Introduction to Universal Acceptance

Mark Svancarek and Luisa Villa

Universal Acceptance Steering Group

About This Document

Purpose

The Internet's technologies, including its naming components, are under continual evolution and change. In recent years, a great number of new TLDs with ASCII characters and IDN top-level domains have been released by ICANN. Examples include .nyc, .hsbc, .eco, and .ストア. However, the response to the change in the naming landscape has not been fast enough. Many applications and services are not being updated to manage new TLDs. This affects the user experience. For example:

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

Many applications and services are not being updated to manage these new TLDs. This affects the user experience.

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

Target Audience

- Software Developers
- Chief Technical Officers (CTOs)
- The technical community in general

Document Structure

Part 1 **Baseline concepts of Universal Acceptance** such as what is a Domain Name and the Domain Name System, ASCII and Unicode, Punycode, 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 Contains the **glossary** and useful **online resources**.

Need more information?

The UASG and the community are available to provide advice to software developers and implementers on what is needed.

- **Contact us** to share your ideas and suggestions on the topic at info@uasg.tech
- **Join the Universal Acceptance** discussion at http://tinyurl.com/ua-discuss
- To **learn more** about the effort, visit http://www.icann.org/universalacceptance

Contents

Introduction

A Brief History of Domain Name Internationalization

In the 1970s, the characters available for registering domain names were 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 (IDN)** through a translation mechanism to support non-ASCII representations of domain names in geographically diverse local scripts (e.g., 普遍接受-测试.世界, ua-test.世界,etc.).

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 Internet-enabled applications, devices and systems.

The Board of Directors of the Internet Corporation for Assigned Names and Numbers (ICANN) 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 addition of IDN ccTLDs and new TLDs has dramatically increased the pace at which TLDs are added to the root zone.

A decade after the IETF released its IDN-related guidelines, and thanks to the ICANN New TLD Program, more than one thousand new TLDs have now been released. In spite of all these efforts, however, much software and many applications are still not Universal Acceptance-ready. This causes problems to Internet users, including those whose languages are written in scripts that include non-ASCII characters.

The Need for Universal Acceptance

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 where all valid domain names and email addresses are **accepted**, **validated**, **stored**, **processed** and **displayed** correctly and consistently by all Internet-enabled applications, devices and systems. In other words, every valid web address resolves to the expected website and every valid email address delivers mail to the expected destination. Due to the rapidly changing domain name landscape, many systems do not recognize or appropriately process new domain names, primarily because they may be in a non-ASCII format, because the software is not aware of the newly released TLD, or because the length of their TLD varies in length. The same is true for email addresses that incorporate these new extensions.

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.

Part 1: Baseline Concepts of Universal Acceptance

This section contains an overview of the basic terms and concepts necessary to understand before reading the more advanced sections of this document.

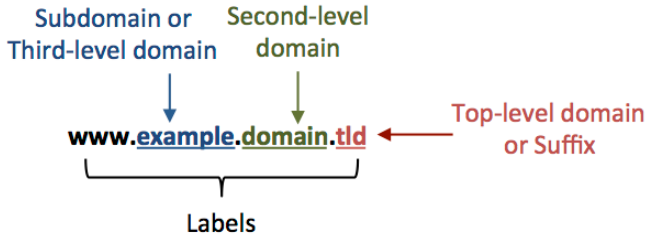
Domain Name

A domain name is a dotted text string used as a human-friendly technical identifier for computers and networks on the Internet. For example:

`www.domain.tld`

How to read a domain name:

- 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.
- The individual words or characters between the dots are called **labels**. For those languages or scripts that are written from **left to right (LTR)**,[1] the label furthest right represents the top-level domain.
- The second label from the end represents the **second-level domain**.
- Any labels that come before the second-level domain are considered **subdomains** of the second-level domain (sometimes called **third-level domains**).



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, the DNS provides a mapping between IP addresses and human-readable domain names. Servers collectively providing a public DNS exist at well-known addresses on the Internet.

