- |
| **Never try to convert the local part of an email address into a different form****✔** 用戶@example.みんな**✖** xn--youq53b@example.xn--q9jyb4c |

Robust UA-ready software and services should be able to handle and treat all these formats correctly, and should be able to handle both UTF-8 local parts and UTF-8 U-labels in addresses, while also accepting A-labels in addresses for backward compatibility.

### Linkification and Its Challenges

Modern software sometimes allows a user to automatically create a hyperlink 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 application recognizes “www.” as an initial label or “.org” as a TLD.

Linkification is the action whereby an application accepts a string and dynamically determines whether it should create a hyperlink to an Internet Location (http:// or https://) or an email address (mailto:). Linkification, if it happens, should work consistently for all well-formed web addresses, email names, or network paths.

Linkification uses algorithms and rules created by software developers to determine whether or not a string should be interpreted as a link. Related to this is how people can identify a string as a domain name. While browsers, email clients, and word processors are obvious places, there are many more applications that make these decisions.

Good Practice Recommendations

1. Attempt to linkify based on explicit protocol prefixes (e.g. “https://”, ftp://”, “mailto:”) but only complete the action if the rest of the string is well formed

|  |  |
| --- | --- |
| **Example String** | **Expected Behavior/ Result** |
| example.com | No linkification because protocol is absent and not inferred. |
| http://example.com | Create hyperlink because protocol is explicit |
| http:example.com | No linkification because of bad syntax (missing //) |
| <http://example.a> | No linkification because “a” is not a TLD.  |
| <http://example..ab> | No linkification because of bad syntax (consecutive dots) |
| http://普遍接受-测试.世界 | Create hyperlink because protocol is explicit. |

2. Attempt to linkify based on implicit protocol prefixes (e.g. “www” infers “http://www”).

|  |  |
| --- | --- |
| **Example String** | **Expected Behavior/ Result** |
| www.example.com | Create hyperlink because protocol is implied[[8]](#footnote-7) |
| label@example.com | Create mailto: label@example.com because protocol is implied. |

3. When linkifying URLs that contain bidirectional text, remember that the surrounding HTML may include codes that affect the direction in which text is displayed. The linkified version should keep the same display direction.

1. If TLDs are used as a ‘special token’ to determine linkability, then all TLDs must be included. A list of TLDs should be updated on a frequent basis.

## Unicode—Background and code point attributes

The Unicode standard has been evolving since Unicode 1.0 in 1991. Each version has added more characters and code points to handle more languages and scripts. The current version is 12.1.

In Unicode, every code point has a set of properties, such as Uppercase\_Letter, Decimal\_Number, or Nonspacing\_Mark. Many characters have a Script property such as Latin, Han (Chinese), or Arabic, while others such as punctuation do not belong to a particular script.

As described below, IDNA uses code point attributes to determine which characters are allowed in IDNs. UAX#44, *Unicode Character Database*, describes the database of code point attributes at <https://www.unicode.org/reports/tr44/>

## UTF8, UTF16, and other encoding methods

A Unicode code point can have a numeric value ranging from zero to 0x10FFFF. Since a single byte can only hold values from 0 to 0xFF, some sort of multi-byte encoding is needed to store Unicode code points.

The original version of Unicode had less than 64K (0xFFFF) code points, so each code point could fit in a 16 bit integer. This led to an two-byte encoding known as UCS or UCS-2. When Unicode expanded past 64K code points, UCS was extended into UTF-16[[9]](#footnote-8), which uses pairs of otherwise invalid 16-bit code points known as *surrogates* to represent values greater than 64K. While this works, it has led to debugging problems since surrogates add complexity to any code that counts the number of code points in a string, or that sorts strings into code point order. An additional problem is that some computers such as those made by IBM store the high byte of a 16-bit value first (“big-endian”) and some such as those made by Intel store the low byte first (“little-endian”) so UTF-16 has two storage variants, UTF-16BE and UTF-16LE. There are techniques to detect and fix endian problems but again they can lead to bugs. At this point, UTF-16 is primarily used in existing applications with Microsoft Windows APIs, and the Java and Javascript languages.

An alternative encoding is UTF-8, which encodes each code point as a variable length string of one to four bytes. UTF-8 has several advantages over UTF-16: the ASCII subset of Unicode is encoded as a single byte, so any ASCII string is automatically a UTF-8 string. UTF-8 is usually more compact than its UTF-16 equivalent. UTF-8 is easier to sort, because UTF-8 strings sorted in byte order are automatically in code point order. IDNA and EAI all require UTF-8 coding.

