Programming to Support Universal Acceptance of Domain Names and Email Addresses

.bh Hackathon workshop

Delivered by: Dr. Fatema Akbar

23 April 2025

Agenda

- Introduction
- Overview of Universal Acceptance
- Fundamentals of Unicode
- Fundamentals for IDNs and EAI
- Programming for UA
  - Processing Domain Names
  - Processing Email Address
- Conclusion

Overview of Universal Acceptance

Kahoot!

Categories of Domain Names and Email Addresses

- It’s now possible to have domain names and email addresses in local languages using UTF8.
  - Internationalized Domain Names (IDNs)
  - Email Address Internationalization (EAI)

- Domain names

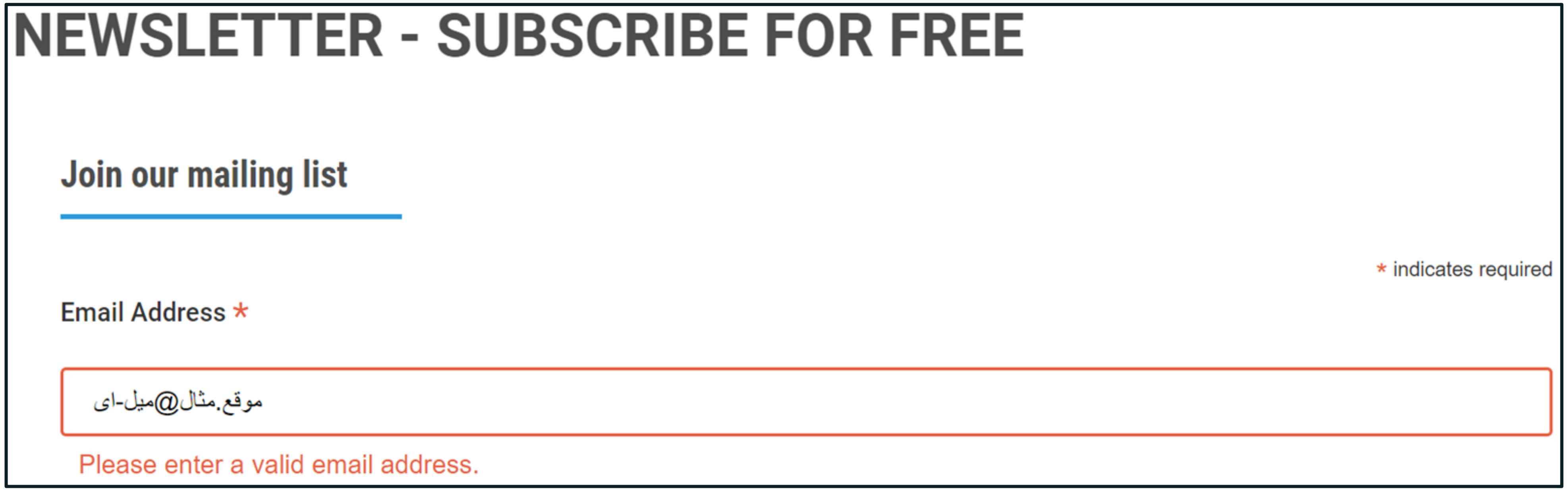
Newer top-level domain names:	example.sky
Longer top-level domain names:	example.abudhabi
Internationalized Domain Names:	普遍接受-测试.世界

- Internationalized email addresses (EAI)

ASCII@IDN	marc@société.org
UTF8@ASCII	ईमेल@example.com
UTF8@IDN	测试@普遍接受-测试.世界
UTF8@IDN; right-to-left scripts	موقع.مثال @ميل-اى

Universal Acceptance and Digital Inclusion

- Many people around the world are currently excluded from experiencing the full benefits of the Internet simply because they're unable to use a domain name or email address of choice in their language and script
- For example, a valid email is rejected by a form in a website (and also incorrectly displayed from left to right instead of RTL):



NEWSLETTER - SUBSCRIBE FOR FREE

Join our mailing list

* indicates required

Email Address *

موقع.مثال@ميل-اى

Please enter a valid email address.

Universal Acceptance of Domain Names and Email

Goal

All domain names and email addresses work in all software applications.

أن تعمل جميع أسماء النطاقات وعناوين البريد الإلكتروني في جميع التطبيقات البرمجية.

Impact

Promote consumer choice, improve competition, and provide broader access to end users.

What are the advantages/importance of being UA ready?

For students	Developers	Governments	Businesses

Scope of UA Readiness for Programmers

- Support all Domain Names including Internationalized Domain Names (IDNs) and all Email Addresses, including Internationalized Email Addresses (EAI):

Accept | Validate | Process | Store | Display

- Accept: The user can input characters from their local script into a text field.
- Validate: The software accepts the characters and recognizes them as valid.
- Process: The system performs operations with the characters.
- Store: The database can store the text without breaking or corrupting.
- Display: When fetched from the database, the information is correctly shown.