Top Level Domains (TLDs)

Human readable domain names are managed by organizations known as **registries.** When a domain name is registered, it consists of multiple text strings representing multiple domain levels, each separated by a "`.`" character. In LTR scripts, the right-most domain level is the top-level domain (TLD). Some TLDs are delegated to specific countries or territories. These are called **Country Code TLDs (ccTDs)**.

[1] Languages or scripts written from right to left (RTL) will be discussed later in this document.

Generic Top Level Domains (gTLDs)

Starting in 2013, ICANN (the organization responsible for the creation and maintenance of TLD assignments) has approved the creation of a large number of new TLDs. These new TLDs can represent brands, communities of interest, geographic communities (cities, regions) and more generic concepts. Collectively, all of these new TLDs are known as Generic Top Level Domains (gTLDs).

Examples of common TLDs	Examples of ccTLDs	Examples of new gTLDs
.com	China = .cn	.app
.gov	Germany = .de	.lawyer
.info	United States = .us	.shopping
.org		.panasonic
		.osaka

Character Sets and Scripts

Languages are written using writing systems. Most writing systems use one script, which is a set of graphic characters used for the written form of one or more languages. A small number of writing systems employ more than one script at the same time. These characters or scripts can be recognized by humans. However, they are not useful to computers. Instead, a computer needs a script to be encoded in a way that it can process (for example, to resolve a web address). The mechanism for this is called a **character mapping** or **coded character set (CCS)**, or a **code page**.[2] A character mapping associates 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 and Unicode.

ASCII and Unicode

In the examples of TLDs above, all of the text strings are represented using the Latin character set. This character set is included in the 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, not all of which can be used in domain names. Domain names are limited to the characters A-Z, the numbers 0-9, and hyphen "-".

[2] There are subtleties to the terms that are not directly relevant to the topic of Universal Acceptance. If you are interested in more information about the terminology, a useful starting point is: https://tools.ietf.org/html/rfc6365

ASCII - ISO 8859-1 (Latin-1) Table [3]

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

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

To see all Unicode character code charts, go to: http://unicode.org/charts

Internationalized Domain Names (IDNs) and Punycode

The use of Unicode enables domain names to contain non-ASCII characters. As noted earlier in this document, domain names that use non-ASCII characters are called Internationalized Domain Names (IDNs).[4] The internationalized portion of a domain name can be in any level – not just the TLD but also the other labels.

Since the DNS itself previously only used ASCII,[5] it was necessary to create an additional encoding to allow non-ASCII Unicode code points to be converted into ASCII strings, and vice versa. 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**.[6]

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

The only RFC-defined[7] use of the Punycode algorithm is for expressing internationalized domains. However, rather than implement Unicode, some developers choose to apply Punycode to other scenarios.

---

[3] Source: California State University. 1997. *ASCII - ISO 8859-1 (Latin-1) Table with HTML Entity Names.* http://web.calstatela.edu/faculty/jchen13/Docs/CS120/Lectures/ASCIITable_with_HTML_Entity_Names.htm

[4] Note that not every non-ASCII character is an IDN.

[5] For current status, see http://tools.ietf.org/html/rfc6055#section-3

[6] ASCII Compatible Encoding (ACE) prefix is used to distinguish Punycode-encoded labels from ordinary ASCII labels.

[7] RFC: Request for Comments. See the Glossary of term in Part 4 of this document for more information.

