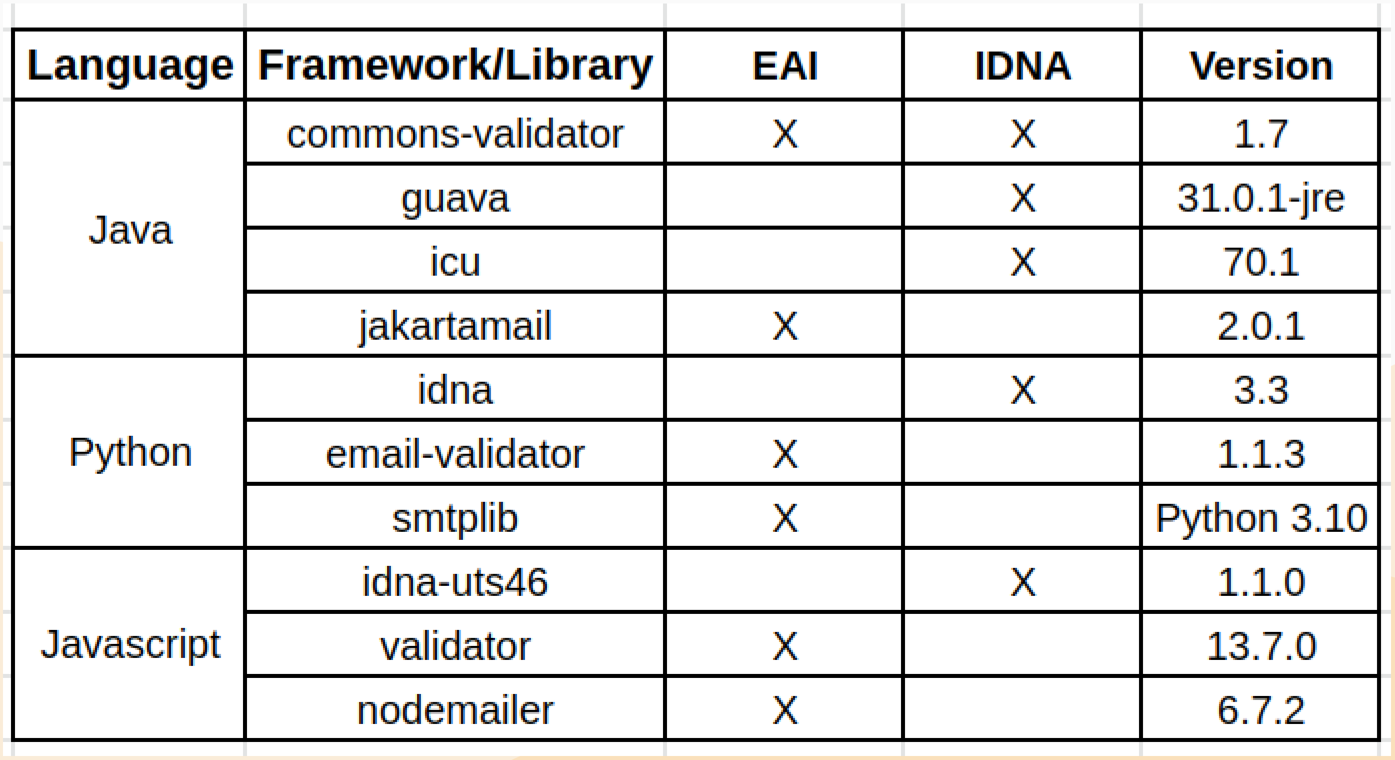
Showcasing UA Readiness and Conducting Bug Reporting with Programming Languages

