

# Email Address Internationalization (EAI) Configuration and Technical Overview

हरीश चौधरी (Harish Chowdhary), UA Ambassador

मेल@ईंटरनेट.भारत, Intern3tgovernance@gmail.com



A collage of diverse hands reaching towards a central white banner with the text "ALL THE DOMAINS AND EMAIL IDs". The hands are of various skin tones and are positioned around the banner, symbolizing global unity and shared information. The background is a solid light pink color.

**ALL THE DOMAINS AND EMAIL IDs**

# Key Fundamental Aspects : Unicode, IDNs, UA

- Encoding glyphs into code points.
- There are multiple ways to use a glyph

“è” = U+00E8

“e`” = “è” = U+0065 U+0300

**Normalization** is a process to insure that no matter the user type, the end representation will be the same.

For the two entries above, Normalization Form C (NFC) will generate

U+00E8 for both

- In specifications, code points are shown in hex using the U+XXXX notation.
- Code points are typically carried using the UTF-8 (Unicode Transformation Format, 8 bit) format.
  - a. Variable number of bytes for a single code point.
  - b. ASCII is used as is.
  - c. Gold standard for carrying Unicode code points in web, protocols, etc

# Domain Names

- A domain name is an ordered set of labels: a.b.c.d
  - A top-level domain is the rightmost label: “d” in left-to-right scripts.
  - The Domain Name System (DNS) is the distributed database and service for querying domain name records.
  - A domain name may have multiple DNS records such as:
    - IPv4 address for that domain name.
    - IPv6 address for that domain name.
    - Hostname of the email server responsible for that domain name and so on
- ....
- A zone is the list of domain name records – called Resource Records (RR) – for the labels under another label (a bit simplified...)

# Types of Domain Names

Domain Name	Category
universal-acceptance-test.international	ASCII.ASCII, new-long
universal-acceptance-test.icu	ASCII.ASCII, new-short
موريتانيا.الشامل-القبول-تجربة	IDN.IDN, RTL
सार्वभौमिक-स्वीकृति-परीक्षण.संगठन	IDN.IDN, Devanagari
ཡངས་ཁྱབ་ངོས་ལེན་བརྟན་དུང་.com	IDN.ASCII, Tibetan
universal-acceptance-test.🇮🇵	ASCII.IDN, RTL

Email Address	Category
email-test@universal-acceptance-test.international	ASCII@ASCII.ASCII, new-long
ਈਮੇਲ-ਪਰਖ@ਸਰਵਵਿਆਪਕ-ਪ੍ਰਵਾਨਗੀ-ਪਰਖ.ਭਾਰਤ	Unicode@IDN.IDN, , Gurmukhi
email-test@universal-acceptance-test.🇮🇵	ASCII@ASCII.IDN, RTL, Hebrew

# Internationalized Domain Names (IDNs)

- Internationalized Domain Names (IDNs) enable the use of non-ASCII characters for any label of a domain name.
- Not all labels of a domain name may be internationalized.  
Example: exâmples.ca
- User uses the IDN version, but the IDN is converted into ASCII for DNS resolution.  
exâmples => exmples-xta => xn--exmples-xta  
The xn-- prefix is added to identify an IDN

## Example process of using an IDN

- User enters in a browser: http://exâmples.ca
- Browser does normalization on the user entry.
- Browser converts exâmples.ca in an ASCII compatible representation called Punycode [RFC3492], and adds 'xn--' in front of it  
exâmples.ca becomes: xn--exmples-xta.ca or नलकसी.भारत becomes:  
xn--11b1b9b0ah0f.xn--h2brj9c
- Browser calls the DNS to get the IP address of xn-- exmples-xta.ca
- Browser connects to the HTTP server at the received IP address

# Internationalized Domain Names

- The protocol for handling IDNs is named IDN for Applications (IDNA)
- Two versions: IDNA2003 and IDNA2008. The latter (IDNA2008) is the one currently used.
- U-label is the Unicode native representation of an IDN label: **निक्सी.भारत**
- A-label is the Punycode representation of an IDN label: **xn--11b1b9b0ah0f.xn--h2brj9c**

# Universal Acceptance (UA)

UA is about how to appropriately support internationalized identifiers, as well as new and long TLDs.

Internationalized identifiers

- \* IDN
- \* EAI



# How to Quantify Universal Acceptance



Accept



Validate



Process



Store



Display

Applicable to both domain names and email addresses

## ACCEPT



- Should accept both A-label and U-label.
- Support UTF-8
- Need large enough input field with total 735 bytes requirements including email local part of up to 64 bytes
- Input to be normalized as per Unicode Normalization Form C (NFC)

## VALIDATE



- Validation against the IDNA 2008 protocol
- TLD verification against the authoritative maintained by IANA.
- Local part of email is defined by service providers, however comply to best practices
- All components of domain name or email(except the TLD name if it is not an IDN) should be in a single script (e.g., Arabic or Han) or closely related scripts (e.g., Japanese Kanji, Katakana, Hiragana, and Romaji).

## STORE



- Use NFC form before storing.
- Use UTF-8 for storage in databases.
- Ensure environment supports Unicode and automatic conversion to or from UTF-8 on input and output.
- Consistent internal representation—either U-labels or A-labels—for IDNs.
- Use most current Unicode standards.
- Regardless of the way addresses and domain names are stored, you must be able to match strings in multiple formats.

## PROCESS




- Use recent versions of s/w libraries and web standards
- Use APIs that support UTF-8
- RTL scripts require special considerations
- Allow as many scripts that the Unicode supports.
- Be aware that some languages can be written using different scripts, and that some scripts can be used to write many different languages.
- Ensure sort order, searches and collation according to locale and languages specifications

## DISPLAY



- Ensure that s/w, applications displays nearly all Unicode code points, with relevant font libraries.
- Display IDNs in their native character form unless there is a specific requirement to display them as A-labels.
- Special consideration while using LTR and RTL scripts..
- It may necessitate to design applications separately for different languages or language groups.