Question

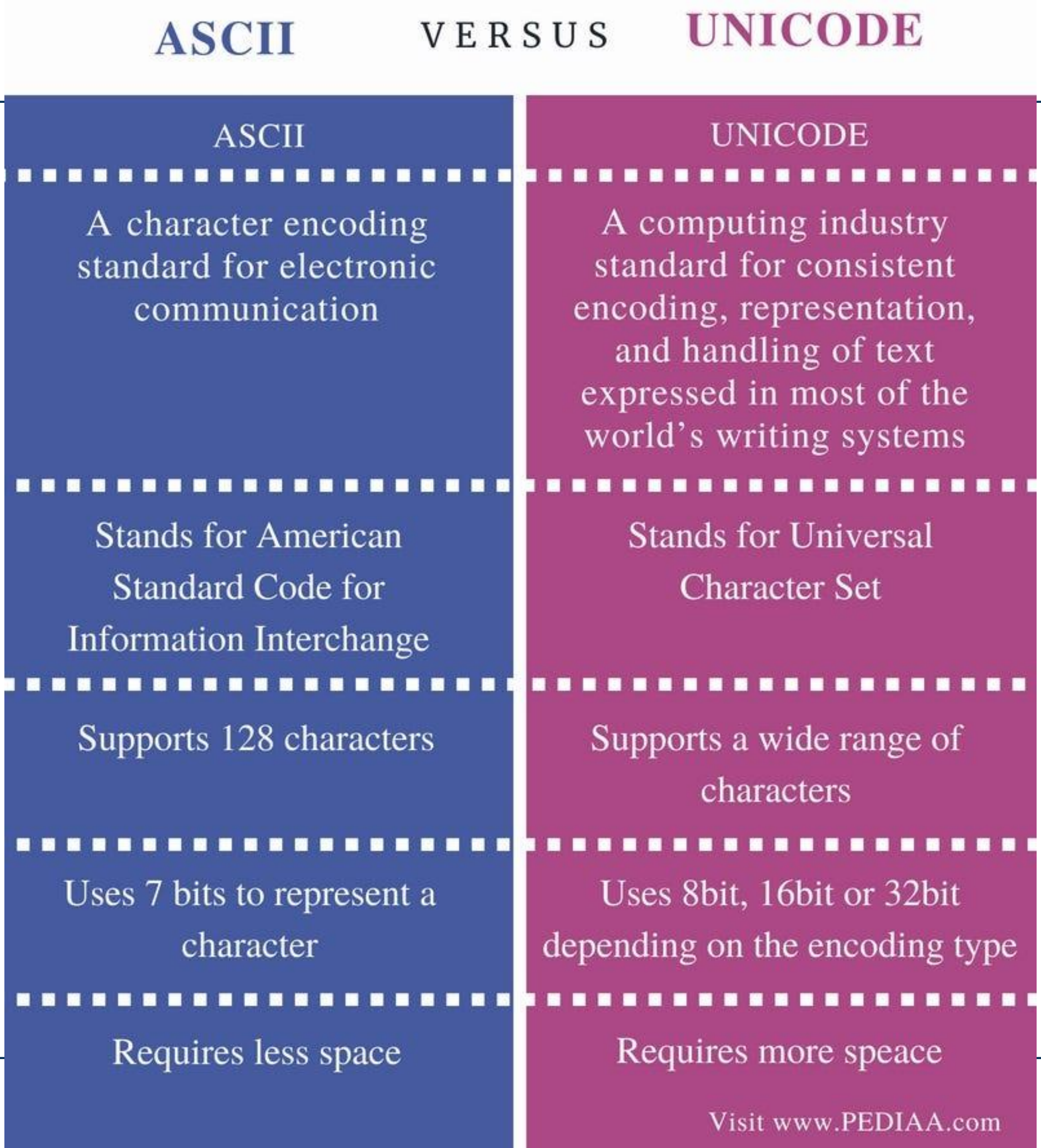
- To enhance systems to be Universal Acceptance (UA) ready, which of the following categories of domain names and email addresses are relevant?
  1. ASCII domain names.
  2. Internationalized Domain Names (IDNs).
  3. Internationalized email addresses (EAI).
  4. All the above.
  5. Only 2 and 3.

Question

- To enhance systems to be Universal Acceptance (UA) ready, which of the following categories of domain names and email addresses are relevant?
  1. ASCII domain names.
  2. Internationalized Domain Names (IDNs).
  3. Internationalized email addresses (EAI).
  4. All the above.
  5. Only 2 and 3.

Fundamentals of Unicode

What do you know about ASCII and Unicode?



ASCII VERSUS UNICODE

ASCII	UNICODE
A character encoding standard for electronic communication	A computing industry standard for consistent encoding, representation, and handling of text expressed in most of the world's writing systems
Stands for American Standard Code for Information Interchange	Stands for Universal Character Set
Supports 128 characters	Supports a wide range of characters
Uses 7 bits to represent a character	Uses 8bit, 16bit or 32bit depending on the encoding type
Requires less space	Requires more speace

Visit www.PEDIAA.com

Character and Character Set

- A label or string such as أهلا, नमस्ते, Hello is formed of characters.
  - Hello --> H e l l o
- A character is unit of information used for the organization, control, or representation of textual data.
- Examples of character:
  - Letters
  - Digits
  - Special characters i.e. Mathematical symbols , punctuation marks
  - Control Characters - typically not visible
- American Standard Code for Information Interchange (ASCII) encodes characters used in computing including letters a-z, digits 0-9 and others.

Character Encoding

Character encoding is a system used to convert characters (like letters, numbers, and symbols) into a format that computers can understand and process.

Essentially, it assigns a unique number (or code) to each character.