UTF-32 is a simple format that stores each code point in a 32-bit integer. It is convenient for internal processing in programs since the code points in an array of UTF-32 can be indexed directly, but it is rarely used in storage because of its bulk.

## IDNA—A Brief History and Current state

Internationalized Domain Names in Applications (IDNA) was first defined by the IETF in 2003, as what is now known as IDNA2003[[10]](#footnote-9). It included an algorithm to map Unicode code points into a standard form in domain name labels known as Nameprep, and an algorithm to encode Unicode code point labels in ASCII known as Punycode. Nameprep includes transformations such as mapping upper to lower case.

After some experience with IDNA, the IETF developed and published a revised spec known as IDNA2008 in 2010[[11]](#footnote-10). IDNA2008 created the terms U-label and A-label, removed the Nameprep step, saying instead that applications should do a mapping appropriate for the locale and application environment. IDNA2008 was updated for Unicode 6.0 by RFC 6452 in 2011, and is reviewed by IETF up until and including Unicode 12.1.

In practice too many implementations are still using IDNA2003, or some version of IDNA2003 applied to newer versions of Unicode. A few libraries do use tables (like the ones included in IDNA2003) created for IDNA2008. No locale mappings exists for IDNA2008 except the standard case folding and normalization rules included in the Unicode Standard.

One exception are a few mappings from UTS#46, Unicode IDNA Compatibility Processing, <https://www.unicode.org/reports/tr46/#Mapping>, This specifies whether a few quite common characters which is mapped in IDNA2003 but allowed as characters in IDNA2008 should be accepted or mapped. It is very important applications do treat these characters according to IDNA2008 and not IDNA2003, and because of this that if UTS#46 is in use, it is used in a way that is compatible with IDNA2008.

## Use Cases for Testing

Software that is intended to handle IDNs and EAI mail addresses should be tested with a wide range of domain names and addresses. See [UASG 004](https://uasg.tech/wp-content/uploads/documents/UASG004-en-digital.pdf), *Use Cases for UA Readiness Evaluation*, for a set of test cases.

## Upgrading software for EAI

EAI conformance requires upgrades to mail servers, submission and delivery software, mail user agents and web mail, and any application that handles email addresses and sends mail.

For a detailed overview of EAI, its issues, and how to implement it, see [UASG 012](https://uasg.tech/wp-content/uploads/documents/UASG012-en-digital.pdf), *Email Address Internationalization (EAI): A Technical Overview.*

# Part 3: Advanced Topics

## Complex Scripts

The details of complex scripts may be of limited interest to those who are not developers creating their own string parsing or display libraries. Nevertheless, a summary is included here to ensure that all readers have sufficient awareness to recognize code bugs related to these scripts when encountered in user experiences.

For formatted HTML text in web pages and email, the HTML standards have elaborate features for handling and displaying complex and bidirectional text, which developers should understand and use to render text. See the WHATWG HTML standard section on rendering[[12]](#footnote-11), and the corresponding section of the W3C HTML standard[[13]](#footnote-12).

### Right to Left Languages and Unicode Conformance

Many scripts display characters from left to right when text is presented in horizontal lines. Other scripts, such as Arabic or Hebrew, the ordering of horizontal text in display is from right to left. The text can also be bidirectional (left to right – right to left) when a right-to-left script uses digits that are written from left to right or when it uses embedded words from English or other scripts.

Challenges and ambiguities can occur when the horizontal direction of the text is not uniform. To solve this issue, there is an algorithm to determine the directionality for bidirectional Unicode text.

There is a set of rules that should be applied by the application to produce the correct order at the time of display which are described by the **Unicode Bidirectional Algorithm**. We generally refer to this as the “**Bidi algorithm**”.

### The Bidi Algorithm

The Bidi algorithm describes how software should process text that contains both left-to-right (LTR) and right-to-left (RTL) sequences of characters. The **base direction[[14]](#footnote-13)**assigned to the phrase will determine the order in which text is displayed. This can be either left to right or right to left and defines in what order sequences of characters are displayed. In this document, the base direction is left to right so all sequences of characters are displayed with the first sequence to the left of the second sequence.

