Abstract

In the 1970s the characters available for registering domain names was limited to ASCII characters (letters a-z, digits 0-9 and the hyphen "-"). Today the majority of Internet users are non-English speakers, however the dominant language used on the Internet is English. In order to work towards the internationalization of the Internet, in 2003 the IETF started releasing standards providing technical guidelines for the deployment of Internationalized Domain Names (IDN) through a translation mechanism to handle non-ASCII characters. A decade later, and thanks to the ICANN New TLD Program, more than a thousand TLDs have been released. In spite of all these efforts, much software and many applications are still not Universal Acceptance-ready, causing problems to the users whose native language includes non-ASCII characters.

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.

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 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 the Internet Corporation for Assigned Names and Numbers (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.

This document gives a broad introduction to Universal Acceptance for Software Developers, CTOs and the technical community in general. In this text they can find information from basic definitions, good practices and advanced topics regarding Universal Acceptance.

The UASG and the community are available to provide advice to software developers and implementers on what is needed. You can contact us to share your ideas and suggestions on the topic at: tld-acceptance@icann.org and/or join the Universal Acceptance discussion at http://tinyurl.com/ua-discuss.

To learn more about the effort, visit us at: http://www.icann.org/universalacceptance.

Luisa Villa 2/24/2016 9:26 PM
Comment [1]: Registered? Well formed? Active?

Luisa Villa 2/24/2016 9:08 PM
Comment [2]: Length varies

Luisa Villa 2/26/2016 5:15 PM
Deleted: t

Luisa Villa 2/26/2016 5:17 PM
Deleted: is more than three characters in length

Luisa Villa 2/24/2016 9:09 PM
Comment [3]: The UASG is a Community Initiative supported by ICANN

Luisa Villa 2/26/2016 5:36 PM
Deleted: The Internet Corporation for Assigned Names and Numbers (ICANN) Universal Acceptance Steering Group (UASG), a community-led, industry wide initiative

Luisa 3/4/2016 5:41 PM
Deleted: happy to help

Contents

Luisa Villa 3/5/2016 1:59 PM
Deleted: 5

Luisa Villa 3/5/2016 1:59 PM
Deleted: 6

Luisa Villa 3/5/2016 1:59 PM
Deleted: 7

Luisa Villa 3/5/2016 1:59 PM
Deleted: 7

Luisa Villa 3/5/2016 1:59 PM
Deleted: 8

Luisa Villa 3/5/2016 1:59 PM
Deleted: 8

Luisa Villa 3/5/2016 1:59 PM
Deleted: 8

Luisa Villa 3/5/2016 1:47 PM
Deleted: 9

Luisa Villa 3/5/2016 1:59 PM
Deleted: 9

Luisa Villa 3/5/2016 1:59 PM
Deleted: 9

Luisa Villa 3/5/2016 1:59 PM
Deleted: 9

Luisa Villa 3/5/2016 1:59 PM
Deleted: 9

Luisa Villa 3/5/2016 1:59 PM
Deleted: 10

Luisa Villa 3/5/2016 1:59 PM
Deleted: 10

Luisa Villa 3/5/2016 1:59 PM
Deleted: 11

Luisa Villa 3/5/2016 1:59 PM
Deleted: 12

Luisa Villa 3/5/2016 1:59 PM
Deleted: 15

Luisa Villa 3/5/2016 1:59 PM
Deleted: 15

Luisa Villa 3/5/2016 1:59 PM
Deleted: 15

Luisa Villa 3/5/2016 1:59 PM
Deleted: 16

Luisa Villa 3/5/2016 1:59 PM
Deleted: 16

Luisa Villa 3/5/2016 1:59 PM
Deleted: 16

Luisa Villa 3/5/2016 1:59 PM
Deleted: 19

Luisa Villa 3/5/2016 1:59 PM
Deleted: 20

Luisa Villa 3/5/2016 1:59 PM
Deleted: 20

Luisa Villa 3/5/2016 1:59 PM
Deleted: 20

Luisa Villa 3/5/2016 1:47 PM
Deleted: 23

Luisa Villa 3/5/2016 1:59 PM
Deleted: 25

Luisa Villa 3/5/2016 1:59 PM
Deleted: 25

Luisa Villa 3/5/2016 1:59 PM
Deleted: 25

Luisa Villa 3/5/2016 1:59 PM
Deleted: 26

Luisa Villa 3/5/2016 1:59 PM
Deleted: 27

Luisa Villa 3/5/2016 1:59 PM
Deleted: 27

Luisa Villa 3/5/2016 1:59 PM
Deleted: 27

Luisa Villa 3/5/2016 1:59 PM
Deleted: 28

Introduction

For the purpose of complying with the ever-changing Internationalization of the Internet, it is necessary to be aware and up-to-date about what is changing and most importantly how to properly address these changes. In October 2009 ICANN's Board of Directors approved the introduction of new IDN ccTLDs (e.g., offering bj.中國 as well as bj.cn) and in June 2011, they approved and authorized the launch of the new TLD program that included new ASCII as well as internationalized domain name (IDN) top-level domains. Since the introduction of IDN ccTLDs into production in May 2010 and the March 2013 release of the new generation of TLDs into production, the pace of introduction of new TLDs into the root zone dramatically increased. New software must be built and, old software and applications need to be updated to keep pace with this new TLD world and be able to achieve a state of 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.

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.

Luisa 3/4/2016 6:10 PM
Deleted: which

Luisa 3/4/2016 6:11 PM
Deleted: has occurred at a fast pace

Luisa 3/4/2016 6:12 PM
Deleted: comply

Luisa 3/4/2016 6:12 PM
Deleted: to

Luisa 3/4/2016 6:13 PM
Comment [4]: Kurt Comment: Potentially add: "where all validly addressed emails and all valid web addresses reach their expected destination."

With this document we want to provide the technical community a guide that serves as a reference to support them in the developing of Universal Acceptance-ready software. This document is divided into four parts.