Encoding Schemes: schemes define how characters are encoded.

Examples

- ASCII: Uses 7 bits to represent characters, suitable for English text.
- Unicode: Uses up to 32 bits, supporting characters from almost all languages and symbols.

Hello World: Python

```
print("Enter your input: ")

inputstr = input() #default character encoding is UTF-8

print("Input data is: ")

print(inputstr)
```

Hello World: Java

```
import java.util.Scanner;

public class ReadWriteUnicode {

      public static void main(String[]args)   {

       Scanner scr = new Scanner(System.in);

       System.out.println("Enter your input");

       String Input = scr.nextLine(); //default character encoding is UTF-8

       System.out.println("Receieved input is: "+Input);

   }

}
```

Unicode Encoding – File Reading/Writing in Python

- Read UTF-8 file

```
file = open("filepath",'r',encoding='UTF-8')
for line in file:
    print(line)
file.close()
```

- Write UTF-8 file

```
file2 = open(“filepath",'w',encoding='UTF-8')
data_to_write='السلام عليكم'
file2.writelines(data_to_write)
file2.close()
```

Unicode Encoding – File Reading in Java

```
 public void ReadFile(String filename){

try {
   FileInputStream fis = new FileInputStream(filename);
   InputStreamReader isr = new InputStreamReader(fis, StandardCharsets.UTF_8)
   BufferedReader br = new BufferedReader(isr);
   String line ="";
   while((line = br.readLine())!=null) {
   System.out.println(line);
   }
   fis.close();

}catch(IOException ex) {

       System.err.println(ex.toString());

}

}
```

Unicode Encoding – File Writing in Java

```
public void WriteFile(String filename, String text){
      try{
           FileOutputStream fis = new FileOutputStream(filename);
           OutputStreamWriter osw = new OutputStreamWriter(fis,StandardCharsets.UTF_8);
           BufferedWriter bw = new BufferedWriter(osw);
           bw.write(text);
           bw.flush();
           fis.close();
      }catch(IOException ex) {
        System.err.println(ex.toString());       }
  }
```

Normalization

- There are multiple ways to encode certain glyphs in Unicode:
  - è = U+00E8
  - e + ` = è = U+0065 + U+0300
  - آ= U+0622
  - ◌ٓ + ا= آ=U+0627 U+0653
- The following string can exist in corpus in the form of first string below, whereas input string is in the form of second string, below. So, search result will be empty.
  - آدم(U+0622 U+062F U+0645)
  - آدم(U+0627 U+0653 U+062F U+0645)
- For searching , sorting and any string operations we need normalization.
- Normalization ensures that the end representation is the same, even if users type differently

Normalization

- Different normalization forms defined by Unicode are listed below:
  - Normalization Form D (NFD)
  - Normalization Form C (NFC)
  - Normalization Form KD (NFKD)
  - Normalization Form KC (NFKC)
- In domain names NFC is used.

Normalization Code - Python

```python
import unicodedata

input_str = input() # take input from user

normalized_input = unicodedata.normalize('NFC',input_str) #normalize user input

print(normalized_input)
```

Normalization Code - Java

```
import com.ibm.icu.text.Normalizer2

Scanner sc = new Scanner(System.in);

Normalizer2 norm = Normalizer2.getNFCInstance(); //get NFC object

String input;

input = sc.nextline();

String normalized_input = norm.normalize(input);
```

Internationalized Domain Names

Two equivalent forms of IDN domain labels

www.exâmple.ca

U-Label	A-Label
Human users (using UTF-8 format E.g. exâmple	Applications or systems internally ASCII e.g. xn----f38am99bqvcd5liy1cxsg

Converting U-label to A-label

Take user input and normalize and check against IDNA2008 to form IDN U-label.

- exâmple
- 普遍接受-测试

Convert U-label to punycode (using RFC3492).

- exmple-xta
- --f38am99bqvcd5liy1cxsg

Add the "xn--" prefix to identify the ASCII string as an IDN A-label.

- xn--exmple-xta
- xn----f38am99bqvcd5liy1cxsg

◉ Use the latest IDN standard called IDNA2008 for IDNs.

- ○ Do not use libraries for the outdated IDNA2003 version.

Convert U-Label => A-Label: Python

```
import unicodedata #library for normalization
import idna        #library for conversion
domainName = 'مصر.صحة'
try:
    domainName_normalized = unicodedata.normalize('NFC', domainName) #normalize to NFC
    print(domainName_normalized)
    domainName_alabel = idna.encode(domainName_normalized).decode("ascii") #U-label to A-label
    print(domainName_alabel)
    domainName_ulabel = idna.decode(domainName_alabel)
    print(domainName_ulabel)
except idna.IDNAError as e:
        print("Domain '{domainName}' is invalid: {e}")  #invalid domain as per IDNA 2008
except Exception as e:
      print("ERROR: {e}")
```

Convert U-Label => A-Label: Java

- International Components for Unicode (ICU).
- The gold standard library for Unicode. It was developed by IBM and is now managed by Unicode. In sync with Unicode standards.
  - IDNA Conversion is based on Unicode UTS46, which supports transition from IDNA2003 to IDNA2008. However, it is possible to configure not to support transition (recommended).
  - IDNA Conversion includes normalization as per IDNA (good!).
  - Check if there are errors in the conversion by calling info.hasErrors().
  - For IDNs, set the options to restrict the validation and use to IDNA2008.
  - The static methods implement IDNA2003, and non-static methods implement IDNA2008.