Cofomo / August 9th, 2022

Universal Acceptance

Target Languages & Libraries

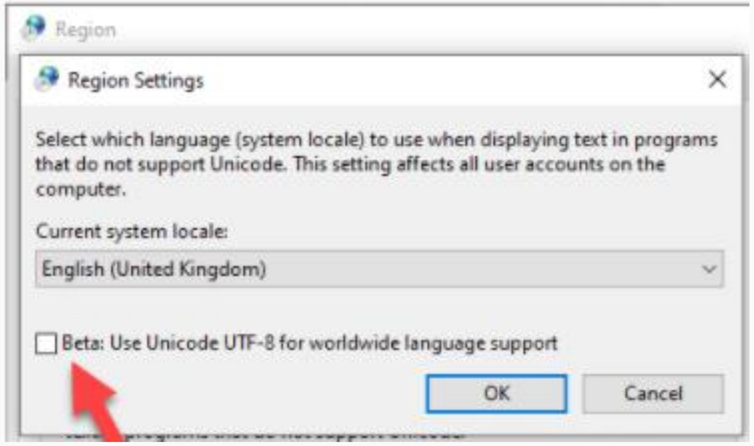
- 10 libraries



Language	Framework/Library	EAI	IDNA	Version
Java	commons-validator	X	X	1.7
	guava		X	31.0.1-jre
	icu		X	70.1
	jakartamail	X		2.0.1
Python	idna		X	3.3
	email-validator	X		1.1.3
	smtplib	X		Python 3.10
Javascript	idna-uts46		X	1.1.0
	validator	X		13.7.0
	nodemailer	X		6.7.2

General recommendations

- Centralize processing of emails and domain names to avoid using different versions of the protocols;
- Refer to the general guidelines in UASG 026 UA Readiness Framework;
- Stay up to date: JavaMail -> JakartaMail (compliance with EAI only since 1.6.4 in 2019);
- Avoid validations based on a static list of TLDs (During a 10 day period in November 2019, 7 TLDs were removed from the root);
- Check "UTF-8 support beta feature" if developing/deploying on Windows;



Select which language (system locale) to use when displaying text in programs that do not support Unicode. This setting affects all user accounts on the computer.

Current system locale:

English (United Kingdom)

☐ Beta: Use Unicode UTF-8 for worldwide language support

Java

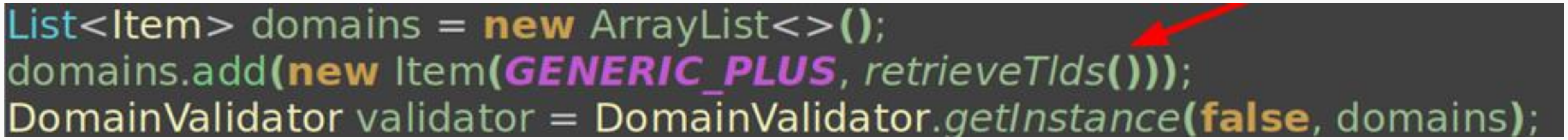
- IDN:
  - Avoid using *java.net.IDN* since it is based on IDNA2003;
  - Prefer *com.ibm.icu.text.IDNA* when the choice is available;
- EAI:
  - Ensure using the latest JakartaMail, the re-brand (maintained) version of JavaMail by Eclipse;

Next slides discussed possible and correct usage of the following libraries:

1. Commons Validator
2. Guava
3. ICU
4. JakartaMail

Java: Common Validators

- IDNA:
  - Uses a static list of TLDs
  - However, one can provide his/her own static list as input to the library.
  - By fetching the latest updated list of TLD from IANA and caching it for a short period of time, one can validate domains the following way:

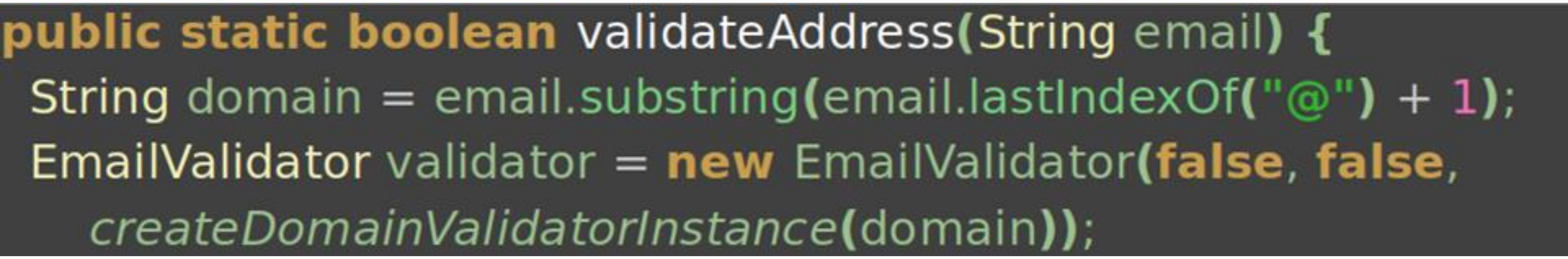


```
List<Item> domains = new ArrayList<>();
domains.add(new Item(GENERIC_PLUS, retrieveTlds()));
DomainValidator validator = DomainValidator.getInstance(false, domains);
```