# What Comes Under the UA Ambit?



<b>Applications and Websites</b> <ul style="list-style-type: none"><li>- Wikipedia.org, ICANN.org, Amazon.com, custom websites globally</li><li>- PowerPoint, Google Docs, Safari, Acrobat, custom apps</li></ul>
<b>Social Media and Search Engines</b> <ul style="list-style-type: none"><li>- Chrome, Bing, Safari, Firefox, local (e.g., Chinese) browsers</li><li>- Facebook, Instagram, Twitter, Skype, WeChat, WhatsApp, Viber</li></ul>
<b>Programming Languages and Frameworks</b> <ul style="list-style-type: none"><li>- JavaScript, Java, Swift, C#, PHP, Python</li><li>- Angular, Spring, .NET core, J2EE, WordPress, SAP, Oracle</li></ul>
<b>Platforms, Operating Systems and System Tools</b> <ul style="list-style-type: none"><li>- iOS, Windows, Linux, Android, App Stores</li><li>- Active Directory, OpenLDAP, OpenSSL, Ping, Telnet</li></ul>
<b>Standards and Best Practices</b> <ul style="list-style-type: none"><li>- IETF RFCs, W3C HTML, Unicode CLDR, WHATWG</li><li>- Industry-based standards (health, aviation, ...)</li></ul>

# Anatomy of an IDN and Internationalized Email

Internationalized Domain  
Name (IDN)

<http://www.bücher.berlin>

Protocol

Subdomain

Domain

TLD

Internationalized Email

[Günte@bücher.berlin](mailto:Gün<span>te</span>@b<span>ü</span>cher.berlin)

Local part

Domain name

# Actions to be Performed on a Domain Name

List of actions needed to handle IDNs effectively:

- Normalization ( é U+0065 U+0301 → é U+00E9)
- U-Label~A-Label Conversion ( لاہور to xn--mgbu9cr81d)
- Length validation of A-Label
- Case-folding (if necessary) (Ω × ω ✓, É × é ✓)
- Validation (किताब ✓ क़िताब ×)
- Unicode Compatible Display (ဟယ်လို့ ✓ ❓❓❓❓❓❓❓❓ ×)

# Basics of Unicode and its Forms

- Unicode encodes different scripts in a machine understandable manner i.e., in numerical form called a “Codepoint.”
  - These Codepoints are in hex form and use the U+XXXX notation, the XXX being the assigned code point value.
- Though Unicode aims at adhering to the one character, one code-point principle, there have been instances where it has ended up providing multiple ways of encoding single code point.

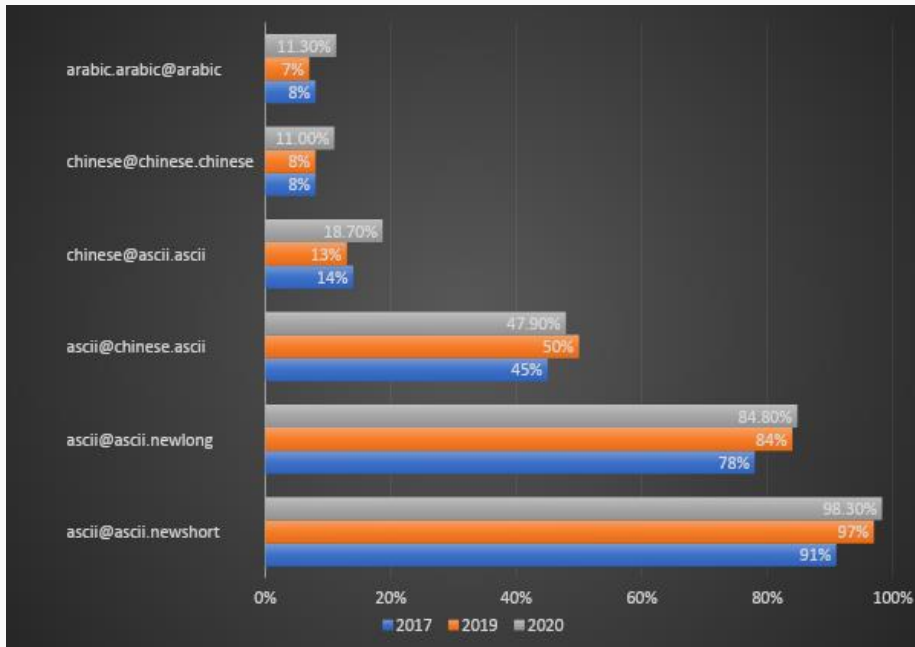
$\begin{aligned} \text{è} &= \text{U+00E8} \\ \text{e} + \text{`} &= \text{è} = \text{U+0065} + \\ &\quad \text{U+0300} \end{aligned}$	$\begin{aligned} \text{क़} &= \text{U+0958} \\ \text{क} + \text{़} &= \text{क़} = \text{U+0915} + \text{U+093C} \end{aligned}$	$\begin{aligned} \text{ĩ} &= \text{U+0622} \\ \text{!} + \text{õ} &= \text{ĩ} = \text{U+0627} + \\ &\quad \text{U+0653} \end{aligned}$
--	--	--

- Normalization ensures that the end representation is the same even if users type differently.
  - IDN standards recommend using [Normalization Form C \(NFC\)](#).