In the first section we explain baseline concepts of Universal Acceptance such as what is a Domain Name and Domain Name System, ASCII and Unicode, Punycode, Email Address Internationalization, and other basic concepts.

In the second part, we describe the 5 criteria of Universal Acceptance as well as the good practices for each of these criteria. In this second part you can also find user scenarios and nonconformance practices to Universal Acceptance, technical requirements and current challenges.

In the third section we explore advanced topics such as right-to-left scripts, the "bidi algorithm", Normalization and Case Folding.

Luisa 3/4/2016 6:15 PM
Deleted: go into

Finally, in the last part of this document you can find the glossary along with other additional useful online resources.

Problem Statement

Since the first reported descriptions of universal access to data and applications via a distributed network of computers emerged in the early 1960's, what has evolved into today's Internet is a network of networks that has been under continual evolution and change. The technologies that comprise the naming components of the Internet are no exception. Since the earliest .COM registration, SYMBOLICS.COM, in

Luisa Villa 2/24/2016 9:22 PM
Comment [5]: 2001, ICANN. TLD 2-3 characters. Problem started. .biz not recognized. Use of statics lists.

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. It is imperative that these changes are understood and accommodated throughout the Internet and related industries to ensure that the value found through the Internet continues to grow correspondingly.

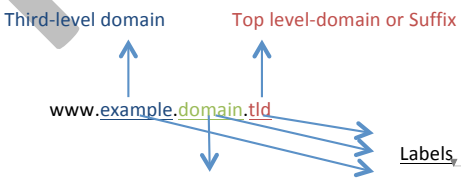
Many applications and services are not being updated to manage these new TLDs therefore affecting the user experience.

In recent years a great number of new TLDs with ASCII characters and IDN top-level domains (examples include .nyc, .hsbc, .eco, and .ストア) have been released by ICANN. However many applications and services are not being updated to manage these new TLDs therefore affecting the user experience. Valid email addresses not being accepted, domain names taken as a search term in the address bar of the browser and requiring users to enter punycoded text are just few of the problems users encounter with software that is not Universal Acceptance-ready. The naming landscape is growing rapidly and the response to this change has not been fast enough. It is not only imperative to create awareness in the technical community and in particular to Software Developers and CTOs but it is also necessary to create and provide more documentation that can be used as a reference for the implementation of these changes.

Part 1: Baseline Concepts of Universal Acceptance

Domain Name

A domain name is a dotted text string used as a human-friendly technical identifier for computers and networks on the Internet, e.g. “www.domain.tld”. Each dot represents a “level” in the Domain Name System (DNS) hierarchy. A Top-Level Domain (TLD) is often called the “suffix” at the end of a domain name (“.tld” in the above example). The individual words or characters between the dots are called **labels**. For those languages or scripts that are written from left to right, the label furthest right represents the top-level domain. The second label from the right represents the second-level domain. Any labels to the left of the second-level domain are considered subdomains of the second-level domain (sometimes called third-level domains).



Languages or scripts written from right to left will be discussed later in this document.

Domain Name System (DNS)

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

Unknown
Deleted:

Top Level Domains (TLDs)

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

Examples of common TLDs

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

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

Examples of ccTLDs

- China = .cn
- Germany = .de
- United States = .us

Generic Top Level Domains (gTLDs)

Starting in 2013, ICANN (the organization responsible for the creation and maintenance of TLD assignments) has generated a large number of new TLDs. These new TLDs can be used for brands, communities of interest, geographic communities (cities, regions) and others that are generic. Nevertheless, all of these new TLDs are known as Generic Top Level Domains (gTLDs). Sometimes new gTLDs are known as nTLDs.

Examples of new gTLDs

- .app
- .lawyer
- .shopping
- .panasonic
- .osaka

Character Sets and Scripts

Domain names, of course, must be written in a commonly used language. 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 are recognized by humans, however they are not useful to computers, which only process numbers. In order to make a computer be able to use a script, (e.g., to resolve a web address) that script needs to be encoded so that the computer can use it.

The mechanism for this is called a coded character set (often abbreviated CCS[1]).

To resolve this, a character mapping (also called coded character set or code page) can be created to associate characters with specific numbers. Many different code pages have been created over time for different purposes, but for this topic we will focus on only two.

[1] There are subtleties and details to the terms that are not directly germane to our current topic; but the interested reader is referred to https://www.rfc-editor.org/rfc/rfc6365.txt as a useful starting point.

Luisa Villa 2/24/2016 9:32 PM
Comment [6]: Michael comments will be sent to the list with editing proposal.

Luisa Villa 2/24/2016 9:35 PM
Deleted: s

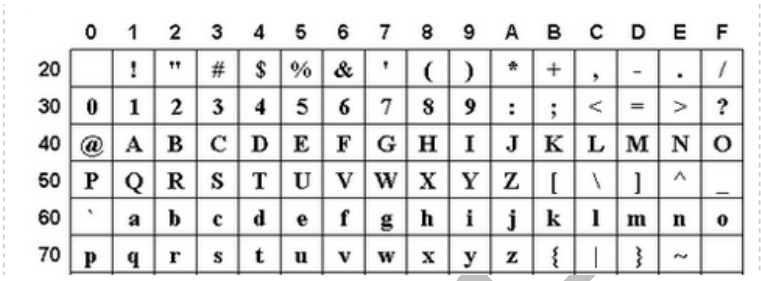
Luisa 3/4/2016 10:54 PM
Deleted: use

Luisa 3/4/2016 11:56 PM
Deleted: ,

ASCII & Unicode

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

ASCII - ISO 8859-1 (Latin-1) Table [2]



	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	{	|	}	~	

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

Because most writing systems do not use the Latin character set, alternate encodings have subsequently been adopted. Unicode (also known as the Universal Coded Character Set, or UCS) is capable of encoding more than 1 million items. 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).