Convert U-Label => A-Label: Java

```
import com.ibm.icu.text.IDNA;
public static String convertULabeltoALabel(String Ulabel) {
        String Alabel = "";
        final IDNA idnaInstance = IDNA.getUTS46Instance(IDNA.NONTRANSITIONAL_TO_ASCII
                | IDNA.CHECK_BIDI
                | IDNA.CHECK_CONTEXTJ
                | IDNA.CHECK_CONTEXTO
                | IDNA.USE_STD3_RULES);

        StringBuilder output = new StringBuilder();
        IDNA.Info info = new IDNA.Info();
        idnaInstance.nameToASCII(Ulabel, output, info);
        Alabel = output.toString();
        if (!info.hasErrors()) {
            return Alabel;
        } else {
            //Conversion fails
            return info.getErrors().stream().toString();
        }

    }
```

Convert U-Label => A-Label: Java

```
import com.ibm.icu.text.IDNA;
public static String convertALabeltoULabel(String Alabel) {
        String Ulabel = "";
        final IDNA idnaInstance = IDNA.getUTS46Instance(IDNA.NONTRANSITIONAL_TO_ASCII
                | IDNA.CHECK_BIDI
                | IDNA.CHECK_CONTEXTJ
                | IDNA.CHECK_CONTEXTO
                | IDNA.NONTRANSITIONAL_TO_UNICODE
                | IDNA.USE_STD3_RULES);
        StringBuilder output = new StringBuilder();
        IDNA.Info info = new IDNA.Info();
        idnaInstance.nameToUnicode(Alabel, output, info);
        Ulabel = output.toString();
        if (!info.hasErrors()) {
            return Ulabel;
        } else {
            return info.getErrors().stream().toString();
        }
    }
```

Domain Name Validation

Validating Domain Name

- 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

Validate Domain Name: Python

```
import unicodedata #library for normalization
import idna        #library for conversion
domainName = 'مصر.صحة'
try:
    domainName_normalized = unicodedata.normalize('NFC', domainName) #normalize to NFC
    print(domainName_normalized)
    #U-label to A-label
    domainName_alabel = idna.encode(domainName_normalized).decode("ascii")
    print(domainName_alabel)
except idna.IDNAError as e:
#invalid domain as per IDNA 2008
print("Domain '{domainName}' is invalid: {e}")
except Exception as e:
      print("ERROR: {e}")
```

Validate Domain Name: Java

```
import com.ibm.icu.text.IDNA;
public static boolean isValidDomain(String DomainName) {
        String Alabel = "";
        final IDNA idnaInstance = IDNA.getUTS46Instance(IDNA.NONTRANSITIONAL_TO_ASCII
                | IDNA.CHECK_BIDI
                | IDNA.CHECK_CONTEXTJ
                | IDNA.CHECK_CONTEXTO
                | IDNA.USE_STD3_RULES);

        StringBuilder output = new StringBuilder();
        IDNA.Info info = new IDNA.Info();
        idnaInstance.nameToASCII(DomainName, output, info);
        Alabel = output.toString();
        if (!info.hasErrors()) {
            return true;
        } else {
            //Conversion fails
            return false;
        }

    }
```

Domain Name Resolution

Domain Name Resolution

- After validation, a software would then use the domain name identifier as:
  - A domain name to be resolved in the DNS.

- Traditional way of doing hostname resolution and sockets resolution cannot be used for IDNs.

- We need to do following:
  1. Take user input and normalize
  2. Convert U-label to A-label (IDNA2008)
  3. Use A-label for hostname resolution

Domain Name Resolution – Python

```
import socket
import unicodedata
import idna
domainName=''
try:
    #normalize domain Name
    domainName_normalized = unicodedata.normalize('NFC', domainName)
    #Convert U-label to A-label form
    domainName_alabelForm = idna.encode(domainName_normalized).decode("ascii")
    #get IP address of the domain
    ip = socket.gethostbyname(domainName_alabelForm)
    print(ip)
except Exception as ex:
    print(ex)
```

Domain Name Resolution – Java

- Normalization and U-label to A-label conversion is same as discussed before.

```
import java.net.InetAddress;

 try {

            InetAddress ad = InetAddress.getByName(domainNameAlabelForm);

            String ip = ad.getHostAddress(); // returns ip for domain

            System.out.println(ip);


        } catch (Exception ex) {

            System.out.println(ex.toString()); //Unknown host exception

        }
```

Domain Name Storage

- We need to ensure that database supports and configure for UTF-8.
- SQL, e.g., MySQL, Oracle, Microsoft SQL Server.
    - Set domain names to max: 255 octets, 63 octets per label.
        - In UTF-8 native, variable length.
    - Recommendation to use variable length String columns.
    - Consider/verify the object-relational mapping (ORM) driver/tool if you are using one.
- noSQL, e.g., MongoDB, CouchDB, Cassandra, HBase, Redis, Riak, Neo4J.
    - Already UTF-8 variable length.

1. Store and retrieve either U-label or A-label in a field consistently.
2. You can also store both U-label and A-label in separate fields.