# Email Terminology

UASG Working Groups	Role
Mail User Agent (MUA)	<p>The software used by the user who sends and receives email.</p> <p>With web mail, the MUA is an application run in a browser environment. <i>E.g. Gmail or Xgen Running in Browser/outlook/thunderbird</i></p>
Mail Transfer Agent (MTA)	<p>A server application that receives mail from the MSA, or from another MTA. It will find (through name servers and the DNS) the MX record from the recipient domain's DNS zone in order to know how to transfer the mail. It then transfers the mail (with SMTP) to another MTA (which is known as SMTP relaying) .</p> <p><i>Examples of MTAs are Postfix, Exim, Sendmail, qmail etc.</i></p>
Mail Submission Agent (MSA)	<p>A server program that receives mail from an MUA, checks for any errors, and transfers it (with SMTP) to the MTA hosted on the same server. Typically, this function is bundled with an MTA</p>
Mail Delivery Agent (MDA)	<p>A software, usually on servers, which receives the email from an MTA and is the final destination for the email.</p> <p>It typically stores the email in a file (or a database) and waits for the MUA of the destination user to fetch the email. Typically, this function is bundled with an MTA.</p> <p>An example is <i>Dovecot</i>, which is mainly a POP3 and IMAP server allowing an MUA to retrieve mail, but also includes an MDA which takes mail from an MTA and delivers it to the server's mailbox</p>

# Internationalized Email Acceptance Rate



Testing of popular websites taken from [UASG027](#)



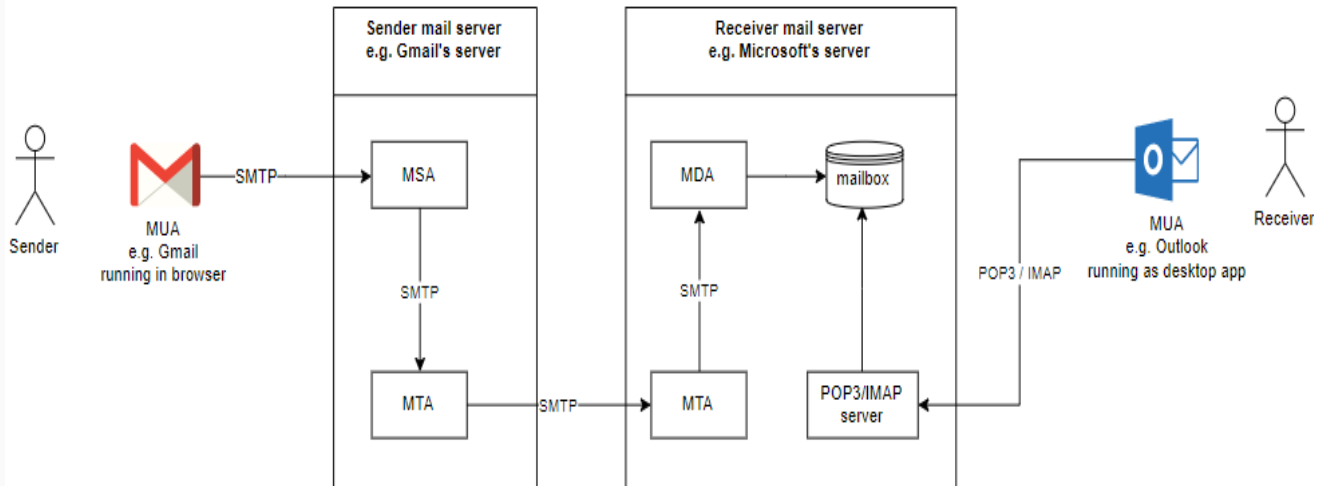
# Email Terminology

For example, Postfix is typically used as an MTA, MDA, and MSA.

## Sending an Email

### Client and server components

There are different components involved in an email transmission from sender to recipient.



# Email : How To Find the Destination Server

When sending email to user@example.com, the method to find the destination email server is by querying the DNS for the MX records of the domain.

For example, the MX records for example.com could be:

- MX 10 server1.example.com
- MX 10 server2.example.com
- MX 20 server3.example.com

The sender email server would then try connecting to either server 1 or server 2 since they have same priority (10)

If none respond, it would then try server 3 since it has a lower priority (20)

The higher number means lower priority

# Email Address Internationalization (EAI)

## **Email syntax: leftside@domainname**

- Domain name can be internationalized as an IDN (U labels or A-labels).
- Left side (also known as local part/mailbox name) with Unicode (UTF-8) is EAI.
- Not all email servers support EAI, so a negotiation protocol is used to only send EAI when the target server supports it. If not, then it falls back and returns an 'unable to deliver' message back to the sender.
- The SMTPUTF8 option is used within the mail transfer protocol (SMTP: Simple Mail Transport Protocol) to signal the support of EAI by an email server

## **Overhead**

- Mail headers need to be updated to support EAI.
- Mail headers are used by mail software to get more information on how to deliver email.

# EAI Protocol Changes

## SMTP

- Is augmented to support EAI.
- Has a signalling flag (SMTPUTF8) to specify support of EAI.
- All SMTP servers in the path must support EAI to successfully deliver the email.

## POP/IMAP

- Are augmented to properly support EAI.
- Have a signalling flag to specify support of EAI.

Could “half support” EAI by providing a downgraded email version to the non-EAI conforming email software clients (downgrading - *While it may look interesting, downgrading may cause many issues for the users and the sysadmin to debug issues. Try to avoid using the downgrading mechanism if you can.* ).

## EAI Protocol Changes : SMTP

- SMTP Server announcing the support of EAI on the initial greeting.

EHLO SMTPUTF8

- SMTP Client connecting to the compliant SMTP Server.

MAIL SMTPUTF8

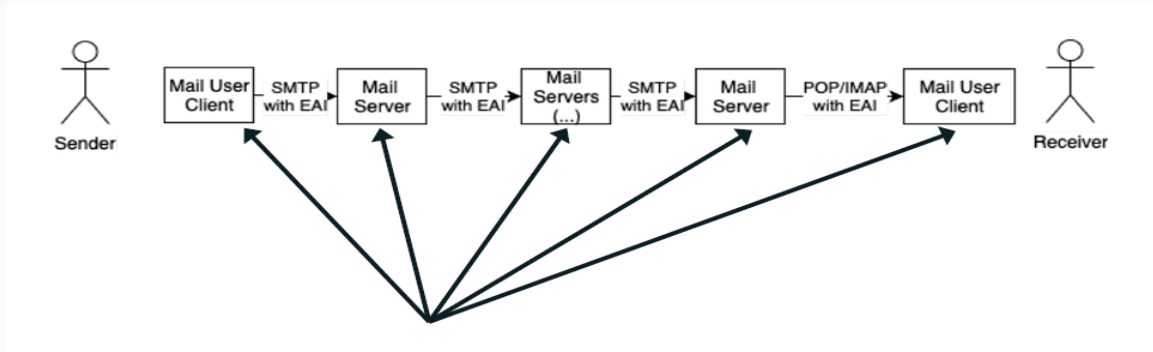
- Headers may have UTF-8 content.
- Email body already supports UTF-8.