To know if a sequence is left-to-right or right-to-left, each character in Unicode has an associated directional property. Most letters are **strongly typed** (**strong characters**) as LTR (left-to-right. or RTL (right-to-left) depending on the script of which they are a part. A sequence of strongly-typed RTL characters will be displayed from right to left. This is independent of the surrounding base direction. For example:

(LTR) example - مثال (RTL).

Text with different directionality can be mixed in line. In such cases, the Bidi algorithm produces a separate **directional run** out of each sequence of contiguous characters with the same directionality.

Spaces and most punctuation are not strongly typed as either LTR or RTL in Unicode because they may be used in either type of script. They are therefore classified as **neutral** or **weak** **characters**. Weak characters are those which are generally used in one direction, but in some contexts may be used in the other. Examples of this type of character include:

· European digits

· Eastern Arabic-Indic digits

· Arithmetic symbols, and currency symbols

· Punctuation symbols that are common to many scripts, such as the colon, comma, full-stop, and the no-break-space

The directionality of **neutral characters** isindeterminate without context. Some examples include:

· Tabs

· Paragraph separators

· Most other whitespace characters

When a neutral character is between two strongly typed characters that have the same directional type, it will also assume that directionality. For example, a neutral characterbetween two RTL characters will be treated as a RTL character itself, and will have the effect of extending the directional run:

· مثال.نطاق

Even if there are several neutral characters between the two strongly typed characters, they will all be treated in the same way.

When a space or punctuation falls between two strongly typed characters that have different directionality, the neutral character (or characters) will be treated as if they have the same directionality as the prevailing base direction. For example (remember that this document is has left to right as base sequence so below *example* is the 2nd level domain and *مثال* the TLD):

· example. مثال

Unless a directional override is present, **numbers** are always encoded (and entered) high digit first, and the numerals rendered LTR. The weak directionality only applies to the placement of the number in its entirety.

To see the full details of the Bidi algorithm, see:<http://unicode.org/reports/tr9/>

### The Bidi Rule for Domain Names

A **Bidi domain name** is one that contains at least one RTL label. The Bidi rule for domain names, specified in RFC 5893[[15]](#footnote-14), limits the code points in names so that, roughly speaking, there are not two names that are different sequences of code points but display the same due to bidirectional display rules.

### Joiners

Some languages use alphabetic scripts in which single phonemes are written using two characters called a **digraph**. In other words, a digraph is a group of two successive letters that represent a single sound (or **phoneme**).

|  |
| --- |
| **Examples of digraphs in English** |
| ch (as in church)ph (as in phony) | th (then)th (think) | sh (shoe)gh (rough) |

Some digraphs are fully joined as **ligatures**. In writing and typography, a ligature occurs where two or more graphemes or letters are joined as a single glyph. An example is the ampersand character (*&*), which evolved from the adjoined Latin letters *e* and *t* (“et” means “and”). In typeset English, fi and ffi often are displayed as ligatures.

If ligatures and digraphs have the same interpretation in all languages that use a given script, Unicode normalization generally resolves the differences and makes them match. When they have different interpretations, matching must use alternative methods, likely chosen at the registry level, or users must be educated to understand that matching will not occur. An example of different interpretation can be found in Section 4.3 of RFC 5894[[16]](#footnote-15). The Unicode Consortium lists two main strategies to determine the joining behavior of a particular character after applying the Bidi algorithm, to deal with zero width joiner characters known as ZWJ and ZWNJ. (To learn more about these Joiners see <http://www.unicode.org/L2/L2005/05307-zwj-zwnj.pdf>.)

* “When shaping, an implementation can refer back to the original backing store to see if there were adjacent ZWNJ or ZWJ characters.
* Alternatively, the implementation can replace ZWJ and ZWNJ by an out-of-band character property associated with those adjacent characters, so that the information does not interfere with the Bidi algorithm and the information is preserved across rearrangement of those characters. Once the Bidi algorithm has been applied, that out-of-band information can then be used for proper shaping.

Domain name registries and any other entity that allows the creation of domain names (e.g. applications that create third and lower level labels) must follow the Bidi Rule for Domain Names to ensure that names will display consistently and to prevent confusing names that can be used for homograph attacks.

To learn more about joiners, see Section 4.3 of RFC 5894.