Email Address Internationalization (EAI)

Email Address

- Email address syntax: mailboxName@domainName.
  - Email has a mailboxName.
  - Email has a domainName.
    - The domainName can be ASCII or IDN.
    - For example:

      myname@example.org

      myname@xn--exmple-xta.ca

EAI

- EAI has the mailboxName in Unicode (in UTF-8 format).
- The domainName can be ASCII or IDN.
  - For example:
    - kévin@example.org
    - すし@ xn--exmple-xta.ca
    - すし@快手.游戏.

Email Addresses Form

- name@exâmple.ca and name@xn--exmple-xta.ca represent equivalent email address.
- Application should be able to treat both forms as equivalent.
- Internally consistently use A-label or U-label, but don’t mix A-label and U-label.
- Technical Recommendation: Backend processing should be in A-label, and U-label for visual inspection.
- For example, new user registration in application with equivalent A-label.

Email Validation: Email Regular Expressions (Regex)

- Basic: something@something.
  - ^(.+)@(.+)$
- From owasp.org (security):
  - [^[a-zA-Z0-9_+&*-]+(?:\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,7}$].
    - Does not support EAI, i.e., mailbox name in UTF8 not allowed: [a-zA-Z0-9_+&*-].
    - Does not support ASCII TLD longer than 7 characters: [a-zA-Z]{2,7}.
    - Does not support U-labels in IDN TLD: [a-zA-Z].
  - But OWASP is THE reference for security.
    - Therefore, you may end up fighting with your security team to use a UA-compatible Regex instead of the “standard” one from OWASP.

Email Regular Expressions (Regex)

- Example of Regex suggested in various forums: ex: List of proposals
  - ^[A-Za-z0-9+_.-]+@(.+)$ does not support UTF8 in mailbox name.
  - ^[a-zA-Z0-9_!#$%&’*+/=?`{|}~^.-]+@[a-zA-Z0-9.-]+$ does not support U-labels.
  - ^[a-zA-Z0-9_!#$%&’*+/=?`{|}~^-]+(?:\\.[a-zA-Z0-9_!#$%&’*+/=?`{|}~^-]+)*@[a-zA-Z0-9-]+(?:\\.[a-zA-Z0-9-]+)*$ does not support U-labels.
  - ^[\\w!#$%&’*+/=?`{|}~^-]+(?:\\.[\\w!#$%&’*+/=?`{|}~^-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6}$ have length restrictions for the TLD between 2 – 6 characters.

- One can come up with an EAI-IDN compatible regex using various Unicode codepoints characteristics.
  - For IDN it would be like a reimplementation of the IDNA protocol tables in regex!

- Given that both sides of an EAI may have UTF8, then one regex for an EAI could be .*@.* which is only verifying the presence of the ‘@’ character.

Validate Email

Email Addresses Validation

- Email has a mailboxName.
- Email has a domainName.
- DomainName validation same as before.
- mailboxName validation require a valid UTF8 String.
- Local administrator defines policy for mailboxName.
    - Gmail policy: firstname.lastname@gmail.com is equivalent to firstnamelastname@gmail.com.
- Guidelines for mailboxName are available by UASG.

EAI Validation - Python

```python
from email_validator import validate_email,EmailNotValidError
logger = logging.getLogger(__name__)
try:
    # As part of process it performs DNS resolution
    # Normalizes email addresses automatically
    # Supports internationalized domain names
    validated = validate_email(email_address, check_deliverability=True)
    print(validated)
    logger.info("'{address}' is a valid email address")
    print("'{address}' is a valid email address")
except EmailNotValidError as e:
    print("'{address}' is not a valid email address: {e}")
except Exception as ex:
    print("Unexpected Exception")
```

EAI Validation - Java

- Apache Common Validator:
  - Has domain and email validators.
  - Do not use as it relies on a static list of TLDs! OUTDATED!

EAI Validation - Java

```
/**
     * Download the list of TLDs on ICANN website
     */
    public static String[] retrieveTlds() {
        String IANA_TLD_LIST_URL = "https://data.iana.org/TLD/tlds-alpha-by-domain.txt";
        StringBuilder out = new StringBuilder();
        try (BufferedInputStream in = new BufferedInputStream(
                new URL(IANA_TLD_LIST_URL).openStream())) {
            byte[] dataBuffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
                out.append(new String(dataBuffer, 0, bytesRead));
            }
        } catch (IOException e) {
            // handle exception
        }
        return Arrays.stream(out.toString().split("\n"))
                .filter(s -> !s.startsWith("#"))
                .map(String::toLowerCase).distinct().toArray(String[]::new);
    }
```

EAI Validation - Java

```
public static DomainValidator createDomainValidatorInstance(String domain,
       boolean use_actual_domains) {
     List<Item> domains = new ArrayList<>();
     if (use_actual_domains) {
       domains.add(new Item(GENERIC_PLUS, retrieveTlds()));
     } else {
       String tld = domain;
       if (domain.contains(".")) {
         tld = domain.substring(domain.lastIndexOf(".") + 1);
       }
       // Convert TLD to A-Label
       String domainConverted = convertULabeltoALabel(tld);
       // if there is an error, do nothing, validator will fail
       if (domainConverted!="") {
         domains.add(new Item(GENERIC_PLUS, new String[]{domainConverted}));
       }
     }

     return DomainValidator.getInstance(false, domains);
   }
```