## EAI IMAP/POP Protocol Changes

POP : UTF8 command

IMAP : ENABLE UTF8=ACCEPT command

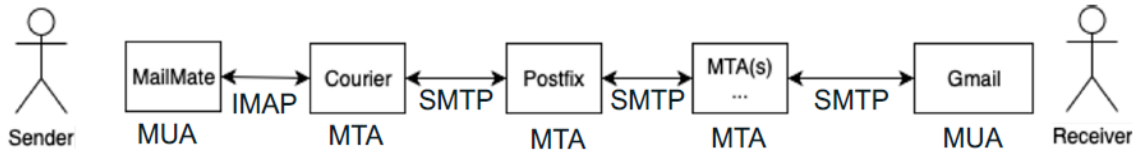
# Protocol Changes, Delivery Path Considerations



To send and receive an email with EAI

- All email parties involved in the delivery path have to be updated for EAI support.
- If a single SMTP server in the path does not support EAI, then the email is not delivered.

# Demonstration : Setup



EAI email path between two (2) users.

## User 1:

[kévin@example.com](mailto:kévin@example.com)

- Using MailMate on MacOSX with SMTPUTF8 enabled.
- Using his own mail server infrastructure.

## User 2:

[peter@example.ca](mailto:peter@example.ca)

- Using Gmail web interface: no configuration necessary apart from the domain and user.
- Using Google mail server infrastructure.

# Demonstration : Setup

For User 1 ([kévin@example.com](mailto:kévin@example.com))

- Its inbound email server is Courier as IMAP server.
- Its outbound email server is Courier as SMTP server.
- Courier uses a Postfix relay server acting as a pure MTA.

For User 2 ([peter@example.ca](mailto:peter@example.ca))

- Its inbound/outbound email servers are Google Gmail servers.

Confirming advertised mail server (Postfix) for [kévin@example.com](mailto:kévin@example.com):

```
dig xn--exmple-xta.com mx
xn--exmple-xta.com. 300 IN MX 10
postfix.xn--exmple-xta.com.
```

Confirming advertised mail server (Gmail) for [peter@example.ca](mailto:peter@example.ca):

```
dig xn--exmple-xta.ca mx
```



# EAI : Configuration

## Postfix: Configuration

These are the specific EAI configuration **requirements**.

```
/etc/postfix/main.cf
```

```
...
```

```
# enable SMTPUTF8
```

```
smtputf8_enable = yes
```

```
#defines the hostname for SMTP. Does not need to be an IDN.
```

```
myhostname = postfix.xn--exmple-xta.com
```

```
#defines the domain of the host. Does not need to be an IDN.
```

```
mydomain = xn--exmple-xta.com
```

```
#domains of the user mailboxes. With smtputf8_enable, this is the key config.
```

```
virtual_mailbox_domains = exâmples.com
```



# EAI : Configuration

## Courier: Configuration

These are the specific EAI configuration **requirements**.

```
/etc/courier/defaultdomain
```

```
# defines the default domain used by Courier  
exâmples.com
```

```
/etc/courier/locals
```

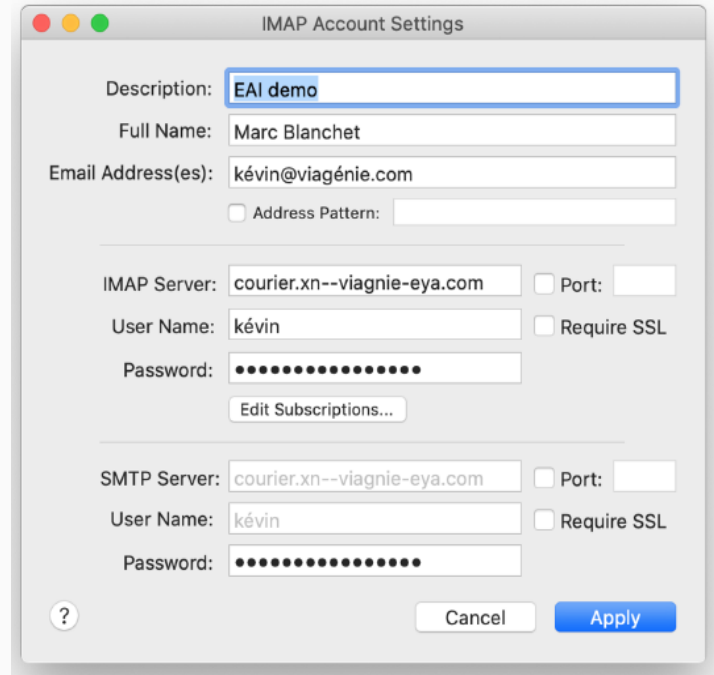
```
# defines the local domain Courier will deliver mail to  
exâmples.com  
localhost
```

# EAI : Configuration

## MailMate: Configuration

Commercial email client on MacOSX.

- ❑ For user [kévin@example.com](mailto:kévin@example.com)
- ❑ Using `courier.xn--exmple-xta.com` running Courier software as its IMAP and SMTP server.
- ❑ Set support for SMTPUTF8:
  - Defaults `write com.freron.MailMate MmSMTPUTF8Enabled -bool YES`



# From MailMate to Gmail

- ◉ Email sent from `kévin@exemple.com` to `peter@exemple.ca`

From: Kévin <kévin@exemple.com>  
Subject: test EAI  
Date: October 31, 2019 at 15:40  
To: peter@exemple.ca

---

Ceci est un test EAI.  
Kévin



test EAI Boîte de réception x

 Kévin via exemple.com  
À peter  
Ceci est un test EAI.  
Kévin

← Répondre ↶ Répondre à tous ➤ Transférer

Message sent from MailMate.