Examples of (imaginary) IDNs

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

To learn more, see the IDN FAQ: http://unicode.org/faq/idn.html

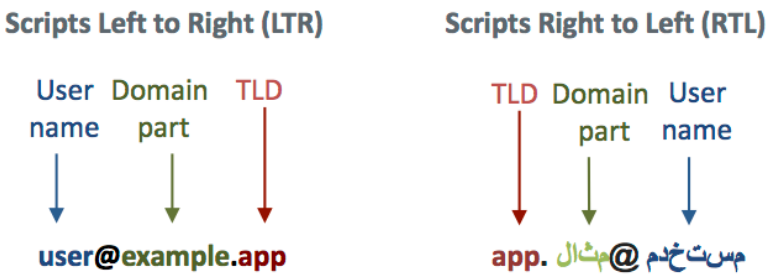
Email

Addresses and Email Address Internationalization (EAI)

Email addresses contain two parts:

1. A local part (the username, before the "@" character)
2. A domain (after the "@" character)

The domain part can contain any TLD, including a new TLD. Both portions may be Unicode U-labels.



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

Examples of (imaginary) Email Addresses including IDNs

user@example.みんな	(Uses internationalized TLD)
user@大坂.info	(Uses internationalized 2nd level domain)
用戶@example.lawyer	(Uses internationalized user name and new gTLD)

Email Address Internationalization (EAI) requires the use of Unicode in all parts of the email address. Each of the examples above could be expressed as EAI, and this is the preferred format.

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.

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 Internet-enabled applications, devices and systems. These five criteria are described below.

1. Accept[8]	***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 into user interfaces, AND/OR • Received from other applications and services via APIs
2. Validate[9]	***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. In other words, validation ensures the syntax correctness of the given information. For domain names and email addresses, many programmers have been using some heuristics (for example, checking that a TLD has the "correct" number of characters, 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 Unicode (non-ASCII) characters • The list of TLDs is growing • A TLD can be up to 63 characters long
3. Store	**The *Storage* process occurs whenever 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.** 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 easily translated to and from RFC-defined formats (this is much less desirable)

[8] Accepting is treated as distinct from Validating in this document. In practice, the abilities may overlap.

[9] Accepting and Processing are treated as distinct from Validating in this document. In practice, the abilities may overlap.

	Although RFCs require the use of UTF-8, other formats may be encountered in legacy code. See the "Good Practices" section below.
4. Process[10]	***Processing* occurs whenever an email address or a domain name is used by an application or service to perform an activity (for example, searching or sorting a list), or transformed into an alternate format (such as storing ASCII as Unicode).** 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 (examples: "Identify all the people associated with New Zealand because they have a name with a `.nz` ccTLD"; "Identify all the pharmacists because they have a `user@example.pharmacy` email address"; "Identify firewalls that might filter DNS requests that don't apply to their policies").
5. Display	**The *Display* process occurs whenever an email address or a domain name is rendered within a user interface.** Displaying domain names and email addresses is usually straightforward when the scripts used are supported in the underlying operating system 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. 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 apps accept these new "`@example.brand`" email addresses as valid as they would with 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, any browser the user wishes to use displays the web address in its native form and accesses the site as the user

[10] Processing is treated as distinct from Validating in this document. In practice, the abilities may overlap.

	expects. The browser does not display Punycoded text 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 exactly as if it were an established TLD such as .biz or .eu.
Accessing an IDN	A user accesses an IDN URL, by typing an address 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 multiple email addresses, some are internationalized (e.g. ĩnfo@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 an EAI 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 EAI 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 even if the address is an EAI 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 Unicode, IDNs and EAI.

Nonconformance to Universal Acceptance Practices

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

✖	Displaying Punycoded text to the user without a corresponding user benefit. For example, to show the mapping between a U-label and a A-label.
✖	Requiring a user to enter Punycoded text when signing up for a new email address or requiring a user to enter Punycoded text 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.
✖	Exposing internal use of Punycoded text to users. For example, converting from EAI to an IDN-style email address 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 TLDs.

Technical Requirements for UA Readiness

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.

2. **Allows multiple character sets that are valid for domain names and email addresses.**

   That is, permits Unicode code points.

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

   See RFC 3490.

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

   For information about RTL scripts, see RFC 5893.

5. **Can communicate data between applications and services in formats that support Unicode and are convertible to/from UTF-8.**

   For information about UTF-8, see RFC 3629.