Internationalized Domain Names (IDNs) & Punycode

By using Unicode, domain names can contain non- ASCII characters. Domain names using non-ASCII characters are called Internationalized Domain Names (IDN).

Since the DNS itself previously only used ASCII (see http://tools.ietf.org/html/rfc6055#section-3 for current status), an additional encoding was created to allow non-ASCII Unicode code points to be converted into ASCII strings.

The algorithm that implements this Unicode-to-ASCII encoding is called Punycode, and the output strings are called A-Labels. A-Labels can be distinguished from an ordinary ASCII label because they always start with the characters "xn--" (which is called the ACE prefix[3]). The Punycode transformation is reversible. It can transform from Unicode to an A-Label and also from an A-label back to Unicode.

The only RFC-defined use of the Punycode algorithm is for expressing internationalized domains. This has not stopped some developers from applying it to other scenarios rather than implementing Unicode.

The internationalized portion of a domain name can be in any level, not just the TLD but also the other labels.

Luisa 3/5/2016 9:53 AM
Deleted: M

Luisa 3/5/2016 9:54 AM
Deleted: items

Luisa Villa 2/24/2016 9:44 PM
Comment [7]: Not allowed non-ASCII characters. Explain (Apendix?) Not every non-ascci character is an IDN.

Luisa Villa 2/24/2016 9:39 PM
Deleted: any

Luisa 3/5/2016 9:56 AM
Deleted: which

[2] Image. California State University, Los Angeles http://web.calstatela.edu/

[3] ASCII Compatible Encoding (*ACE*) prefix, is used to distinguish Punycode encoded labels from ordinary ASCII labels.

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)

Email addresses & Email Address Internationalization (EAI)

Email addresses contain two parts: a local part (the username, before the "@" character) and a domain (after the "@" character). The domain part can contain can 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 using nTLDs and IDNs

- user@example.lawyer (uses new gTLD)
- 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 4 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 name or network path. For example, typing "www.icann.org" into an email message may result in a clickable link to http://www.icann.org being automatically created if the app treats "www." as a special prefix or ".org" as a special suffix.

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

Part 2: Universal Acceptance in Action

Five Criteria of Universal Acceptance

As mentioned above, *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 5 criteria are described below.

Accept

Applications and services allow domain names and email addresses to be entered into user interfaces and/or received from other applications and services via APIs. If this process includes a validation to verify that the data has been presented in a supported format, meeting this criterion will depend on the application or service being aware of current valid formats, which include different lengths and character sets than was the case previously.

Validation

Validation is intended to ensure that the entered information is either valid or at least definitely not invalid. For domain names and e-mail addresses, many programmers have been using some heuristics (e.g. checking that a top-level domain has the "correct" number of letters, or that the letters are from the ASCII character set). 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 top-level domain names is growing, and the top-level domain name can be up to 63 characters.

Luisa Villa 2/24/2016 9:51 PM
Comment [8]: Foot note. Example RTL, LTR email address.

Luisa 3/5/2016 9:57 AM
Comment [9]: Kurt comment: CLARIFY: Can the A-label be used instead? And why is the U-label preferred?

Luisa Villa 2/24/2016 10:01 PM
Deleted: Universal Acceptance is

Luisa 3/5/2016 10:01 AM
Comment [10]: Kurt comment: I thing the explanations in the brochure are clearer compared to these explanations. Maybe do a mashup or start with the brochure explanations and augment

Luisa Villa 2/24/2016 10:02 PM
Comment [11]: Overlaps with Validation and Processing

Luisa Villa 2/24/2016 10:04 PM
Deleted: valid

Luisa 3/5/2016 10:01 AM
Comment [12]: Kurt comment: Does this validation belong in "Accept," i.e., are there potentially two validations Or should this sentence be moved to Validation below for clarity?

Acceptance is a strong word, maybe we should say that acceptance is merely the accepting of what is typed into the API so readers don't think there is some sort of real acceptance process going on here.

Luisa Villa 2/24/2016 10:05 PM
Deleted:

Store

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 either in RFC-defined formats, or in alternate formats which can be easily translated to and from RFC-defined formats (the latter is much less desirable). Although RFCs require the use of UTF-8, other formats may be encountered in legacy code. See Good Practices, below.

Process

Processing means using domain names and email strings in a feature. Additional validation may occur during processing. There is no limit to the number of ways that domain names and email addresses could be processed (e.g. "identify all the people associated with New Zealand because they have a name with a `.nz` ccTLD", or "identify all the pharmacists because they have a `user@*.pharmacy` email address", or firewalls that might filter DNS requests that don't apply to their policies.)

Display

Displaying domain names and email addresses are usually straightforward when the scripts used are supported in the underlying OS and when the strings are stored in Unicode; if these conditions are not met, application-specific transformations may be required.

User Scenarios

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

Here 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 address customername@example.brand; web apps accept these new "@example.brand" email addresses as valid as they would with legacy TLDs (e.g. 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. Even though the TLD is new, any browser the user wishes to use displays the web address in its native form and access the site as the user expects (and 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. 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 older 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. 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. Even though some of the email addresses are EAI, the user can send to and receive from any email address and using any email client.

Luisa Villa 2/24/2016 10:06 PM
Deleted: proprietary

Luisa Villa 2/24/2016 10:06 PM
Deleted: Best

Luisa Villa 2/24/2016 10:09 PM
Deleted: in

Luisa Villa 2/24/2016 10:10 PM
Comment [13]: .pharmacy?

Luisa 3/5/2016 10:11 AM
Deleted: `cist`

Luisa Villa 2/24/2016 10:08 PM
Deleted: conventions

Luisa 3/5/2016 10:06 AM
Deleted: which

Luisa Villa 2/24/2016 10:12 PM
Deleted: .,

Luisa 3/5/2016 10:12 AM
Deleted: will

Luisa Villa 2/24/2016 10:20 PM
Comment [14]: remove will?