  - It is also possible provide the TLD of the domain to be tested in A-label form directly to the validator being instantiated to avoid the process of validating the TLD against IANA list.
  - Once the domain validator is properly instantiated, one can provide it to the *UrlValidator*

Java: Common Validators

- EAI:
  - Like *UrlValidator*, *EmailValidator* accepts a *DomainValidator* instance:



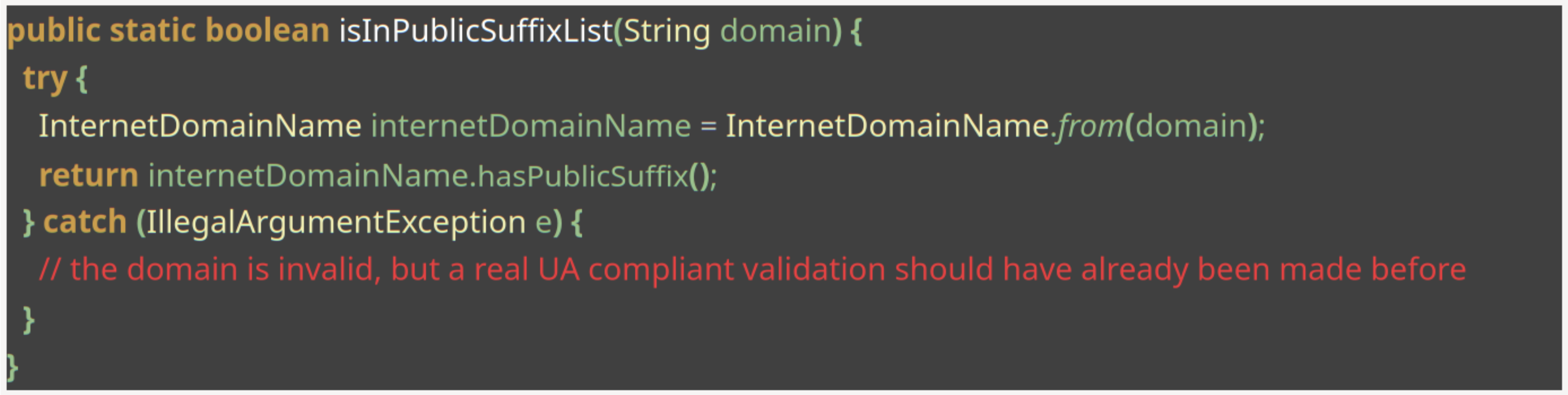
```java
public static boolean validateAddress(String email) {
  String domain = email.substring(email.lastIndexOf("@") + 1);
  EmailValidator validator = new EmailValidator(false, false,
    createDomainValidatorInstance(domain));
```

"*lastIndexOf*" is important here, because emails can contains '@' in the local part if it is wrapped in double quotes.

- Bug report:
  - Domain validator converts the domain to A-label with java.net.IDN that is only IDNA 2003 compliant

Java: Guava

* IDNA
  * Only performs some basic check (label length, no ending dash, ...) on domain names
  * IDNs are accepted by default without further validation so another validation should be performed
  * Can only be used to check whether the domain is in the public suffix list, which is not a reliable method to consider a domain as invalid