EAI Validation - Java

```
public static  boolean isValidEmail(String emailaddress){
        emailaddress = Normalizer2.getNFCInstance().normalize(emailaddress);
        String[]emailparts = emailaddress.split("@");
        if(emailparts.length==2){
            String mailboxname = emailparts[0];
            String domainName = emailparts[1];
            String domainNameAlabelForm =convertULabeltoALabel(domainName);
            try {

                  EmailValidator em = new EmailValidator(false, false,
                        createDomainValidatorInstance(domainName,true));
                    if(em.isValid(mailboxname+"@"+domainNameAlabelForm)){
                        return true;
                    }

                return false;
            } catch (Exception ex) {
                System.out.println(ex.toString());
                return false;
            }
        }
        else{
            return false;
        }
}
```

Sending and Receiving Email

Sending and Receiving

- We need to be able to send to either form:
  - mailboxName-UTF-8@A-labelform.
  - mailboxName-UTF-8@U-labelform.

- We need to be able to receive to either form:
  - mailboxName-UTF-8@A-labelform.
  - mailboxName-UTF-8@U-labelform.

- Storage of email should be consistent with domain name in either A-label or U-label form.

- Backend send/receive should be managed by mail server.

- Handover process (Front end application email server).
  - Libraries used in handover process should be EAI Compliant.
  - Mail server should also be EAI compatible.
    - How to make mail server EAI compatible is out of scope of this training?

Sending and Receiving – Python

- **Smtplib** can be used to send EAI-compliant emails.
- It does not validate the domain compliance with IDNA 2008, therefore another validation method should be used before trying to send an email.
    - For instance, using the email-validator library.

Sending and Receiving – Python

```
try:
    to = 'kévin@example.com'
    local_part, domain = to.rsplit('@', 1)
    domain_normalized= unicodedata.normalize('NFC',domain)#normalize domain name
    to = '@'.join((local_part,idna.encode(domain_normalized).decode('ascii’)))#convert U-label to A-label
    validated = validate_email(to, check_deliverability=True) #validate email address
    if validated:
        host=''
        port=''
        smtp = smtplib.SMTP(host, port)
        smtp.set_debuglevel(False)
        smtp.login(‘useremail’,’password')
        sender=‘ua@test.org'
        subject='hi'
        content='content here'
        msg = EmailMessage()
        msg.set_content(content)
        msg['Subject'] = subject
        msg['From'] = sender
        msg['to']=to
        smtp.send_message(msg, sender, to)
        smtp.quit()
        logger.info("Email sent to '{to}'")
except smtplib.SMTPNotSupportedError:
    # The server does not support the SMTPUTF8 option, you may want to perform downgrading
    logger.warning("The SMTP server {host}:{port} does not support the SMTPUTF8 option")
    raise
```

Sending and Receiving – Java

- **Jakarta Mail** can be used for sending email.

```
import com.sun.mail.smtp.SMTPTransport;
import jakarta.mail.Message;
import jakarta.mail.MessagingException;
import jakarta.mail.PasswordAuthentication;
import jakarta.mail.Session;
import jakarta.mail.Transport;
import jakarta.mail.internet.InternetAddress;
import jakarta.mail.internet.MimeMessage;
import java.util.Date;
import java.util.Properties;
```

Sending and Receiving – Java

```
public static boolean sendEmail(String to, String host, String sender,
String subject, String content,String username,String password){
        if(isValidEmail(to))
        {
            Properties props =  new Properties();
            props.put("mail.smtp.host", host);
            props.put("mail.smtp.port", "587");
            props.put("mail.smtp.auth", "true");
            props.put("mail.smtp.starttls.enable", "true");
            // enable UTF-8 support, mandatory for EAI support
            props.put("mail.mime.allowutf8", true);
            Session session = Session.getInstance(props,
            new jakarta.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
               return new PasswordAuthentication(username, password);
                }
             });
```

Sending and Receiving – Java(2)

```
/*
       * Jakarta mail is EAI compliant with 2 issues:
       *   - it rejects domains that are not NFC normalized
       *   - it rejects some unicode domains
       * In such case, first try to normalize, then convert domain to A-label. We do normalization
       * first to get an email address the closest possible to the user input because once
       * converted in A-label it may be displayed as is to the user.
       */

String[] add_parts = to.split("@");

String mailboxName = add_parts[0];

String domainName = add_parts[1];

String domainNameNormalized =Normalizer2.getNFCInstance().normalize(domainName);

String domainNameAlabelForm = convertULabeltoALabel(domainNameNormalized);

String compliantTo = mailboxName+"@"+domainNameAlabelForm;
```

Sending and Receiving – Java(3)