Luisa 3/5/2016 10:13 AM
Deleted: will

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. Even though the script used in the email address is different than the language settings of both their operating system and their browser, the bank or airline site accepts the EAI identity exactly as if it were a non-EAI identity.

Dynamically creating a Hyperlink in an Application

A user types a web address into a document or email message. 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. The tools used by the developers include libraries which enable Universal Acceptance by supporting Unicode, IDNs and EAI.

Nonconformance to Universal Acceptance Practices

The following are considered **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 suffixes even though new suffixes are regularly being added
- Exposing internal use of Punycoded text to users (e.g. 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 new TLDs

Technical Requirements for "UA-readiness"

High level requirements

An application or service that supports universal acceptance:

- Supports all domain names regardless of length or character set. To learn more see RFC 5892 https://tools.ietf.org/html/rfc5892)
- Allows international characters valid for domain names and email addresses (i.e., all Unicode code points)
- Can correctly render all code points in Unicode strings. (See RFC 3490 https://www.ietf.org/rfc/rfc3490.txt)
- Can correctly render right-to-left strings such as those in Arabic and Hebrew. (See more about right-to-left scripts at RFC 5893 at https://tools.ietf.org/html/rfc5893)
- Can communicate data between applications and services in formats that support Unicode and are convertible to/from UTF-8. To learn more about UTF-8 go to RFC 3629 https://tools.ietf.org/html/rfc3629)
- Offers public APIs that support Unicode & UTF-8
- Offers private APIs that support Unicode & UTF-8 (these private APIs apply only to inter-service calls by the same vendor)

Luisa Villa 2/24/2016 10:22 PM
Deleted: . R

Luisa 3/5/2016 10:19 AM
Comment [15]: Kurt comment: I am not sure what this means

Luisa Villa 2/24/2016 10:24 PM
Deleted: .

Luisa Villa 2/24/2016 10:26 PM
Comment [16]: Define Universal Acceptance-Readiness

Luisa 3/5/2016 10:20 AM
Deleted: which

Luisa Villa 2/24/2016 10:26 PM
Deleted:

Luisa Villa 2/24/2016 10:26 PM
Deleted:

Luisa Villa 2/24/2016 10:26 PM
Comment [17]: Clarify

Luisa 3/5/2016 10:22 AM
Comment [18]: Kurt comment: I am dating myself but does IDNA accept ALL Unicode code points?

Luisa 3/5/2016 10:23 AM
Deleted: which

Luisa Villa 2/24/2016 10:27 PM
Deleted:

Luisa Villa 2/24/2016 10:31 PM
Deleted:

Luisa 3/5/2016 10:23 AM
Deleted: which

Luisa 3/5/2016 10:23 AM
Deleted: which

Luisa Villa 2/24/2016 10:28 PM
Comment [19]: Good practices?

- 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)
- Supports all domain name strings in the authoritative ICANN TLD list and the community-served Public Suffix List regardless of length or character set
- Can send email to and receive from recipients regardless of domain or character set (See RFC 6530 at https://tools.ietf.org/html/rfc6530)

- Treats EAI addresses the same way as their Punycoded equivalents (IDN email format) ASCCI@A-LABEL

Developer Considerations

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

Robustness (Postel's law)

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

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

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

Good Practices for developing and updating software affecting Universal Acceptance

Accept

- Always offer Unicode equivalents. Users should be allowed, but not required, to enter ASCII Compatible Encoded (AKA "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 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 and strings up to 256 characters

Validate

- Validate only to the minimum extend necessary. **Don't validate at all unless it's 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 such as http://www.internic.net/domain/root.zone or http://www.dns.icann.org/services/authoritative-dns/index.html or 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.
  - Require repeated entry of an e-mail 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. [RFC5892]

Luisa 3/5/2016 10:23 AM
Deleted: which

Luisa Villa 2/24/2016 10:32 PM
Comment [20]: DEFINE

Luisa Villa 2/24/2016 10:31 PM
Deleted: <#>Can receive email from senders regardless of domain or character set

Luisa Villa 2/24/2016 10:44 PM
Comment [21]: Improve, clarify

Luisa Villa 2/24/2016 10:38 PM
Deleted: developing and updating Universal Acceptance Software

Luisa Villa 2/24/2016 10:51 PM
Comment [22]: Listen to the call recording (Richard comments)

Luisa Villa 2/24/2016 10:53 PM
Deleted: easiest

Luisa Villa 2/24/2016 10:55 PM
Deleted: Do r

Luisa 2/22/2016 2:49 PM
Comment [23]: Don Comment: Can we add an example of syntax validation code in an appendix or footnote or endnote, please

- Limit validation of labels itself to a small number of whole-label rules defined in the RFCs. [RFC5894]
- If a string resembling a domain name contains the ideographic full stop character '。', convert it to a '.'.
- 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 in 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 origin as an address.
- If you *don't* store in Unicode, must be able to match strings in multiple formats (e.g. a search for `example.みんな` should also find `example.xn--q9jyb4c`.)

Process

- Do ensure all server responses should have the Unicode specified in the content type.
- Do specify Unicode in the web server http header and directly in a web file. Every web file should include the UTF-8 charset. It is important to ensure that the encoding is specified on every response.
- Consider all end-to-end scenarios before converting A-Labels (Punycode) to U-Labels and vice versa during procesing. 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 in both formats.
- Do ensure that the product or feature handles sort order, searches, and collation according to locale/language specifications, and that it addresses multilanguage searching and sorting.
- Do use MIME for email encoding.
- Don't use URL-encoding for domain names (e.g. example.みんな is correct, but 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 Internationalizing Domain Names in Applications (IDNA) Protocol [RFC5891] and Tables [RFC5892] documents for Internationalized Domain Names (IDNs).
- Process in UTF-8 format wherever possible.