Message received on Gmail.

# Important points

## Case Folding

Mail sent from peter@viagenie.ca to KÉVIN@exâmples.com

- Email was NOT delivered.

Why?

kevin and KEVIN as local parts are typically handled as same user. Case folding is automatic for ASCII local parts.

But k evin and K EVIN are by default not the same user. **Unicode case folding is typically not done by servers.**

Typically in ASCII email addresses, the local part can be uppercase or lowercase. In UTF-8, case folding is not the same as ASCII.

# Important points

## Avoiding Interpretation as Spam

- Spam filtering may think EAI is spam even with SPF, DKIM, etc.
  - This is because the spam filtering does not know about EAI.
- New TLDs and IDNs may also be viewed as spam.
- Spam filtering may think EAI is spam even with SPF, DKIM, etc. This is because the spam filtering does not know about EAI.
- New TLDs and IDNs may also be viewed as spam. Some Mail User Agent (MUA) software uses your contacts database to determine if an email is known to you or not
- This will make it more likely to be considered spam if the email of the sender is not in your contacts database.
- Make sure the contacts database contains the EAI email of the contacts.

# Enabling UA the Python Way

*Coding for UA compliant domain  
and email validators*

A decorative pattern at the bottom of the slide consisting of multiple rows of triangles pointing up and down, creating a zigzag effect. The triangles are in various shades of gray, from light to dark, and are arranged in a repeating sequence.

# About Python Programming Language

- Python is an open-source language operating under OSI-approved open-source license.
- A general purpose, interpreted, high level programming language with a very vibrant developer support community.
- Most of the code/packages are open- source, at least all those concerning UA.

# Python 2: Datatypes and Unicode Enablement

- The prevalent Python version is **Python 3**, however, there are many older implementations which still use **Python 2** due to legacy reasons.
- As per the survey\*, Python 2 is still employed by 6% of Python implementations.
- In **Python 2**, the default encoding for the String Type “str” was ASCII.
  - The Unicode encoding required to be explicitly declared by prepending “u” before the string literal.
  - It used to show datatype of such a string as “unicode”, instead of the default “str”
- Python had a major Unicode support enablement in **Python 3**
  - \*Python Developers [Survey](#) 2020 conducted by JetBrains





# Python 3: Datatypes and Unicode Enablement

- Python 3 by default uses **UTF-8** for its entire source code.
- The primary string datatype “str” thus is by default Unicode enabled.
- In Python 3 onwards, one can even use Unicode in the string literal, meaning, one can have Unicode variable names too.
- For more information, refer to the Python official documentation on Unicode support [here](#).
- In short, the basic “str” datatype in Python is sufficient to cater to Unicode handling requirements.



# Unicode Normalization

- Though Unicode aims at adhering to the one character, one code-point principle, there have been instances where it has ended up providing multiple ways of encoding single code point.

$\text{è} = \text{U+00E8}$ $\text{e} + \text{`} = \text{è} = \text{U+0065} + \text{U+0300}$	$\text{क} + \text{◌} = \text{क़} = \text{U+0915} + \text{U+093C}$ $\text{क़} = \text{U+0958}$	$\text{ĩ} = \text{U+0622}$ $\text{ĩ} = \text{U+0627} + \text{U+0653}$
--	--	--

- Normalization ensures that the end representation is the same, even if users type differently.
  - IDN standards recommend using [Normalization Form C \(NFC\)](#).



# IDNA Conversion

- The default python library “encodings.idna” for IDN Compatibility Processing, is based of IDNA2003 implementation. It is strongly recommended not to use the library.
  - Ref: <https://docs.python.org/2.4/lib/module-encodings.idna.html>
- Alternatively, the “idna” package present on Python Package Index (PyPI) ably supports the IDNA2008 implementation.
- Can be installed in two ways:
  - pip install idna
  - Downloading files from <https://github.com/kjd/idna> followed by:
    - "python setup.py install"
- The “idna” library is [highly recommended](#) by the UASG for UA compliant code development by developers.

# IDNA Conversion - IDNA Package

- There are two major functions:
  - Encode: converts the IDN to its Punycode equivalent
  - Decode: converts the Punycode domain to its IDN equivalent
- There are three modes of operation:
  - `idna.encode(<IDN_STRING>)` & `idna.decode(<PUNYCODE_STRING>)`
    - E.g., `idna.encode(exâmp.le.ca')` or `idna.decode('xn--exmple-xta.ca')`
  - `<IDN_STRING>.encode('idna')` & `<PUNYCODE_STRING>.decode('idna')`
    - *Here variables holding <IDN\_STRING> and <PUNYCODE\_STRING> will be used*
  - `idna.alabel('UNICODE_Label')` & `idna.ulabel('PUNYCODE_LABEL')`
    - `idna.alabel('périple')` & `idna.ulabel('xn--priple-bva')`
- For bug-report/issue-tracking, please visit:

<https://github.com/kjd/idna/issues>



# Storing as U-Label or A-Label

www.例如.中国

U-Label

www.xn--fsqu6v.xn--fiqs8s

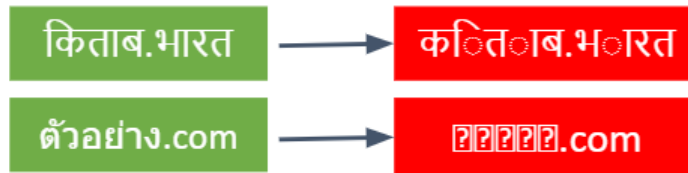
A-Label

- As long as the labels are normalized, the U-Label and A-Label are fully interconvertible and can be stored on the database in either form.
- However, if the business requirement dictates some operations which depend on various forms of Unicode processing (Validating, sorting, searching), it becomes incumbent on the developer to store in U-Label format.