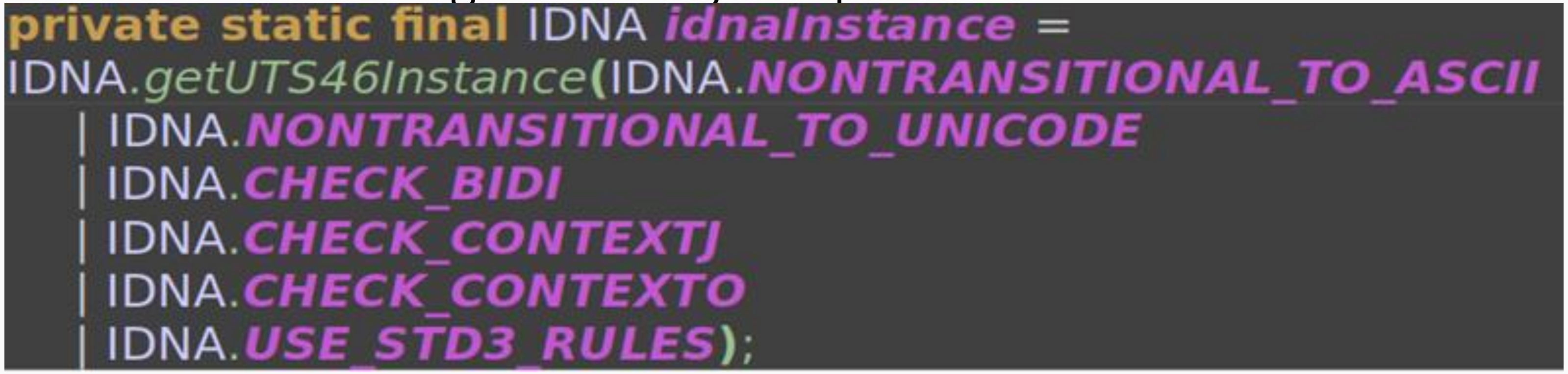


```
public static boolean isInPublicSuffixList(String domain) {
  try {
    InternetDomainName internetDomainName = InternetDomainName.from(domain);
    return internetDomainName.hasPublicSuffix();
  } catch (IllegalArgumentException e) {
    // the domain is invalid, but a real UA compliant validation should have already been made before
  }
}
```

* Bug report:
  * [Pull request to clarify the documentation about IDN]

Java: ICU

* IDNA
  * Convert domain to A-label prior to using it on wire. This would mitigate the risk induced by other libraries which may have a poor UA compliance level;
  * Combination of flags to be fully compliant with IDNA2008:

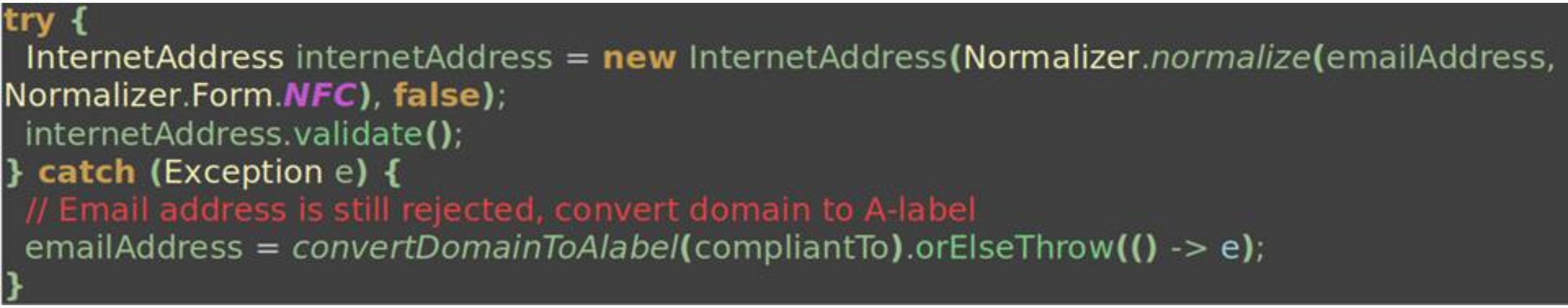


```
private static final IDNA idnaInstance =
IDNA.getUTS46Instance(IDNA.NONTRANSITIONAL_TO_ASCII
    | IDNA.NONTRANSITIONAL_TO_UNICODE
    | IDNA.CHECK_BIDI
    | IDNA.CHECK_CONTEXTJ
    | IDNA.CHECK_CONTEXTO
    | IDNA.USE_STD3_RULES);
```

* Bug report:
  * ICU performs some normalization on domain and therefore, some invalid characters as per IDNA 2008 but valid as per UTS46 are considered valid