### Homoglyphs and Confusingly Similar Characters

Homoglyphs are characters that, due to similarities in size and shape, appear identical or confusingly similar. They frequently occur when mixing Latin, Cyrillic, and Greek scripts, for example Latin “o” (code U+006f), Cyrillic small letter “о” (code U+043e), and Greek small letter omicron “ο” (code U+03bf.) In some cases there are homoglyphs in a single script, such as the small Croatian letter “ǉ” (code U+01c9) and the two letters “lj” (code U+006c U+006a). See the table at <http://homoglyphs.net/> for more examples.

To prevent domain names with homoglyphs, registries should use Label Generation Rules (LGRs) that limit the code points in a label to a set from a single script or compatible scripts. Each registry should have LGRs for each script in which it accepts registrations[[17]](#footnote-16).

To learn more about Unicode security mechanisms for confusable detection, see:

• <http://www.unicode.org/reports/tr39/#Confusable_Detection>

To learn more about confusingly similar characters and good practice, see:

• M3AAWG Unicode Abuse Overview and Tutorial

<https://www.m3aawg.org/sites/default/files/m3aawg-unicode-tutorial-2016-02.pdf>

• M3AAWG Best Practices for Unicode Abuse Prevention

<https://www.m3aawg.org/sites/default/files/m3aawg-unicode-best-practices-2016-02.pdf>

## Normalization, Case Folding, and String Preparation

Unicode Normalization helps to determine whether any two Unicode strings are equivalent to each other, and provides standard forms to use to process and store strings. Some characters can be represented in Unicode by several code sequences. This is called **Unicode equivalence**. Unicode provides two types of equivalences:

· Canonical

· Compatibility

Sequences representing the same visual character are called **canonically equivalent.** These sequences have the same appearance and meaning when printed or displayed. For example:

|  |  |
| --- | --- |
| U+006E (Latin lowercase “n”) followed by U+0303 (the combining tilde “◌̃”) | = **ñ** |
| U+00F1 (lowercase letter “ñ” of the Spanish alphabet) | = **ñ** |

Unicode defines NFC, Normalization Form C, as Canonical Decomposition, followed by Canonical Composition. This reduces text to a minimal number of code points while not changing its appearance. It should be noted that in this example, three characters above are valid to be used according to IDNA2008.

**Compatibility equivalents** are sequences which can appear different, but in some contexts the same meaning. It is a weaker type of equivalence between characters or sequences of characters. For example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|

|  |  |
| --- | --- |
| U+FB00 (the typographic ligature “ﬀ”) | = **ff** |
| U+0066 U+0066 (two Latin “f” letters) | = **ff** |

 |

In the example above, the code point U+FB00 is defined to be compatible, but not canonically equivalent to the sequence U+0066 U+0066. Sequences that are canonically equivalent are also compatible, but the opposite is not always true.

It should be noted that the code point U+FB00 is not valid according to IDNA2008.

Unicode defines NFKC, Normalization Form KC, a Compatibility Decomposition, followed by Canonical Composition. This reduces text to a standard set code points and may change its appearance. For example, NKFC turns the ligature “ﬀ” into the two letters “f f” and the ante meridian symbol ㏂ (U+33C2) into the four characters “a.m.” (U+0061 U+002E U+006D U+002E.)

To avoid interoperability problems arising from the use of canonically equivalent, yet different, character sequences, the W3C recommends using NFC for all text.

To see alist of all characters that may change in any of the Normalization Forms, see:<http://www.unicode.org/charts/normalization>

Some other points to note:

* The characters in IDN labels must be in NFC form.
* When two applications share Unicode data, but normalize them differently, errors and data loss can occur.
* The Unicode consortium asserts that Normalization Forms must remain stable over time. In other words, a string must remain normalized under all future versions of Unicode for backward compatibility.
* As pointed out earlier, be conservative when looking at what code points to allow in a domain name.

|  |
| --- |
| **Tip for software developers** |
| **✖** | Don’t attempt to normalize by converting to uppercase, or ignoring non-spacing characters, because this makes sorting, data copying import and export, and data retrieval by client applications difficult and may result in data loss or corruption. |
| **✖** | Never allow code points in domain names which are not allowed according to IDNA2008. |

To learn more about Unicode normalization, see:

· <http://www.w3.org/TR/charmod-norm>

· <http://unicode.org/reports/tr15>

### Case Folding and Mapping

**Case folding** and **case mapping** is the process of turning all of the characters in a string into the same case, usually lower case. Mapping upper case [A-Z] to lower case [a-z] works for ASCII-only text documents, but it is far more complex in languages that use additional characters. Case mapping can be *context-dependent*, with the mapped character depending on the context in which it occurs, e.g., various forms of the Greek sigma. It can also be *locale-dependent*, with the mapped character depending on the locale in which the text is interpreted, e.g., Turkish dotted and undotted upper and lower case I. Case folding is locale-independent, for strings that will be interpreted by software, while case mapping is locale-dependent and is intended for text to be read by people. Finally, it is not the case mapping to upper case and mapping to lower case are inverse functions.