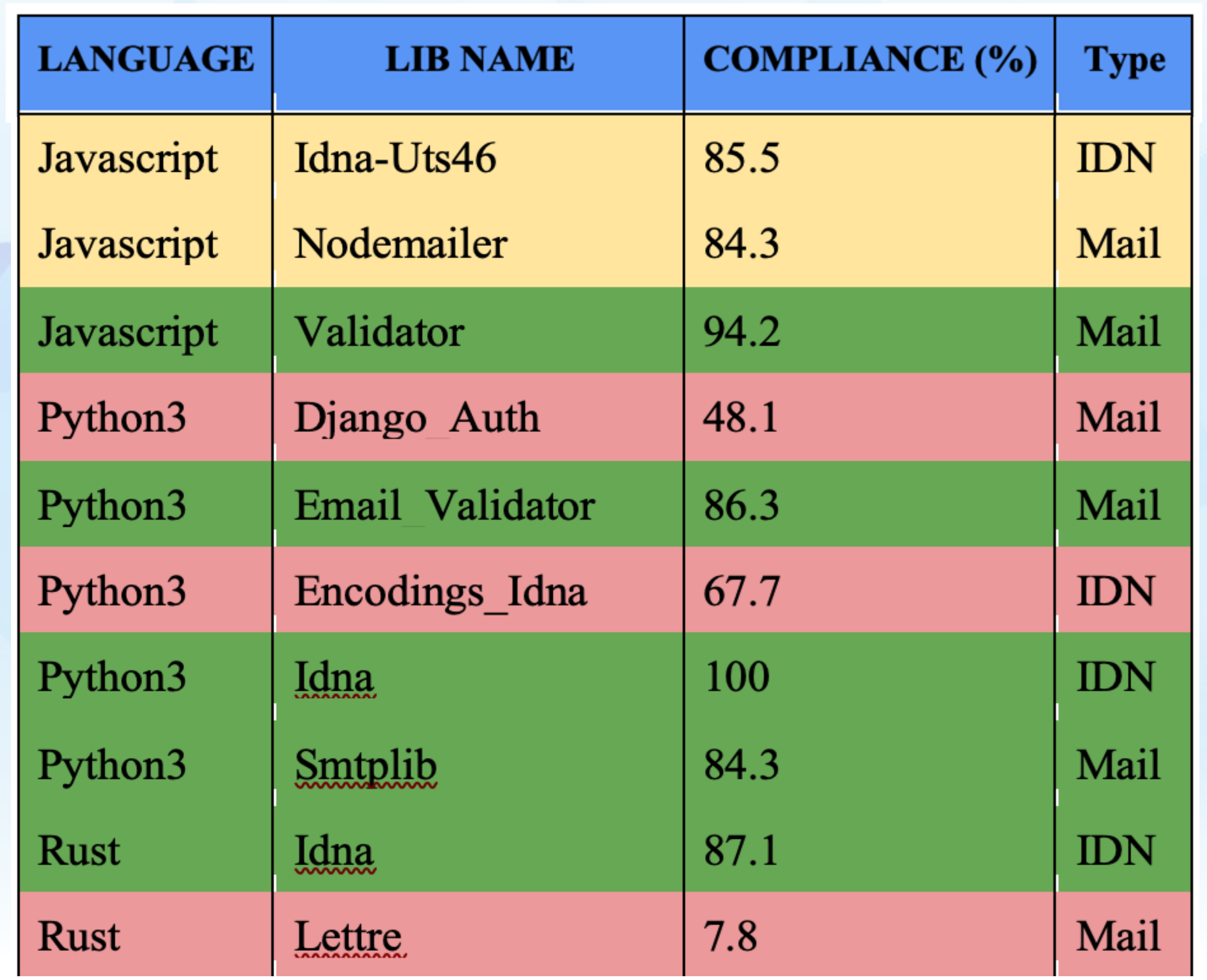
```
try (Transport transport = session.getTransport())
            {
                  if (transport instanceof SMTPTransport && !((SMTPTransport) transport).supportsExtension("SMTPUTF8")) {
                   try {
                           MimeMessage message = new MimeMessage(session);
                           //set message headers for internationalized content
                           message.addHeader("Content-type", "text/HTML; charset=UTF-8");
                           message.addHeader("Content-Transfer-Encoding", "8bit");
                           message.addHeader("format", "flowed");

                           message.setFrom(new InternetAddress(sender));
                           message.setSubject(subject, "UTF-8");
                           message.setText(content, "UTF-8");
                           message.setSentDate(new Date());
                           message.setRecipient(Message.RecipientType.TO, new InternetAddress(compliantTo));
                           Transport.send(message);
                           return true;
                         } catch (Exception e) {
                           System.out.println(String.format("Failed to send email to %s: %s", to, e));
                         }
                }
                else
                { return false;}
          } catch (MessagingException e) {
            // ignore
          }
        }
        return false;}
```