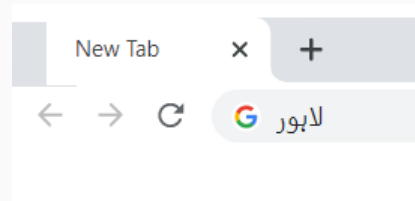
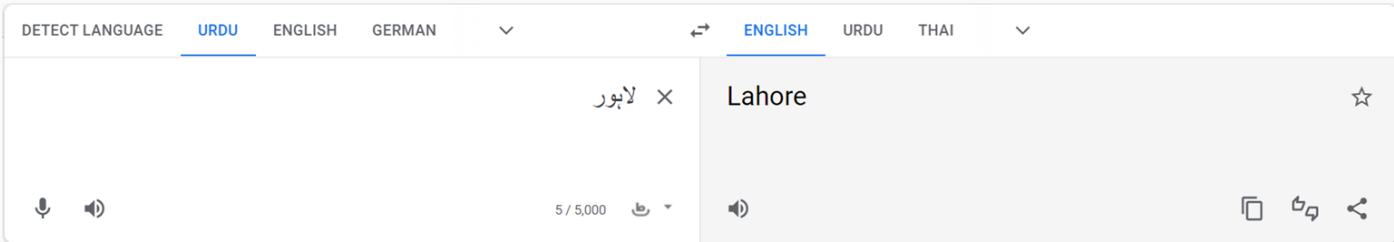


# Displaying the Unicode Properly

- Make sure that your applications/interfaces support the necessary text-layout engines viz. Uniscribe / DirectWrite (Windows), Pango and ICU (Linux), AAT (IOS). Most of the platforms/languages already support them implicitly. This is more relevant for legacy applications.
- Always ensure U-Label for user-facing interfaces unless A-Label is explicitly needed.
- Make sure you use appropriate fonts for ensuring proper rendering and legibility for the end user.



# Displaying Scripts from Right-to-Left (RTL)



# Validators

## *Domain and Email Validation*





# Domain Name Validation

- Domain name validation typically involves checking if the user-submitted domain name complies with the URL protocol (**protocol compliance validation check**).
- The user-submitted domain resolves to a registered domain name (**functional validation check**).



# Domain Validation



- Validators Package
  - Python Data Validation for Humans™
- Domain validation is RegEx-based
  - With very rudimentary support for Punycode inclusion.
- Email Validation is also RegEx-based
  - Seems to give results but other important tasks like normalization and cannot be expected.
- Better to be avoided until the required features are well supported.

# Domain Validation



- “idna” package available through PyPI
- Provides the “validation” through exception mechanism (typically not the expected way though).

```
>>> import idna
>>> idna.encode('Königsgäßchen')
```

- Raises exception:
  - `idna.core.InvalidCodepoint: Codepoint U+004B at position 1 of 'Königsgäßchen' not allowed`
- Likewise, the library raises various exceptions as per the specifications from the IDNA2008 specification.

# Domain Validation



- “idna” package available through PyPI
- More specific exceptions that are generated are:
  - `idna.IDNABidiError` - when the error reflects an illegal combination of left-to-right and right-to-left characters in a label.
  - `idna.InvalidCodepoint` - when a specific codepoint is an illegal character in an IDN label (i.e.. INVALID).
  - `idna.InvalidCodepointContext` - when the codepoint is illegal based on its positional context (i.e., it is CONTEXTO or CONTEXTJ but the contextual requirements are not satisfied).

# Domain Validation



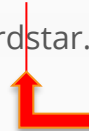
- `idna.IDNABidiError` - when the error reflects an illegal combination of left-to-right and right-to-left characters in a label
  - `>> import idna`
  - `>> idna.encode('कखب.com')`
- Raises exception:
  - `IDNABidiError('Invalid direction for codepoint at position {0} in a left-to-right label'.format(idx))`
  - `idna.core.IDNABidiError: Invalid direction for codepoint at position 3 in a left-to-right label`

# Domain Validation



- `idna.InvalidCodepointContext` - when the codepoint is illegal based on its positional context (i.e., it is `CONTEXT0` or `CONTEXTJ` but the contextual requirements are not satisfied).

- `>> import idna`
- `>> idna.encode('wordstar.com')`



- *CONTAINS ZERO WIDTH JOINER, A CONTEXTJ Character*
- *word **ZWJ** star.com*

- Raises exception:
  - `raise IDNAError('Unknown codepoint adjacent to joiner {0} at position {1} in {2}'.format(`
  - `idna.core.IDNAError: Unknown codepoint adjacent to joiner U+200D at position 5 in 'word\u200dstar'`

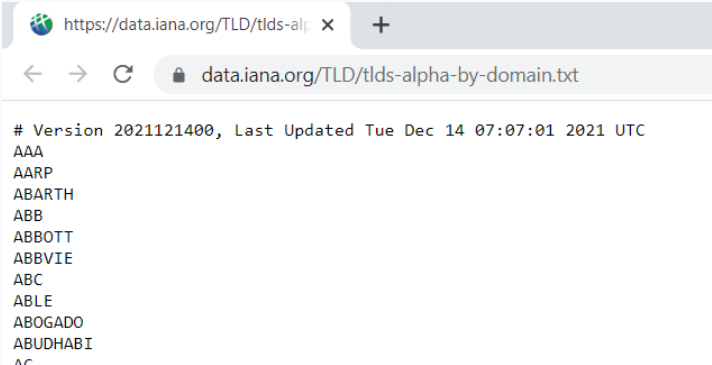
# TLD Validation

Validating the TLD part with IANA repository

`https://थिंकट्रान्स.भारत`

`https://<DOMAIN_NAME>.<TOP_LEVEL_DOMAIN>`

- Validate the TLD part of the domain name directly with the IANA repository.
- TLD allocation process involves many other processes than mere IDNA protocol compliance.
- Checking directly with the IANA repository will always yield the most accurate results on the date.



```
# Version 2021121400, Last Updated Tue Dec 14 07:07:01 2021 UTC
AAA
AARP
ABARTH
ABB
ABBOTT
ABVIE
ABC
ABLE
ABOGADO
ABUDHABI
AC
```