For IDNs, IDNA2008 allows applications to use any appropriate case mapping because the mapping is happening before the validation of code points takes place. In practice, locale-specific identifier mappings do not exist and everyone uses the mappings from Unicode’s UTS#46[[18]](#footnote-17).

|  |
| --- |
| **Tip for software developers** |
| **✔** | Consider the desired goal before attempting case mapping: is it a generic map for labels, a string in a known language, or something else? |
| **✔** | Perform Unicode Normalization before case folding. |

# Part 5: Glossary and Other Resources

**Glossary**

|  |  |
| --- | --- |
| **A-label** | The ASCII-compatible encoded (ACE) representation of a label in an internationalized domain name, used internally within the DNS protocol. A-labels always commence with the ACE prefix “xn--”. It can be converted to a U-label and back without loss of information. |
| **ACE prefix** | ASCII Compatible Encoding Prefix “xn--”. |
| **ASCII Characters** | American Standard Code for Information Interchange. These include unaccented Latin characters and the European-Arabic digits. ASCII can be viewed as a subset of Unicode, and every ASCII character is also a Unicode character. |
| **API** | An Application Programming Interface (API) is a set of routines, protocols, and tools for building software and applications. An API may be for a web based system, operating system, or database system, and it provides facilities to develop applications for that system using a given programming language. |
| **Codespace** | Range that define the lower and upper bounds for an encoding. |
| **Code Points** | Code points are a numerical values in the code space. They are used to distinguish both, the number from an encoding as a sequence of bits, and the abstract character from a particular graphical representation (glyph). |
| **DNS Root Zone** | The root zone is the central directory for the DNS, which is a key component in looking things up in DNS, for example translating host names into IP addresses. |
| **EAI** | Email Address Internationalization allows UTF-8 characters in an email address, in the domain name, the local part or both. |
| **IANA** | Internet Assigned Numbers Authority. Its functions include:· Maintenance of the registry of technical Internet protocol parameters· Administration of certain responsibilities associated with Internet DNS root zone· Allocation of Internet numbering resources |
| **ICANN** | The Internet Corporation for Assigned Names and Numbers (ICANN) is an internationally organized, non-profit corporation that has responsibility for Internet Protocol (IP) address space allocation, protocol identifier assignment, generic (gTLD) and country code (ccTLD) Top-Level Domain name system management, and root server system management functions. |
| **IDN** | Internationalized Domain Names. IDNs are domain names that include UTF-8 characters beyond the twenty-six letters of the basic Latin alphabet “a-z”, the numbers 0-9, and the hyphen “-“. |
| **IDNA** | Internationalized Domain Names in Applications. |
| **IDN ccTLD** | Country Code Top-level Domain that includes characters beyond the twenty-six letters of the basic Latin alphabet “a-z”. Examples:· .рф (Russia)· صر. (Egypt)· السعودية. (Saudi Arabia) |
| **IETF** | The Internet Engineering Task Force (IETF) is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual. The IETF develops Internet Standards and in particular the standards related to the Internet Protocol Suite (TCP/IP) and the protocols used for the web like HTTP and TLS. |
| **Language** | The method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way. |