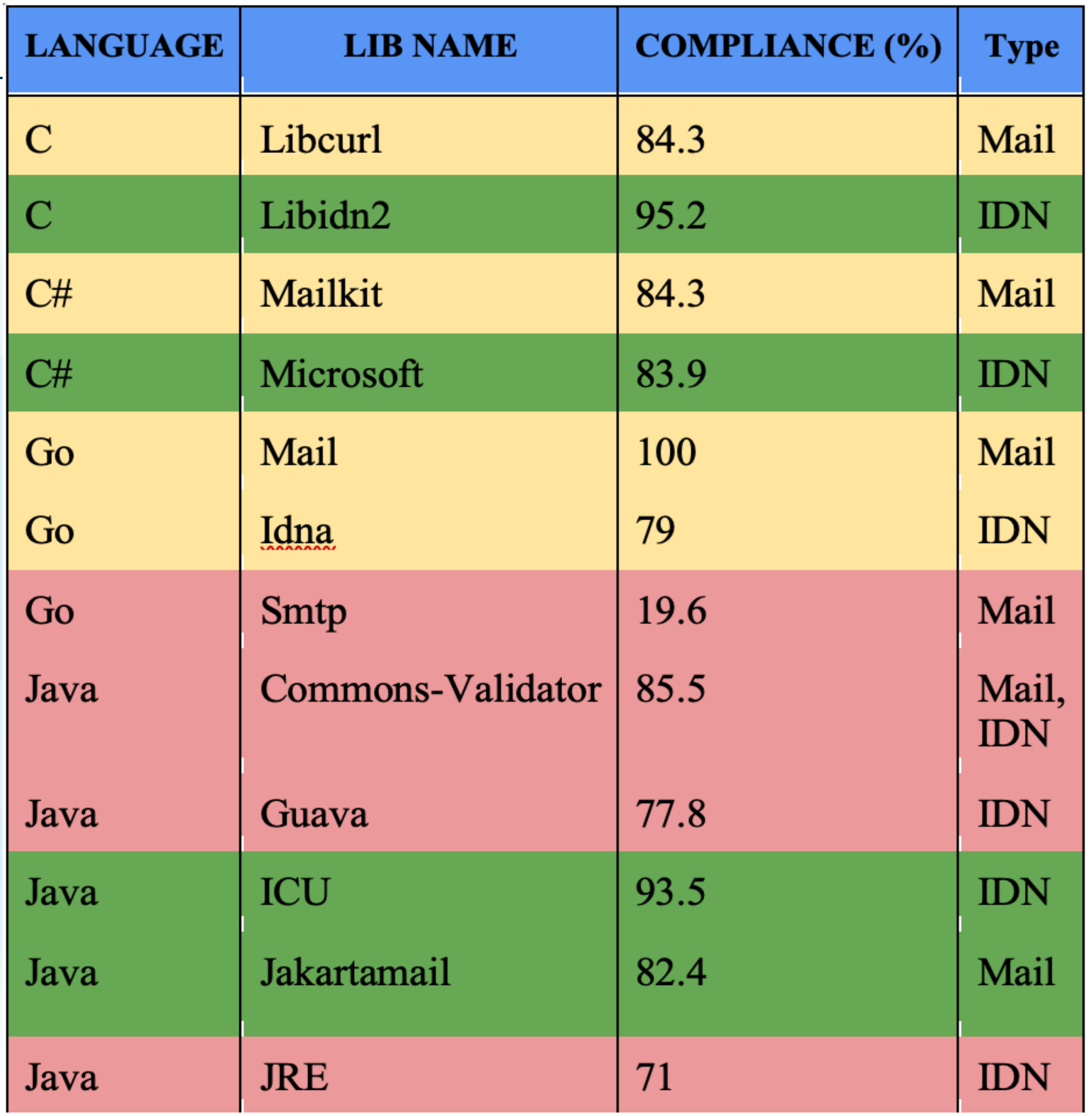
Conclusion

Prog. Languages Support

UASG018A – note that some have improved since.



LANGUAGE	LIB NAME	COMPLIANCE (%)	Type
Javascript	Idna-Uts46	85.5	IDN
Javascript	Nodemailer	84.3	Mail
Javascript	Validator	94.2	Mail
Python3	Django_Auth	48.1	Mail
Python3	Email_Validator	86.3	Mail
Python3	Encodings_Idna	67.7	IDN
Python3	Idna	100	IDN
Python3	Smtplib	84.3	Mail
Rust	Idna	87.1	IDN
Rust	Lettre	7.8	Mail



LANGUAGE	LIB NAME	COMPLIANCE (%)	Type
C	Libcurl	84.3	Mail
C	Libidn2	95.2	IDN
C#	Mailkit	84.3	Mail
C#	Microsoft	83.9	IDN
Go	Mail	100	Mail
Go	Idna	79	IDN
Go	Smtp	19.6	Mail
Java	Commons-Validator	85.5	Mail, IDN
Java	Guava	77.8	IDN
Java	ICU	93.5	IDN
Java	Jakartamail	82.4	Mail
Java	JRE	71	IDN

Conclusion

- Be aware that UA identifiers may not be fully supported in software and libraries.
- Use the right libraries and frameworks.
- Adapt your code to properly support UA.
- Do unit and system testing using UA test cases to ensure that your software is UA ready.

Get Involved!

- For more information on UA, email info@uasg.tech or UAProgram@icann.org
- Access all UASG documents and materials at: https://uasg.tech and https://icann.org/ua

Some Relevant Materials

- See https://uasg.tech for a complete list of reports.
  - Universal Acceptance Quick Guide: UASG005
  - Introduction to Universal Acceptance: UASG007
  - Quick Guide to EAI: UASG014
  - EAI – A Technical Overview: UASG012
  - UA Compliance of Some Programming Language Libraries and Frameworks – UASG018A
  - EAI – Evaluation of Major Email Software and Services: UASG021B
  - Universal Acceptance Readiness Framework: UASG026
  - Considerations for Naming Internationalized Email Mailboxes: UASG028
  - Evaluation of EAI Support in Email Software and Services Report: UASG030
  - UA of Content Management Systems (CMS) Phase 1 – WordPress: UASG032
  - UA-Ready Code Samples in Java, Python, and JavaScript - UASG043

Thank You

One World, One Internet

Visit us at **icann.org** Email: sarmad.hussain@icann.org

@icann

facebook.com/icannorg

youtube.com/icannnews

flickr.com/icann

linkedin/company/icann

slideshare/icannpresentations

soundcloud/icann