6. **Offers public APIs that support Unicode & UTF-8.**

7. **Offers private APIs that support Unicode & UTF-8.**

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

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

   Such conversions would be visible only to the product/service owner.

9. **Supports all domain name strings in the authoritative ICANN TLD list and the community-served Public Suffix List regardless of length or character set.**

   See https://newgtlds.icann.org/en/program-status/delegated-strings.

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

    See RFC 6530.

11. **Treats EAI addresses the same way as their Punycoded equivalents (IDN email format).**

Developer Considerations

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

A Guiding Principle for Achieving Universal Acceptance: 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. In 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."

In other words, be conservative in what you send and be liberal in what you accept. This is also a good approach when dealing with the vagaries of Universal Acceptance currently implemented in the ecosystem.

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

Accept	
✔	Always offer Unicode equivalents. Users should be allowed, but not required, to enter ASCII Compatible Encoded (or "Punycoded") text in place of its Unicode equivalent. However, Unicode should be shown by default, with Punycoded text only shown to the user only when it provides a benefit.
!	**Don't** generate IDN-Style email addresses, 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 support Unicode, labels up to 63 characters, and domain name strings up to 253 characters. • See RFC 1035.

Validate	
✔	Validate only to the minimum extend necessary**.** **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 not represent domain names or email addresses currently in use on the Internet.

!	If you must validate, consider the following:
	
	• Verify the TLD portion of a domain name against an authoritative table. Examples of some authoritative tables that you can use are:
	  ○ http://www.internic.net/domain/root.zone
	  ○ http://www.dns.icann.org/services/authoritative-dns/index.html
	  ○ 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
	  ○ Consider using the GETDNS API (http://getdnsapi.net/)
	• Require repeated entry of an email address to preclude typing errors

	• 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
	  ○ See RFC 5892
	• Limit validation of labels itself to a small number of whole-label rules defined in the RFCs
	  ○ See RFC 5894
	• If a string resembling a domain name contains the Arabic full stop character "-" (U+06D4), or the ideographic full stop character "。" (U+002E), convert it to the full stop "." (U+002E).
	• 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

Store	
✔	Applications and services should support the appropriate Unicode standards.
✔	Information should be stored in the UTF-8 (Unicode Transformation Format) whenever possible. Some systems may require support for UTF-16 as well, but generally UTF-8 is preferred. UTF-7 and UTF-32 should be avoided.
!	Consider all end-to-end scenarios before converting A-Labels (Punycode) to U-Labels and vice versa when storing. It may be desirable to maintain only U-Labels in a file or database, because it simplifies searching and sorting. However, conversion may have implications when interoperating with older, non-Unicode-enabled applications and services. Consider storing and indexing both formats.
✔	Clearly mark email addresses and domain names during storage for easier access. Instances where email addresses and domain names have been filed under the "author" field of a document or "contact info" in a log file have led to the loss of the original address.
✔	If you *don't* store in Unicode, 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 server responses have Unicode specified in the content type.
✔	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

!	Consider all end-to-end scenarios before converting A-Labels (Punycode) to U-Labels and vice versa during processing. It may be 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. Consider storing in both formats.
✔	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.
✖	Don't use URL-encoding for 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 will not "break" the user experience. Missing fonts in the underlying operating system may result in non-displayable characters (frequently the "□"character is used to represent these), but this situation should not result in a fatal crash.
✔	Use supported Unicode-enabled APIs.
✔	Use the latest Internationalized Domain Names in Applications (IDNA) Protocol and Tables documents for IDNs: • RFC 5891 • RFC 5892
✔	Process in UTF-8 format wherever possible.
✔	Upgrade applications and servers/services together. If the server is Unicode and client is non-Unicode, or vice versa, the data will need to be converted to each code page every time the data travels between server and client.
✔	Perform code reviews to avoid buffer overflow attacks. When doing character transformation, text strings may grow or shrink substantially.