# TLD Validation

Validating the TLD part with IANA repository

अक्षत@थिंकट्रान्स.भारत

<LOCAL\_PART>@<DOMAIN\_NAME>.<TOP\_LEVEL\_DOMAIN>

## SAMPLE CODE FOR VALIDATING ASCII/A-LABEL TLDs WITH LIVE IANA REPOSITORY:

```
import urllib3
http = urllib3.PoolManager()
strTLD = "XN--H2BRJ9C"
resp = http.request("GET", "https://data.iana.org/TLD/tlds-alpha-by-domain.txt")
strValidTLDs = resp.data.decode().split('\n')
if(strTLD.upper() in strValidTLDs):
    print ("TLD FOUND!")
else:
    print ("TLD NOT FOUND!")
```

*For IDN TLDs, always convert them to their Punycode equivalent, uppercase, before checking*

*Take appropriate action here as per your code logic*

SAMPLE INPUT	OUTPUT
AMAZON	TLD FOUND!
XN--H2BRJ9C	TLD FOUND!
ABCDE	TLD NOT FOUND!

**CAUTION:** CONVERT U-LABEL TO A-LABEL BEFORE CALLING THIS!



# Python 3: Socket, Hostname Resolution

```
socket.gethostbyname(hostname)
```

**Does not support IPV6  
Addresses**

```
socket.getaddrinfo(host, port, family=0, type=0, proto=0, flags=0)
```

**Supports IPV4/IPV6 dual stack**

- Both above methods take A-label as well as the U-label as the host name.
- Raises “exception socket.gaierror” if provided Punycode instead of the U-label.

# Python 3: Socket, IP Resolution

`socket.getfqdn([name])`

Returns a fully qualified domain name for *name*. If *name* is omitted or empty, it is interpreted as the local host.

`socket.gethostbyaddr(ip_address)`

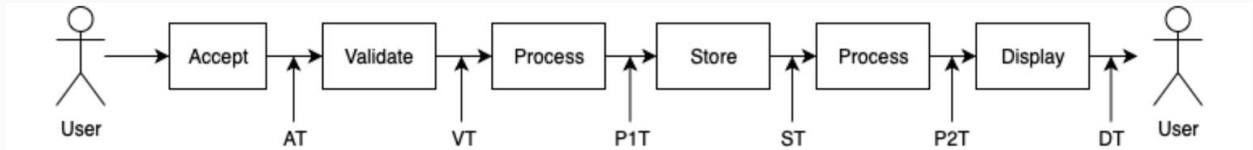
Returns a triple (hostname, aliaslist, ipaddrlist). Supports both IPV4 and IPV6 Addresses

- Both above methods take the U-label as the host name.
- Raises “exception `socket.gaierror`” if provided Punycode instead of the U-label.

# Major UA Challenges

- Some applications are still verifying domain names incorrectly by using one of the outdated methods:
  - Check for a fixed length of TLD between 2-4 characters (TLD can be up to 63 characters).
  - Check from a fixed set of TLDs, e.g., using static list of strings.
  - Check for only ASCII characters.
- Some applications do not cater to additional requirements for validating EAI:
  - Check mailbox name to be a valid string in UTF-8 format.
  - DomainName can be ASCII or IDN.

# UA-Readiness Testing Framework



AT: Accept test

ST: Store test

VT: Validate test

P2T: Process test on the output

P1T: Process test on the input

DT: Display test

Detailed UA-Readiness Testing Framework is available in the form of a Technical Guide UASG026 [here](#).

# Validating User Input

- Validating user input, or any input, is extremely useful for various reasons, some of which include: a better user experience, increased security, and avoiding irrelevant issues.
- Validating domain names and email addresses is useful.
- Some validation methods for domain names and email addresses:
  - Basic syntax checks: is the syntax of the string correct?
    - Does the domain name contain '!'?
    - Does the email address contain '@' and a valid domain name part?
  - Functional checks: does the domain name or email address work?
    - Is the top-level domain (TLD) in use?
    - Is the whole domain name in use?
    - Is the email in use?

# Validating Domain Names

## Validating syntax:

- ASCII: RFC1035
  - Composed of letters, digits, and hyphen.
  - Max length is 255 octets with each label up to 63 octets.
- IDN: IDNA2008 (RFCs 5890-5894)
  - Valid A-labels
  - Valid U-labels

## Validating function:

- Is the TLD in use?
  - Verify against the list of TLDs.
  - Verify using a DNS request.
- Is the whole domain name in use?
  - Verify using a DNS request.

# Validating Email Addresses

## Validating syntax:

- An email address is composed of:  
mailboxName@domainName
- Validating syntax:
  - For domainName, see earlier discussion.
  - For mailboxName:
    - ASCII
    - UTF8 (for EAI)

## Validating function:

- Is the domain name set up to send and receive emails?
- Is the mailbox name able to send and receive emails?

# Email Validation





# Composition of an Email

<LOCAL PART> @ <DOMAIN NAME>



- Has to be validated as per the general guidelines laid down in the RFCs.
  - A good guide to the same is released by the UASG titled *"UASG 028 Considerations for Naming Internationalized Email Mailboxes EN"*. It can be accessed [here](#).
- Has to be validated as per the general "Domain Name Validation" process, as discussed.

# Email Validation

Validating an Email is much more than just validating for certain characters.

<LOCAL\_PART>@<DOMAIN\_NAME>.<TOP\_LEVEL\_DOMAIN>

Each of the above get governed by different sets of Internet standards/processes.

**LOCAL\_PART:** Gets governed by the policies of the Email Service Providing Entity, typically the domain owning body. Please refer to best practices recommended by the UASG [here](#).

**DOMAIN\_NAME:** Gets governed by IETF Protocols (IDNA 2008) for domain names.

**TOP\_LEVEL\_DOMAIN:** [IANA TLD Repository](#)

# Email Validation



- A python developer should avoid the temptation of using an “re” package and begin the first attempts at validating an email address with naïve RegEx’s like below:

```
'^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[\.]\w{2,3}$'
```

- For obvious reasons, in the context of IDNs, validating for only “a-z” is a non-starter.
- So is validating domain name and top-level domain (TLD) with a “\w”.