- Ensure that the product or feature handles numbers as expected. For example, ASCII numerals and Asian ideographic number representations should be treated as numbers. [RFC5892]
- 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, or when operating a registry, consider the languages supported and make sure OS 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. Use Punycoded text to the user only when it provides a benefit.
- Consider 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, and 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. Learn more at RFC 5892: https://tools.ietf.org/rfc/rfc5892.txt.

Unicode

- Use supported Unicode-enabled APIs - don't spin your own 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.
- Recognize that mixed-script strings will become more common. Don't assume that mixed-script strings are intended for malicious purposes, such as phishing, and if your user interface calls such strings to the user's attention, be sure that it does so in a way that is not prejudicial to users of non-Latin scripts.
- Use Unicode in cookies so they can be read correctly by applications.
- Use IDNA 2008 Protocol [RFC5891] and Tables [RFC5892] documents. Don't use IDNA 2003.

Luisa 3/5/2016 10:28 AM
Deleted: which

Luisa 3/5/2016 10:29 AM
Deleted: .

Luisa 3/5/2016 10:29 AM
Deleted: which

- Do not automatically assume that external APIs can consume data that has been NFKC[4] converted.
- Maintain IDNA and Unicode tables that are consistent with regard to versions, i.e., unless the application actually executes the classification rules in the Tables document [RFC5892], its IDNA tables must be derived from the version of Unicode that is supported more generally on the system. As with registration, the tables need not reflect the latest version of Unicode, but they must be consistent.
- 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.
- 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[5] 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."
- Ensure that the product or feature handles numbers correctly. For example, ASCII numerals and Asian ideographic number representations should all be treated as numbers.
- Upgrade your app and server/service together. If the server is Unicode and client is non-Unicode, or vice versa, data needs to be converted to each code page every time the data travels from server to client or vice versa.
- 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 paths in unexpected places
    - Single-label machine names regardless of loaded system codepage
    - Fully-qualified machine names regardless of loaded system codepage

Luisa 3/5/2016 10:30 AM
Deleted: which

[4] **NFKC** (*Normalization Form Compatibility Composition)*: Characters are decomposed by compatibility, then recomposed by canonical equivalence.

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

- Use GB18030 (China) for Chinese language support[6].
- Restrict the code points allowed when generating new domain names and email addresses.
  - All products that use email addresses must accept internationalized email addresses, allowing characters > U+007f. That is, no characters > U+007f are disallowed.
  - However, an app or service need not allow all of these characters when a user creates a new IDN or email address. Use only this allowed list of characters for IDN: http://unicode.org/reports/tr36/idn-chars.txt
  - Some likely security and accessibility concerns can be mitigated by preventing certain IDNs or email addresses from being created in the first place. (NOTE: Postel's Law would still require software to accept such strings if presented.)
- It is important to note that Universal Acceptance cannot always be measured through automated test cases alone. For example, testing how an app or protocol handles network resource may not always be possible and sometimes it is best to verify the compliance through functional spec review and design review.
- Don't automatically assume that because a component does not directly call name-resolution APIs, or directly use email addresses, it does not mean that it is not affected the by them.
  - Understand how network names are obtained by the component; it is not always through user interaction. Following are some examples on how the component can get a network name:
    - Group Policy
    - LDAP query
    - Configuration files
    - Registry
    - Transferred to/from another component/feature.
- 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 would be the DNS root zone itself. It is DNSSEC signed, so the list is properly authenticated. You can obtain the root zone at http://www.internic.net/domain/root.zone or from https://www.dns.icann.org/index.html%3Fp=196.html.

Luisa Fernanda Vill…, 2/20/2016 1:44 AM
Comment [24]: This link is broken. Do you have the correct link?

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 it is 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

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

Other Challenges

General

- In some applications IDNs are encoded in Punycode as per IDNA if the name is identified as an Internet name, but UTF-8 is used if the name is identified as an intranet name.
- Some older email applications were encoded in a local code page and they did not have a set mechanism for detecting and converting charset as needed. This was especially true for the email header (i.e. TO, CC, BCC, Subject).
- Some applications that do IDNA (e.g., IE7+) break for non-DNS protocols, and could affect accessing resources using non-DNS protocols.
- When allowing a user to generate a domain name or email address, consider avoiding the use of visually confusing characters to prevent homograph attacks. Use only this allowed list of characters for IDN: http://unicode.org/reports/tr36/idn-chars.txt.
- When a user is aliasing multiple email addresses it may be tricky to manage these addresses as a single user identity. Email programs can direct traffic to such aliases to the same mailbox, but the application will still perceive these emails to pertain to different identities.

"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 part 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):

- 用戶@example.みんな
- xn--youq53b@example.xn--q9jyb4c

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

UA-ready software and services should be able to handle and treat them the same. Nevertheless, UA-ready software should not generate email addresses that use an A-label- should support true EAI only.

Linkification challenges

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

Example string	Likely result
example.com	No change
www.example.com	Link to www.example.com
http://example.com	Link to http://example.com
http:example.com	No change
http://.com	Link to http://.com
example.news	No change
www.example.news	Link to www.example.news
http://example.news	Link to http://example.news
example.photography	No change

Luisa 3/5/2016 10:46 AM
Deleted: using

Luisa 2/22/2016 10:10 PM
Comment [25]: Don Comment: This should be expanded and clarified. Why are the above not equivalent?

Luisa 3/5/2016 10:48 AM
Comment [26]: Kurt comment: take this out?

Luisa 3/5/2016 10:49 AM
Deleted: IDN and EAI linkification may not

Luisa 2/22/2016 10:10 PM
Comment [27]: Don Comment: There needs to be some additional context setting here.

www.example.photography	Link to www.example.photography
http://example.photography	Link to http://example.photography
http://.photography	Link to http://.photography
田中.com	No change
www.田中.com	Link to www.田中.com
http://田中.com	Link to http://田中.com
田中。Com	No change
http://田中。com	No change
español.com	No change
www.español.com	Link to www.español.com
http://español.com	Link to http://xn--espaol-zwa.com/
http:///español.com	Link to http://xn--espaol-zwa.com/

Part 3: Advanced Topics

Complex Scripts

Right to left languages and Unicode conformance

- Most scripts display characters from left to right when text is presented in horizontal lines.
- 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 are 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* assigned to the phrase will determine the order in which text is displayed. It establishes a directional context that the bidi algorithm refers to at various points to choose how to handle the text.
- To know if a sequence is left-to-right or right-to-left character, each character in Unicode has an associated directional property. Most letters are ***s**trongly 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) Dubai - دبي (AL).
- Text with different directionality can be mixed in line. In that case 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 also fall within this category.