Display	
✔	Display all Unicode code points that are supported by the underlying operating system. If an application maintains its own font sets, comprehensive Unicode support should be offered to the collection of fonts available from the operating system.
✔	When developing an app or a service consider the languages supported and make sure operating systems and applications cover those languages.
✔	Convert non-Unicode data to Unicode 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 Unicode by default. Display Punycoded text to the user only when it provides a benefit.
!	Be aware that mixed-script addresses will become more common. • Some Unicode characters may look the same to the human eye, but different to computers • Don't assume that mixed-script strings are intended for malicious purposes, such as phishing • 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 scripts Learn more about Unicode Security Considerations at: http://unicode.org/reports/tr36
✔	Use Unicode IDNA Compatibility Processing in order to match user expectations. To learn more, go to: http://unicode.org/reports/tr46
✔	Be aware of unassigned and disallowed characters for domain names. • See RFC 5892

Unicode	
✔	Use supported Unicode-enabled APIs.
✖	Don't build your own 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 or UTF-32. • UTF-7 is generally not used as a native representation within applications as it is very awkward to process. Despite its size advantage over the combination of UTF-8 with either quoted-printable or base64, the Internet Mail Consortium recommends against its use. • The main disadvantage of UTF-32 is that it is space inefficient, using four bytes per code point. Non-BMP characters are so rare in most texts[citation needed], they may as well be considered non-existent for sizing issues, making UTF-32 up to twice the size of UTF-16 and up to four times the size of UTF-8.
✔	Use Unicode in cookies so they can be read correctly by applications.
✔	Use IDNA 2008 Protocol and Tables documents: • RFC 5891

	• RFC 5892
✖	Don't use IDNA 2003; in nearly all cases it has been superseded by IDNA 2008.
✖	Do not automatically assume that external APIs can consume data that has been NFKC[11] converted.
!	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 (RFC 5892), its IDNA tables must be derived from the version of Unicode that is supported more generally on the system. As with registration, the tables do not need to reflect the latest version of Unicode, but they must be consistent.
!	Validate the characters in labels only to the extent of determining that the U-label does not contain "DISALLOWED"[12] code points or code points that are unassigned in its version of Unicode.
✔	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[13] 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	
✔	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".

[11] **NFKC** (*Normalization Form Compatibility Composition)*: Characters are decomposed by compatibility, then recomposed by canonical equivalence. See: http://unicode.org/reports/tr15

[12] DISALLOWED: Code points that should not be included in IDNs. See: https://tools.ietf.org/html/rfc5892

[13] CONTEXTJ: Contextual Rule for Join controls. See: https://tools.ietf.org/html/rfc5892

✔	Ensure that the product or feature handles numbers correctly. For example, ASCII numerals and Asian ideographic number representations should all be treated as numbers.
!	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
!	Look for potential IRI[14] paths in unexpected places: • Single-label machine names regardless of loaded system codepage • Fully-qualified machine names regardless of loaded system codepage
✔	Use GB18030 (China) for Chinese language support[15] in addition to UTF-8.
!	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 list of allowed characters for IDNs: http://unicode.org/reports/tr36/idn-chars.txt Preventing certain IDNs or email addresses from being created in the first place can mitigate some likely security and accessibility concerns. (NOTE: Postel's Law 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 automatically assume that because a component does not directly call name-resolution APIs, or directly use email addresses, it does not mean that they do not affect it. Understand how network 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 network name: • Group policy • LDAP query • Configuration files

[14] IRI: Internationalized Resource Identifiers. See: https://www.ietf.org/rfc/rfc3987.txt

[15] GB 18030-2000 is a Chinese government standard that specifies an extended code page for use in the Chinese market. See: http://icu-project.org/docs/papers/unicode-gb18030-faq.html

	• Windows Registry • Transferred to/from another component/feature
✔	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

DNS Root Zone

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 the list is properly authenticated. You can obtain the root zone from any of the following links:

- http://www.internic.net/domain/root.zone
- http://www.dns.icann.org/services/authoritative-dns/index.html
- http://data.iana.org/TLD/tlds-alpha-by-domain.txt