| **Punycode** | It is an algorithm that represents UTF-8 in the limited character subset of ASCII supported by the Domain Name System. Punycode is used in A- labels in the Internationalized Domain Names in Applications (IDNA) framework. |
| **Registrar** | An organization where domain names are registered by users. The registrar keeps records of the contact information and submits the technical information to a central directory known as the “registry”. |
| **Registry** | The authoritative, master database of all domain names registered in each Top Level Domain. |
| **RFC** | A Request for Comments (RFC) is a formal document from the Internet Engineering Task Force (IETF) that is the result of committee drafting and subsequent review by interested parties. |
| **Script** | The collection of letters or characters used in writing, representing the sounds of a language. |
| **Second-level domain name** | In the Domain Name System (DNS) hierarchy, a **s**econd-level domain (SLD or 2LD) is a domain that is directly below a top-level domain (TLD). For example, in example.com, example is the second-level domain of the .com TLD. |
| **U-label** | A "U-label" is an IDNA-valid string of Unicode characters including at least one non-ASCII character. It can be converted to an A-label and back without loss of information.  |
| **UA-ready Software or UA-Readiness** | Universal Acceptance Ready Software. It is a software that has the ability to Accept, Store, Process, Validate and Display all Top Level Domains equally and all IDNs, hyperlink and email addresses equally. |
| **Unicode** | A universal character encoding standard. It defines the way individual characters are represented in text files, web pages, and other types of documents. Unicode was designed to support characters from all languages around the world. It can support roughly 1,000,000 characters. See:<http://unicode.org> |
| **UTF** | Unicode Transformation Format. It is a way of representing Unicode code points as a stream of bytes. UTF-8 is the preferred UTF for handling IDN and EAI. UTF-8 converts Unicode to 8-bit bytes. |
| **M3AAWG** | The Messaging, Malware and Mobile Anti-Abuse Working Group (M3AAWG) is where the industry comes together to work against botnets, malware, spam, viruses, DoS attacks and other online exploitation. See:<https://www.m3aawg.org/> |
| **W3C** | The World Wide Web Consortium (W3C) is an international community where [Member organizations](https://www.w3.org/Consortium/Member/List), a full-time [staff](https://www.w3.org/People/), and the public work together to develop [Web standards](https://www.w3.org/standards/) like HTML. See:<https://www.w3.org/> |
| **WHATWG** | The Web Hypertext Application Technology Working Group (WHATWG) is a community of people interested in evolving the web through standards and tests.The WHATWG was founded by individuals of Apple, the Mozilla Foundation, and Opera Software in 2004, after a W3C workshop. See <https://whatwg.org/> |
| **ZWJ** | Zero-Width Joiner is non-printing character used in the computerized typesetting of some complex scripts such as the Arabic script or any Indic script. When placed between two characters that would otherwise not be connected, a ZWJ causes them to be printed in their connected forms. |
| **ZWNJ** | Zero-Width Non-Joiner is a non-printing character used in the computerization of writing systems that make use of ligatures. When placed between two characters that would otherwise be connected into a ligature, a ZWNJ causes them to be printed in their final and initial forms, respectively. This is also an effect of a space character, but a ZWNJ is used when it is desirable to keep the words closer together or to connect a word with its morpheme. |