# Email Validation

- Since individual parts of the email ID is governed by different entities with different policies, it is best to delegate the validation function of an email ID to a proper dedicated library.
- The Python community is rich with options when it comes to using open-source libraries which “just work”.
- Here are some of the options at hand:
  - email-validator 1.0.5
  - pylsEmail 1.3.2

# Email Validation



**email-validator:** (Current version 1.1.3)

- Basic syntax checking
- Checks resolvability
- Supports IDN email by including SMTPUTF8 in resolvability check and Internationalized Local part check
- For storing email addresses, provides “Normalization” option
- Complies with IDNA2008

**Installation:** *pip install email-validator*

# Email Validation



## **email-validator:** (Current version 1.1.3)

- Checks if the domain part of the email resolves to a valid domain name.

```
>>from email_validator import validate_email, EmailNotValidError
>>valid = validate_email("test@mydomain.tld")
>>print(valid.email)
```

The domain name mydomain.tld does not exist.

```
>>valid = validate_email("proprietaire@rhône.hôtel")
>>print(valid.email)
```

proprietaire@rhône.hôtel

# Email Validation



**email-validator:** (Current version 1.1.3)

- Provides normalized form of the email part
- This example uses a non-NFC character
- `a + ^ (U+0061 + U+0302)` (Normalizes to `â - U+00E2` )

```
>>from email_validator import validate_email, EmailNotValidError
>>valid = validate_email("utilizator@exâmples.ca")
>>print(valid.email)
>>print(email)
"utilizator@exâmples.ca"
```

**Normalized  
String**

**Non-Normalized  
String**

# Email Validation



**pyIsEmail** : (Current version 1.3.2)

- This package is supposed to be doing purely protocol checking for an email.
- However, it does not seem to support EAI at the moment.

```
>>from pyisemail import is_email
>>address = "utilizator@example.ca"
>>bool_result = is_email(address)
>>detailed_result = is_email(address, diagnose=True)
>>print(bool_result)
>>print(detailed_result)
False
<InvalidDiagnosis: EXPECTING_ATEXT>
```



# Test Cases for IDNs and Internationalized Email Addresses

Available in: "UASG 004 Use Cases for UA Readiness Evaluation EN"

Link: <https://uasg.tech/download/uasg-004-use-cases-for-ua-readiness-evaluation-en/>

	Category	Domain Name	Script
1	ASCII.ASCII (new-long)	<a href="#">universal-acceptance-test.international</a>	Long ASCII
2	ASCII.ASCII (new-short)	<a href="#">universal-acceptance-test.icu</a>	Short ASCII
3	IDN.IDN (RTL)	موريتانيا	
4	IDN.IDN	համընդհանուր-ընկալում-թեստ.հայ	
5	IDN.IDN	সর্বজনীন-স্বীকৃতি-পরীক্ষা.ভারত	
37	Unicode@IDN.IDN	почта-тест@универсальное-принятие-тест.москва	Cyrillic
38	Unicode@IDN.IDN	ईमेल-परीक्षण@सार्वभौमिक-स्वीकृति-परीक्षण.संगठन	Devanagari
39	Unicode@IDN.IDN	ფლტის-ტესტი@უნივერსალური-თავსობადღის-ტესტი.ge	Georgian
40	Unicode@IDN.IDN	ηλεκτρονικό-μηνυμα-δοκιμή@καθολική-αποδοχή-δοκιμή.eu	Greek
41	Unicode@IDN.IDN	ઇમેઇલ-પરીક્ષણ@સાર્વત્રિક-સ્વીકૃતિ-પરીક્ષણ.ભારત	Gujarati

# Mail Software and Services

## EAI Support

- MailMate (MUA) on MacOSX: v.1.9.4 minimum
- Postfix (SMTP) v3.0 minimum
  - [http://www.postfix.org/SMTPUTF8\\_README.html](http://www.postfix.org/SMTPUTF8_README.html)
- Courier (IMAP, POP, SMTP) v1.0 minimum (IMAP version 5.0.8)

## EAI NOT supported (as of October 2019):

- Dovecot (IMAP, POP)
- Zimbra
- Mozilla Thunderbird

## Services supporting EAI

- Gmail
- Outlook

More info in UASG021A available at [www.uasg.tech](http://www.uasg.tech)

# Conclusion

- ❑ EAI is essentially supporting UTF-8 local parts of an email address.
  - Which also means supporting in the headers.
  - Requires changes to mail servers and mail clients.
- ❑ All SMTP servers in the path must be EAI-enabled to deliver the mail to the destination.
- ❑ Courier and Postfix are two open-source software that support EAI with very limited changes in configuration.
- ❑ In configuration files, be careful to use A-labels, U-labels, or both, depending on the situation.

# Additional Information for EAI, IDNs, UA

## ❑ Universal Acceptance Steering Group (UASG)

<https://uasg.tech>

<https://www.icann.org/ua>

## ❑ Internationalized Domain Names (IDNs)

<http://icann.org/idn>

## ❑ UASG028 : Considerations for Naming Internationalized Email Mailboxes

- The document is intended for email systems administrators to help them provision mailboxes, configure, and manage systems compatible with internationalized email addresses.
- It outlines the considerations for naming internationalized mailboxes and helps administrators make good choices when setting their mailbox names policy.



Universal Acceptance

# By the Numbers

**1,180**

TLD zones

**210,811,274**

second level domains

**34,996,159**

unique mail servers

**2,537,159**

unique IP addresses

**Of the tested IP addresses:**

**32.33%**

did not respond

**60.63%**

did not support  
internationalized  
email addresses

**7.04%**

were set up to accept  
an internationalized  
email address

спасибо 谢谢  
GRACIAS

**THANK YOU**

ありがとうございました MERCI

DANKE धन्यवाद

شُكْرًا **OBRIGADO**