Public Suffix List

The Public Suffix List (PSL), managed by volunteers of the Mozilla Foundation, provides an accurate list of domain name suffixes. This list is a set of DNS names or wildcards concatenated with dots and encoded using UTF-8. If you need to use the PSL as an authoritative source for domain names, your software must regularly receive PSL updates. Do not bake static copies of the PSL into your software with no update mechanism. You can use the link below to make your app download an updated list periodically. The list gets updated once per day from Github:

- https://publicsuffix.org/list/public_suffix_list.dat

Other Challenges

General

Variable encoding of IDNs	In some applications, IDNs are encoded: • In Punycode, as per IDNA, if the name is identified as an Internet name, BUT • In UTF-8, if the name is identified as a name on the local area network ("intranet")
Mechanism to detect and convert charsets	Some older email applications were encoded in a local code page and did not have a set mechanism for detecting and converting charset as needed. This was especially true for the email header (TO, CC, BCC, Subject).
Failure to handle non-DNS protocols	Some applications that do IDNA (for example, IE7+) break for non-DNS protocols. This could affect accessing resources using non-DNS protocols.
Mechanism to manage multiple email addresses into a single user identity	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 may still perceive these emails to pertain to different identities.

Tip for software developers	
✔	When allowing a user to generate a domain name or email address, consider avoiding the use of visually confusing characters to prevent homograph attacks. Use only this list of allowed characters for IDN: http://unicode.org/reports/tr36/idn-chars.txt

IDN-Style Email and Why It Is Not the Same as EAI

EAI is defined as using Unicode only; A-Labels (Punycode) 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):

Not all software will treat these two IDN-Style emails as functionally equivalent

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

However, some software will not robustly treat these addresses as equivalent, even though are both valid, because there is no requirement for software to process 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 even the local part of the email address using Punycode, creating something that looks like an A-LABEL in the local part of the address. This is not allowed under the existing RFCs, and is very likely to result in failures to receive email by certain systems and to generate searching and sorting difficulties as explained above.

Never convert the local part of an email address using Punycode

✔ 用戸@example.みんな

✖ xn--youq53b@example.xn--q9jyb4c

Robust UA-ready software and services may be able to handle and treat all these formats identically, even those which are not RFC-compliant. Nevertheless, UA-ready software should not generate true EAI email addresses only.

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 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.

Linkification is the action where an application accepts a string and dynamically determines whether it should create a hyperlink to an Internet Location (URL) or an email address (mailto:)

Linkification uses algorithms and rules created by software developers to determine whether a string should be deemed a link – or not. 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. "http://", 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 //)

Example String	Expected Behavior/ Result
http://example.a	No linkification because ICANN Policies require TLD to be at least two characters. NB: This syntax could be supported within an internal network.
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[16]
label@example.com	Create mailto: label@example.com because protocol is implied.

3. Map the Ideographic Full Stop "。" (U+3002) to Full Stop "." (U+002E) (e.g. http://田中。com à http://田中.com) if string is otherwise well formed.

4. If TLDs are used as a 'special suffix' to determine linkability, then all TLDs must be included. A list of valid TLDs should be updated dynamically on a frequent basis.

[16] Note: it might be the case that the actual website requires that end users type https:// instead of http://. If this is the case, then the hyperlink may not resolve or may return an error page.

Part 3: Advanced Topics

Complex Scripts

The details of complex scripts may not be of interest to those who are not developers creating their own string parsing 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.

Right to Left Languages and Unicode Conformance

Most scripts display characters from left to right when text is presented in horizontal lines. However, there are also several scripts, such as Arabic or Hebrew, where 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**[17] assigned to the phrase will determine the order in which text is displayed.

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). Letters from right-to-left scripts are strongly typed as RTL (right-to-left). 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 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 with vague directionality. 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** is indeterminate without context. Some examples include:

[17] 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` attribute.