For a complete ICANN glossary, go to:<https://www.icann.org/resources/pages/glossary-2014-02-03-en>

## RFCs and Key Standards

|  |
| --- |
| **IDN RFCs** |
| **RFC 3492** | **Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)**RFC 3492 describes Punycode as:"*a simple and efficient transfer encoding syntax designed for use with Internationalized Domain Names in Applications (IDNA)"*Punycode transforms uniquely and reversibly a Unicode string into an ASCII string. This RFC defines a general algorithm called **Bootstring**. This algorithm allows a string of basic code points to uniquely represent any string of code points drawn from a larger set.<https://tools.ietf.org/html/rfc3492> |
| **RFC 5890** | **Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework**This RFC describes the usage context and protocol for a revision of Internationalized Domain Names for Applications (IDNA).<https://tools.ietf.org/html/rfc5890> |
| **RFC 5891** | **Internationalized Domain Names in Applications (IDNA) Protocol**This RFC specifies the protocol mechanism, called Internationalized Domain Names in Applications (IDNA), for registering and looking up IDNs in a way that does not require changes to the DNS itself.<https://tools.ietf.org/html/rfc5891> |
| **RFC 5892** | **The Unicode Points and Internationalized Domain Names for Applications (IDNA)**The RFC 5892 specifies rules for deciding whether a code point, considered in isolation or in context, is a candidate for inclusion in an Internationalized Domain Name (IDN).<https://tools.ietf.org/html/rfc5892> |
| **RFC 5893** | **Right-to-left scripts for Internationalized Domain Names for Applications (IDNA)**This RFC provides a new Bidi rule for Internationalized Domain Names for Applications (IDNA) labels, for the use of right-to-left scripts in Internationalized Domain Names.<https://tools.ietf.org/html/rfc5893> |
| **RFC 5894** | **Internationalized Domain Names for Applications (IDNA): Background, Explanation and Rationale**This informational document provides an overview of a revised system to deal with newer versions of Unicode and provides explanatory material for its components.<https://tools.ietf.org/html/rfc5894> |
| **RFC 5895** | **Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008**This RFC describes the actions that can be taken by an implementation between receiving user input and passing permitted code points to the new IDNA protocol (2008). It describes an operation that is to be applied to user input in order to prepare that user input for use in an “on the network” protocol. It also includes a general implementation procedure for mapping.<https://tools.ietf.org/html/rfc5895> |
| **EAI RFCs** |
| **RFC 6530** | **Overview and Framework for Internationalized Email**This standard introduces a series of specifications that define mechanisms and protocol extensions needed to fully support internationalized email addresses. This document describes how the various elements of email internationalization fit together and the relationships among the primary specifications associated with message transport, header formats, and handling.<https://tools.ietf.org/html/rfc6530> |
| **RFC 6531** | **SMTP Extension for Internationalized Email**The document defines a Simple Mail Transfer Protocol extension so servers can advertise the ability to accept and process internationalized email addresses and internationalized email headers.<https://tools.ietf.org/html/rfc6531> |
| **RFC 6532** | **Internationalized Email Headers**This document specifies an enhancement to the Internet Message Format and to MIME that allows use of Unicode in mail addresses and most header field content. This document specifies an enhancement to the Internet Message Format (RFC 5322) and to MIME that permits the direct use of UTF-8, rather than only ASCII, in header field values, including mail addresses. A new media type, message/global, is defined for messages that use this extended format. This specification also lifts the MIME restriction on having non-identity content-transfer-encodings on any subtype of the message top-level type so that message/global parts can be safely transmitted across existing mail infrastructure.<https://tools.ietf.org/html/rfc6532> |
| **RFC 6533** | **Internationalized Delivery Status and Disposition Notifications**This specification adds a new address type for international email addresses so an original recipient address with non-ASCII characters can be correctly preserved even after downgrading. This also provides updated content return media types for delivery status notifications and message disposition notifications to support use of the new address type.<https://tools.ietf.org/html/rfc6533> |

## Key Standards

|  |  |
| --- | --- |
| **ISO 10646 (Unicode)** | To provide a common technical basis for the processing of electronic information in various languages, the International Organization for Standardization (ISO) has developed an international coding standard called ISO 10646. The ISO 10646 provides a unified standard for the coding of characters in all major languages in the world, including traditional and simplified Chinese characters. This large character set is called the Universal Character Set (UCS). The same set of characters is defined by the Unicode standard, which further defines additional character properties and other application details of great interest to implementers.Unicode is a character coding system designed by the Unicode Consortium to support the interchange, processing and display of the written texts of all major languages in the world. ISO 10646 and Unicode define several encoding forms of their common repertoire: UTF-8, UCS-2, UTF-16, UCS-4 and UTF-32.<http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=63182> |
| **GB18030 (China)** | GB 18030-2000 is a Chinese government standard that specifies an extended code page for use in the Chinese market in addition to UTF-8. The internal processing code for the character repertoire can and should be Unicode; however, the standard stipulates that software providers must guarantee a successful round-trip between GB18030 and the internal processing code. All products currently sold or to be sold in China must plan the code page migration to support GB18030 without exception. GB18030 is a “mandatory standard” and the Chinese government regulates the certification process to reinforce GB18030 deployment.<http://icu-project.org/docs/papers/unicode-gb18030-faq.html> |