- The directionality of **Neutral characters** is indeterminable without context. Some examples include tabs, paragraph separators, and 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: امارات.دبي 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.امارات

- Unless a directional override is present **numbers** are always encoded (and entered) big-endian[7], and the numerals rendered LTR. The weak directionality only applies to the placement of the number in its entirety.

- **Explicit formatting characters**, are also referred to as "directional formatting characters", these are special Unicode sequences that direct the unicode algorithm to modify its default behavior. These characters can be subdivided into "marks", "embeddings", "isolates", and "overrides". Their effects continue until the occurrence of either a paragraph separator, or a "pop" character.

    - *Marks:* These characters are very light-weight codes. They act in the same way as right-to-left or left-to-right characters, with the exception that they do not have any other semantic effect. If a "weak" character is followed by another "weak" character, the algorithm will check for the first neighboring "strong" character. Sometimes this can lead to unintentional display errors. These errors are corrected with "marks". The mark (U+200E left-to-right mark (HTML `&#8206;` · `&lrm;` · LRM) or U+200F right-to-left mark (HTML `&#8207;` · `&rlm;` · RLM)) is to be inserted into a location to make an enclosed weak character inherit its writing direction.

RLM	Right-to-Left Mark	Right-to-left zero-width character
LRM	Left-to-Right Mark	Left-to-right zero-width character

    - Embeddings: An "embedding" indicates that a portion the of text is to be treated as directionally distinct. The text within the scope of the embedding formatting characters is not independent of the neighboring text. Also, characters within an embedding can affect the ordering of characters outside. As of Unicode 6.3, embedding is being discouraged in favor of "isolates".

[7] *"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".* To learn more go to: http://searchnetworking.techtarget.com/definition/big-endian-and-little-endian

RLE	Right-to-Left Embedding	Treat the following text as embedded right-to-left.
LRE	Left-to-Right Embedding	Treat the following text as embedded left-to-right.

- *Isolates:* The "isolate" directional formatting characters indicate that a portion of the text is to be treated as directionally isolated from its surroundings. As of Unicode 6.3, these are the formatting characters that are recommended in new documents. Isolates can be nested, and may be located within embeddings and overrides.
- *Overrides:* The "override" directional formatting characters allow for special cases, such as for part numbers (e.g. to force a part number made of mixed English, digits and Hebrew letters to be written from right to left). "Overrides" are recommended to avoid wherever possible. "Overrides" can be nested one inside another, and in embeddings and isolates.

RLO	Right-to-Left Override	Force following characters to be treated as strong right-to-left characters.
LRO	Left-to-Right Override	Force following characters to be treated as strong left-to-right characters.

- *Pops:* The "pop" directional formatting characters put an end to the scope of the most recent "embedding", "override", or "isolate".

PDF	Pop Directional Format	Restore the bidirectional state to what it was before the last LRE, RLE, RLO, LRO.

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

The "bidi" Rule for Domain Names

The section 2 of the RFC 5893 lists the following six conditions to be met for the labels in Bidi domain names. A “Bidi domain name” contains at least one RTL label.

1. The first character must be a character with Bidi property L, R, or AL. If it has the R or AL property, it is an RTL label; if it has the L property, it is an LTR label.
2. In an RTL label, only characters with the Bidi properties R, AL, AN, EN, ES, CS, ET, ON, BN, or NSM are allowed.
3. In an RTL label, the end of the label must be a character with Bidi property R, AL, EN, or AN, followed by zero or more characters with Bidi property NSM.
4. In an RTL label, if an EN is present, no AN may be present, and vice versa.
5. In an LTR label, only characters with the Bidi properties L, EN, ES, CS, ET, ON, BN, or NSM are allowed.
6. In an LTR label, the end of the label must be a character with Bidi property L or EN, followed by zero or more characters with Bidi property NSM.

- To learn more about the bidi rule go to: https://tools.ietf.org/html/rfc5893

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

- 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*).
- Common digraphs in English include *ch* (as in *church*), *ph* (*phone*), *sh* (*shoe*), *th* (*then*), and *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 character *æ* as used in English, in which the letters *a* and *e* are adjoined. Another example of a ligature is the ampersand (*&*).
- If ligatures and digraphs have the same interpretation in all languages that use a given script, Unicode normalization generally resolves the differences and make 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 the Nordic languages. In the Norwegian language the ligature "ae" is the 27th letter of its 29-letter extended Latin alphabet, which also happens to be the equivalent of the 28th letter of the Swedish alphabet (U+00E4 LATIN SMALL LETTER A WITH DIAERESIS). The same (U+00E4) character is also part of the German alphabet where, unlike in the Nordic languages, the two-character sequence "ae" is usually treated as a fully acceptable equivalent orthography for the "umlauted a" character (ä). The opposite though is not true, and those two characters cannot be combined into an "umlauted a" (ä).
- 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 characters.*[8] (To learn more about ZWNJ/ZWJ go to: http://www.unicode.org/L2/L2005/05307-zwj-zwnj.pdf)
    - *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.*[9]
- 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¨go to: https://tools.ietf.org/html/rfc5894#page-19

[8] Mark Davis, Aharon Lanin, Andrew Glass. 2015. *Unicode*. [ONLINE] Available at: http://unicode.org/reports/tr9/. [Accessed 11 February 16].