- 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 character between two RTL characters will be treated as a RTL character itself, and will have the effect of extending the directional run:

- نطاق.مثال

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:

- example. مثال

Unless a directional override is present **numbers** are always encoded (and entered) big-endian[18], and the numerals rendered LTR. The weak directionality only applies to the placement of the number in its entirety.

To see the Bidi algorithm in detail, go to: http://unicode.org/reports/tr9/tr9-11.html

The Bidi Rule for Domain Names

A **Bidi domain name** is one that contains at least one RTL label. There is a rule that determines the conditions to be met for the labels in Bidi domain names. This rule can be found on Section 2 of RFC 5893: https://tools.ietf.org/html/rfc5893

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 diagraphs in English

ch (as in *church*)	*th* (*then*)	*sh* (*shoe*)
ph (*phone*)	*th* (*think*)	

Some digraphs are fully joined as **ligatures**. In writing and typography, a ligature happens 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").

[18] "Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the 'big end' (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the 'little end' (least significant value in the sequence) is stored first."
Source: http://searchnetworking.techtarget.com/definition/big-endian-and-little-endian

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: https://tools.ietf.org/html/rfc5894

The Unicode Consortium lists two main strategies to determine the joining behavior of a particular character after applying the Bidi algorithm:

- "When shaping, an implementation can refer back to the original backing store to see if there were adjacent ZWNJ or ZWJ[19] 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."[20]

In the absence of care by registries about how strings that could have different interpretations under IDNA2003 and the current specification are handled, it is possible that the differences could be used as a component of name-matching or name-confusion attacks. Such care is therefore appropriate.

To learn more about joiners, see Section 4.3 of RFC 5894: https://tools.ietf.org/html/rfc5894

Homoglyph and Confusingly Similar Characters

Homoglyphs are characters that, due to similarities in size and shape, might appear identical at first glance.

Examples of homoglyphs

Cyrillic character **а**	=	Unicode number 0430
Latin character **a**	=	Unicode number 0061

To prevent confusingly looking domain names being registered, registries can use the "homoglyph bundling" procedure.[21]

Homoglyph bundling is when you register an IDN and the registration system automatically bundles all the homoglyphs of that name (if there are any). This means that several domain names are bundled at one time, and none of the other domain names in that bundle can be registered.

Homoglyph bundling is a good practice for registries to avoid possible phishing practices that intend to trick the user with visually confusing characters.

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

[19] To learn more about ZWNJ/ZWJ, go to: http://www.unicode.org/L2/L2005/05307-zwj-zwnj.pdf
[20] Source: Mark Davis, Aharon Lanin, Andrew Glass. 2015. *Unicode*. http://unicode.org/reports/tr9

[21] https://www.icann.org/resources/pages/idn-guidelines-2011-09-02-en

- http://www.unicode.org/reports/tr39/#Confusable_Detection

To see a list of homoglyphs, go to:

- http://homoglyphs.net

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 and Case Folding

Normalization

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

- Canonical (NFD)
- Compatibility (NFK)

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

Examples of canonically equivalent characters

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

Compatibility equivalents are sequences which can have different appearances, but in some contexts the same meaning. It is a weaker type of equivalence between characters or sequences of characters.

Examples of compatibility equivalent characters

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 necessarily true.

To avoid interoperability problems arising from the use of canonically equivalent, yet different, character sequences, the W3C recommends using Normalization Form C[22] for all content.

To see a list of all characters that may change in any of the Normalization Forms, go to: http://www.unicode.org/charts/normalization

Some other points to note:

- Only strings NOT transformed by NFKC[23] are valid.
- When two applications share Unicode data, but normalize them differently, errors and data loss can occur.
- Normalization Forms must remain stable over time. In other words, a string must remain normalized under all future versions of Unicode (backward compatibility).

Tip for software developers

✖	Don't normalize by converting to uppercase, or ignoring non-spacing 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.