## Online Resources

|  |  |
| --- | --- |
| **APIs** | Windows APIs[https://www.msdn.microsoft.com/enus/library/windows/desktop/ff818516%28v=vs.85%29.aspx](https://www.msdn.microsoft.com/enus/library/windows/desktop/ff818516%28v%3Dvs.85%29.aspx) SharePoint APIs<https://msdn.microsoft.com/en-us/library/office/jj860569.aspx>Public Suffix List<https://publicsuffix.org/list/public_suffix_list.dat>ICANN Authoritative TLD list<http://data.iana.org/TLD/tlds-alpha-by-domain.txt>Android APIs<http://developer.android.com/guide/index.html>MAC IOS APIs<https://developer.apple.com/library/mac/navigation>.Net Framework [https://msdn.microsoft.com/en-us/library/system.text.encoding(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.text.encoding%28v%3Dvs.110%29.aspx) |
| **Unicode Security** | Unicode Security considerations<http://www.unicode.org/reports/tr36>Unicode security mechanisms<http://www.unicode.org/reports/tr39> |
| **Unicode character groupings** | Unicode code planes<http://en.wikipedia.org/wiki/Mapping_of_Unicode_character_planes> Overview of GB18030<http://en.wikipedia.org/wiki/GB_18030>Authoritative mapping table between BG18030-2000 and Unicode<http://source.icu-project.org/repos/icu/data/trunk/charset/data/xml/gb-18030-2000.xml>Unicode normalization<https://en.wikipedia.org/wiki/Unicode_equivalence> |
| **Unicode exploits** | Section 3.1, “UTF-8 Exploits” in Unicode Technical Report #36<http://unicode.org/reports/tr36/#UTF-8_Exploit>M3AAWG Best Practices for Unicode Abuse Prevention<https://www.m3aawg.org/sites/default/files/m3aawg-unicode-best-practices-2016-02.pdf>M3AAWG Unicode Abuse Overview and Tutorial<https://www.m3aawg.org/sites/default/files/m3aawg-unicode-tutorial-2016-02.pdf> See also:<http://www.unicode.org> |
| **Miscellaneous** | URIs<http://tools.ietf.org/html/rfc3986>The Domain Name System: A Non-Technical Explanation – Why Universal Resolvability Is Important<http://www.internic.net/faqs/authoritative-dns.html>ICANN glossary<https://www.icann.org/resources/pages/glossary-2014-02-03-en> |

1. ASCII Compatible Encoding (ACE) prefix which distinguishes Punycode-encoded labels from other ASCII labels. [↑](#footnote-ref-0)
2. Accepting is treated as distinct from Validating in this document. In practice, the actions may overlap. [↑](#footnote-ref-1)
3. Accepting and Processing are treated as distinct from Validating in this document. In practice, the actions may overlap. [↑](#footnote-ref-2)
4. The 63-character length limit applies to the label itself if it is an ASCII label, or to the A-label form of the label if it is an IDN. [↑](#footnote-ref-3)
5. Processing is treated as distinct from Validating in this document. In practice, the actions may overlap. [↑](#footnote-ref-4)
6. It is important to recognize that display is not straightforward, even when these conditions are met, for some complex scripts. [↑](#footnote-ref-5)
7. See https://www.unicode.org/reports/tr39/#Restriction\_Level\_Detection [↑](#footnote-ref-6)
8. Note: the actual website might be https-only and require https:// instead of http://. If this is the case, then the hyperlink may not resolve. [↑](#footnote-ref-7)
9. See section 3.10 of the Unicode standard for technical details of UTF-8, UTF-16, and UTF-32, at <https://www.unicode.org/versions/Unicode12.0.0/ch03.pdf> [↑](#footnote-ref-8)
10. The definition is in RFCs [3490](https://tools.ietf.org/html/rfc3490), [3491](https://tools.ietf.org/html/rfc3491), and [3492](https://www.ietf.org/rfc/rfc3492.txt). [↑](#footnote-ref-9)
11. The definition is in RFCs [5890](https://tools.ietf.org/html/rfc5890), [5891](https://tools.ietf.org/html/rfc5891), [5892](https://tools.ietf.org/html/rfc5892), [5893](https://tools.ietf.org/html/rfc5893), [5894](https://tools.ietf.org/html/rfc5894), and [5895](https://tools.ietf.org/html/rfc5895). [↑](#footnote-ref-10)
12. Available at <https://html.spec.whatwg.org/multipage/rendering.html> [↑](#footnote-ref-11)
13. Available at <https://www.w3.org/TR/2018/WD-html53-20181018/rendering.html> [↑](#footnote-ref-12)
14. In HTML the base direction is either inherited from the default direction of the document, which is left-to-right, or explicitly set by the nearest parent element that uses the “dir” direction attribute. [↑](#footnote-ref-13)
15. *Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)*, RFC 5893, <https://www.rfc-editor.org/info/rfc5893> [↑](#footnote-ref-14)
16. *Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale*, RFC 5894, <https://www.rfc-editor.org/rfc/rfc5894.html#section-4.2> [↑](#footnote-ref-15)
17. IANA has a collection of registry LGRs in its Repository of IDN Practices at <https://www.iana.org/domains/idn-tables>. [↑](#footnote-ref-16)
18. UTS#46, *Unicode IDNA Compatibility Processing*, <https://www.unicode.org/reports/tr46/#Mapping> [↑](#footnote-ref-17)