[9] Mark Davis, Aharon Lanin, Andrew Glass. 2015. *Unicode*. [ONLINE] Available at: http://unicode.org/reports/tr9/. [Accessed 11 February 16].

Homoglyph Bundling

- **Homoglyphs** are characters which, due to similarities in size and shape, might appear identical at first glance. For example, Cyrillic character **а** → Unicode number 0430 and Latin character **a** → Unicode number 0061.
- To prevent that confusingly looking domain names are registered, registries can use the "homoglyph bundling" procedure.

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.

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 (NF) and 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:

  U+006E (the Latin lowercase "n") followed by U+0303 (the combining tilde "◌̃") = ñ

  U+00F1 (the 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.

  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 necessarily true.
- There are four Unicode Normalization forms:
  - **NFD** (*Normalization Form Canonical Decomposition)*: Characters are decomposed by canonical equivalence, and various combining characters are arranged in a defined order.
  - **NFC** (*Normalization Form Canonical Composition)*: Characters are decomposed and then recomposed by canonical equivalence.
  - **NFKD** (*Normalization Form Compatibility Decomposition)*: Characters are decomposed by compatibility, and various combining characters are arranged in a defined order.
    NFKC (*Normalization Form Compatibility Composition)*: Characters are decomposed by compatibility, then recomposed by canonical equivalence.
- Note that none of the Normalization Forms are closed under string concatenation.
- **Singletons** are characters which never remain in the text after normalization. One example is the omega symbol (Ω) Source: (2126)/ NFD and NFC: (03A9).

- **Canonical composites**, also known as precomposed characters are usually precomposed in the C forms and decomposed in the D forms. For example Ô source (00F4) /NFD O (006F) + ^ (0302) / NFC Ô (00F4).
- Normalization provides a uniform order for all D and C forms and a unique order for **multiple combining marks**. For example: Source Ṩ (1E69) / NFD S (0073) + ̣ (0323) + ̇ (0307) / NFC Ṩ (1E69).
- Many formatting distinctions are removed in the NFKC and NFKD forms, they use **compatibility composites** like in the example below:

Source	NFD	NFC	NFKD	NFKC
ﬁ	ﬁ	ﬁ	f i	f i
FB01	FB01	FB01	0066 + 0069	0066 + 0069

- The composition phase of NFC and NFKC are the same, only their decomposition phase is different, in this case NFKC applies compatibility decompositions.
- It is recommended by the W3C to use Normalization Form C for all content in order to avoid interoperability problems arising from the use of canonically equivalent, yet different, character sequences.
- Normalization Forms KC and KD may remove distinctions that are important to the semantics of the text. In this case it is best to think of these Normalization Forms as being like uppercase or lowercase mappings.
- To see a list of all characters that may change in any of the Normalization Forms please go to: http://www.unicode.org/charts/normalization/
- Don't normalize by converting to uppercase, or ignoring nonspacing characters, because this may also make sorting, data copy, data import and export, data retrieval by client applications rather difficult and may result in data loss or corruption.
- Only strings NOT transformed by NFKC 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).
- 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 identical, which differ in case but are otherwise "the same".
- 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.* The **common** fold mappings are those which have a simple, straight-forward mapping to a single matching (mainly lowercase) code point. The **full** fold mappings are those which would normally require more than one Unicode character.
- An example of a 'full' case fold mapping is the character ß U+00DF LATIN SMALL LETTER SHARP S, a letter that is commonly used in the German language. The '"full" mapping of this character is to two ASCII letters "s".
- Some languages need case-folding to be tailored to meet specific linguistic needs. One common example of this are Turkic languages written in the Latin script. The classic example, the Turkish

word "Diyarbakır" contains both the dotted and dotless letters i. When rendered into upper case, this word appears like this: DİYARBAKIR. Notice that the ASCII letter i maps to U+0130 LATIN CAPITAL LETTER I WITH DOT ABOVE, while the letter ı (U+0131 LATIN SMALL LETTER DOTLESS I) maps to the ASCII uppercase I.

- One important consideration according to the W3C 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.[10]

Best Practices for Case Folding

- Consider Unicode Normalization in addition to case folding. To learn more about "Unicode normalization" go to: http://www.w3.org/TR/charmod-norm/ or to http://unicode.org/reports/tr15/
- Normalize case in a language sensitive manner.
- Always use the language (locale) when case folding and look out for specific case folding idiosyncrasies.
    - If your comparison should be the same regardless of language or locale, always pass the US English or empty (root, C, POSIX, null) locale to your case-folding function.
    - If your application is comparing internal values for equality (as opposed to sorting lists or comparing values linguistically), you should use a consistent caseless compare function.

Open Issues

Topics for potential proposals to ecosystem, ICANN, IETF

- Due to differences in IDNA2003 and 2008, similar strings such as foosball.de and foSSball.de may or may not resolve to the same address. They might not even belong to the same owner!
- Can/should we encourage "bundling" at the registration level for compatibility? (I.e. should we require a registry to sell both to the same customer?)
- Should ICANN restrict the delegation of homograph domain names (at any level, not just TLD)?
- Define IDN-style email
- UTR#36 – does not discuss structured text?
- Structural separators like – and numerals – define behavior esp. for bidirectional
- Do tooltips correctly show the TLD for mixed RTL/LTR? (show visual example) – best practices?

Part 4: Glossary and other resources

Glossary

- **A-label:** The ASCII-compatible encoded (ACE) representation of an internationalized domain name, i.e. 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.

[10] Addison Phillips. 2015. *W3C*. [ONLINE] Available at: https://www.w3.org/TR/charmod-norm/. [Accessed 11 February 16].