To learn more about Normalization Forms go to: http://www.unicode.org/reports/tr15

Case Folding

Case folding is the process of making two texts, which differ in case but are otherwise "the same", identical. Mapping [a-z] to [A-Z] works for most simple ASCII-only text documents. However, it begins to break down with languages that use additional characters.

Unicode defines the default case fold mapping for each Unicode code point. There are **common** and **full case fold mappings**:

- **Common fold mappings** are those that have a simple, straightforward mapping to a single matching (mainly lowercase) code point
- **Full fold mappings** are those that would normally require more than one Unicode character

One important consideration, according to the W3C,[24] is whether the values are restricted to the ASCII subset of Unicode or if the vocabulary permits the use of characters (such as accents on Latin letters or a broad range of Unicode including non-Latin scripts) that potentially have more complex case folding requirements.[25]

[22] NFC: Canonical Decomposition, followed by Canonical Composition.

[23] NFKC: Compatibility Decomposition, followed by Canonical Composition.

[24] W3C: The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff and the public work together to develop Web standards. See: https://www.w3.org

[25] Source: A Phillips. 2015. *Character Model for the World Wide Web: String Matching and Searching*. https://www.w3.org/TR/charmod-norm

Tip for software developers	
✔	Consider Unicode Normalization in addition to case folding.

To learn more about Unicode normalization, see:

- http://www.w3.org/TR/charmod-norm
- http://unicode.org/reports/tr15

For recommendations about case folding, go to:

- https://www.w3.org/International/wiki/Case_folding

Part 4: Glossary and Other Resources

Glossary

A-label	The ASCII-compatible encoded (ACE) representation of an internationalized domain name, e.g. how it is transmitted internally within the DNS protocol. A-labels always commence with the prefix "xn--". Contrast with U-label.
ACE prefix	ASCII Compatible Encoding Prefix.
ASCII Characters	American Standard Code for Information Interchange. These are characters from the basic Latin alphabet together with the European-Arabic digits. These are also included in the broader range of "Unicode characters" that provides the basis for IDNs.
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	A code point or code position is any of the numerical values that make up 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 translating readable host names into numeric IP addresses.
EAI	Email Address Internationalization is an email address that requires the use of Unicode in all parts of the email address.
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 characters used in the local representation of languages that are not written with 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 used in the local representation of languages that are nor written with the twenty-six letters of the basic Latin alphabet “a-z”. Examples: • .рф (Russia) • .مصر (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).
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 to represent Unicode with the limited character subset of ASCII supported by the Domain Name System. Punycode is intended for the encoding of 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. Conversions between U-labels and A-labels are performed according to the Punycode specification [RFC3492].
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 and supports up to 4 bytes for each character. See: http://unicode.org
UTF	Unicode Transformation Format. It is a way of transforming Unicode code points into 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 (M^3AAWG) 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, a full-time staff, and the public work together to develop Web standards. See: https://www.w3.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

PUNYCODE 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
IDN RFCs	

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
Unicode Technical Standard #46: Unicode IDNA Compatibility Processing	This specification defines a mapping consistent with the normative requirements of the IDNA 2008 protocol, and which is as compatible as possible with IDNA 2003. For client software, this provides behavior that is the most consistent with user expectations about the handling of domain names with existing data. http://unicode.org/reports/tr46/

Online Resources

APIs	Windows APIs https://www.msdn.microsoft.com/enus/library/windows/desktop/ff818516%28v=vs.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
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 BG18038-2000 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

Acknowledgements

The authors gratefully acknowledge the following people for their contributions and collaboration on this document:

Eleeza Agopian

Gwen Carlson

Edmon Chung

Samantha Dickinson

Don Hollander

Chantal Lebrument

Antonietta Mangiacotti

Richard Merdinger

Ram Mohan

David Morrison

Carolyn Nguyen

Michael D. Palage

Kurt Pritz

André Schappo

Zheng Song

Lars Steffen

Andrew Sullivan

Dennis Tan

Winnie Yu