- **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.
- **Brand Top-level Domain:** A Brand TLD is an innovative type of top level domain name (TLD) that is made possible through the implementation of ICANN's new gTLD Program. A Brand TLD provides the opportunity for branded corporations to use their corporate name as their website's top-level identifier instead of using a more traditional .com or .biz domain space.
- **ccSLD:** Country Code second-level domain.
- **ccTLD:** Country Code top-level domain. These two-letter domains correspond to a country, territory, or other geographic location. i.e. .de for Germany, .us for USA.
- **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).
- **Community TLD:** This is a TLD restricted to a specific community with high degree of social awareness. Examples of community TLDs include: .catholic, .thai, .aarp
- **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.
- Geographical TLD: This TLD represents a particular city or region; support of the local government is required for these TLDs, examples include: .nyc, .berlin, .tokyo.
- **FQDN:** A fully qualified domain name (FQDN) also referred to as an *absolute domain name*, is a domain name that specifies its exact location in the tree hierarchy of the Domain Name System (DNS). It specifies all domain levels, including the top-level domain and the root zone.
- **gTLD:** Most TLDs with three or more characters are referred to as "generic" TLDs, or "gTLDs". They can be subdivided into two types, "sponsored" TLDs (sTLDs) and "unsponsored TLDs (uTLDs).
- **IANA:** Internet Assigned Numbers Authority. Its functions includes the maintenance of the registry of technical Internet protocol parameters; the administration of certain responsibilities associated with Internet DNS root zone and the 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".
- **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”. For example, Russia .рф (Russia), مصر. (Egypt), السعودية. (Saudi Arabia).
- **IESG:** The Internet Engineering Steering Group (IESG) is responsible for technical management of IETF activities and the Internet standards process. The IESG is directly responsible for the actions associated with entry into and movement along the Internet "standards track," including final approval of specifications as Internet Standards.

- **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: T**he method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way.
- **Morpheme:** In linguistics, a morpheme is the smallest grammatical unit in a language. In other words, it is the smallest meaningful unit of a language.
- **Punycode:** It is a way 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:** A Registrar is a company where domain names are registered. The registrar keeps records of the contact information and submit the technical information to a central directory known as the "registry."
- **Registry:** A Registry is 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 letters or characters used in writing.
- **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. Some domain name registries introduce a second-level hierarchy to a TLD that indicates the type of entity intended to register an SLD under it. For example, in the .uk namespace a college or other academic institution would register under the `.ac.uk` ccSLD, while companies would register under `.co.uk`.
- **Sponsored TLD:** A sponsored TLD is a specialized top-level domain that has a sponsor representing a specific community served by the domain. The sponsor carries out delegated policy-formulation responsibilities over many matters concerning the TLD. **U-label:** The Unicode representation of an internationalized domain name, i.e. how it is shown to the end-user. Contrast with A-label.
- **UA-ready Software:** 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:** It is 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.
- **Unsponsored TLD:** Unsponsored Top-level domains are intended to be relatively large, generally available domains operating under policies established by the global Internet community directly through the ICANN process. Examples of unsponsored TLDs are .com, .net, .pro, .org.
- **UTF:** Unicode Transformation Format. It is a method of converting Unicode characters, which are 16 bits each, into 7- or 8-bit characters. UTF-7 converts Unicode into ASCII for transmission over 7-bit mail systems, and UTF-8 converts Unicode to 8-bit bytes.
- **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.

Cross-link to RFCs

RFC3492 (Punycode)

- ***Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)***

The RFC3492 describes Punycode as "*a simple and efficient transfer encoding syntax designed for use with Internationalized Domain Names in Applications (IDNA)*"[11] *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. Go to RFC 3492:* https://www.ietf.org/rfc/rfc3492.txt

RFC5890-94 (IDN)

- ***RFC5890: 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). Go to RFC 5890: https://tools.ietf.org/html/rfc5890

- ***RFC5891: 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. Go to RFC 5891: https://tools.ietf.org/html/rfc5891

- ***RFC5892: 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). Go to RFC 5892: https://tools.ietf.org/html/rfc5892

- ***RFC5893: 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. Go to RFC 5893: https://tools.ietf.org/html/rfc5893

- ***RFC5894: Internationalized Domain Names for Applications (IDNA): Background, Explanation and Rationale***

[11] A. Costello. 2003. IETF Network Working Group https://www.ietf.org/rfc/rfc3492.txt

This informational document provides an overview of a revised system to deal with newer versions of Unicode and provides explanatory material for its components. Go to RFC 5894: 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. Go to RFC 5895: https://tools.ietf.org/html/rfc5895

RFC6530-33 (EAI)

- ***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. Go to RFC 6530: 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. Go to RFC6531: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 [RFC5322] 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. Go to RFC6532: 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. Go to RFC6533: https://tools.ietf.org/html/rfc6533

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.

GB18030 (China)

GB 18030-2000 is a Chinese government standard that specifies an extended code page for use in the Chinese market. 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.

Unicode Technical Standard #46: Unicode IDNA Compatibility Processing

This specification defines a mapping consistent with the normative requirements of the IDNA2008 protocol, and which is as compatible as possible with IDNA2003. For client software, this provides behavior that is the most consistent with user expectations about the handling of domain names with existing data. To learn more go to: http://unicode.org/reports/tr46/

Online resources

- 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
- OS APIs https://developer.apple.com/library/mac/navigation/
- Unicode Security considerations http://www.unicode.org/reports/tr36/
- Unicode security mechanisms http://www.unicode.org/reports/tr39/
- For more details on Unicode character groupings, read the following:
- Unicode planes http://en.wikipedia.org/wiki/Mapping_of_Unicode_character_planes
- Overview of GB18030
- Authoritative mapping table between BG18038-2000 and Unicode: http://source.icu-project.org/repos/icu/data/trunk/charset/data/xml/gb-18030-2000.xml
- GB18030 FAQ - Unicode normalization
- http://unicode.org/reports/tr36/#UTF-8_Exploit
- Unicode.org for security exploits
- .NET Framework 4.5 and higher
- URIs - http://tools.ietf.org/html/rfc3986
- http://www.internic.net/faqs/authoritative-dns.html
- 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

Acknowledgements

**Please provide me your names