Java: Jakarta Mail

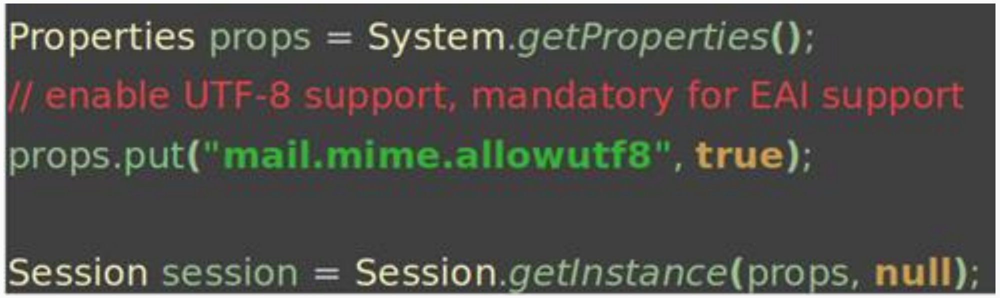
* EAI
  * Some valid Unicode characters as per IDNA 2008 are rejected, for such case, convert domain to A-label



```
try {
 InternetAddress internetAddress = new InternetAddress(Normalizer.normalize(emailAddress,
Normalizer.Form.NFC), false);
 internetAddress.validate();
} catch (Exception e) {
 // Email address is still rejected, convert domain to A-label
 emailAddress = convertDomainToAlabel(compliantTo).orElseThrow(() -> e);
}
```

where *convertDomainToAlabel* use ICU discussed earlier.

* Session transport must be initialized with *allowutf8*:

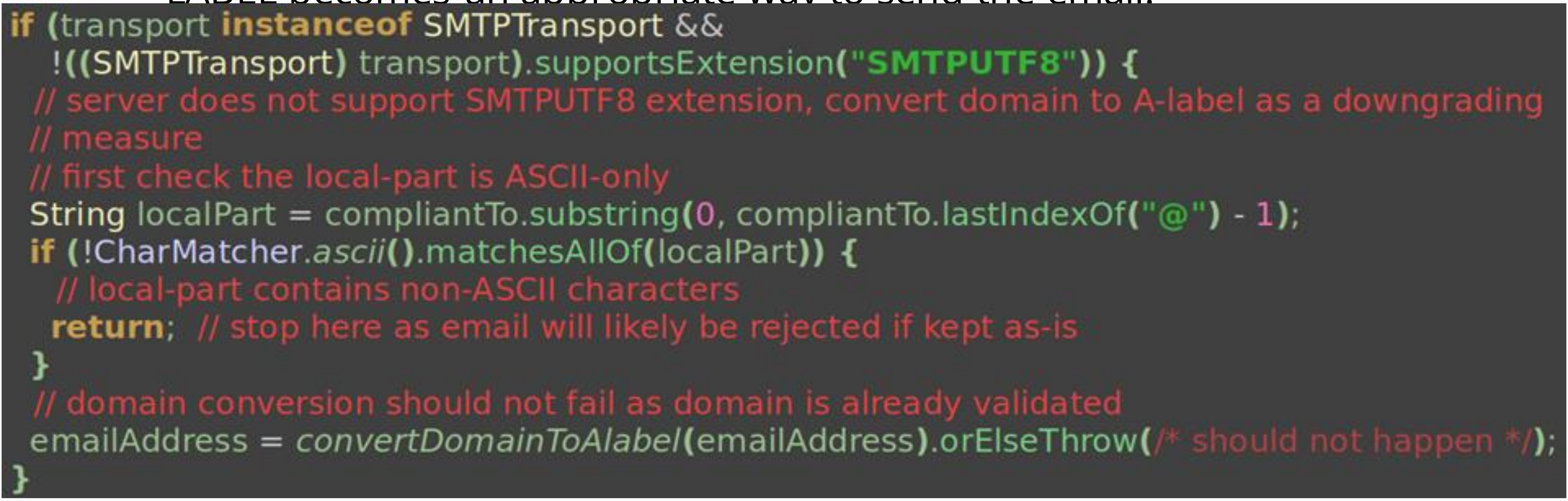


```
Properties props = System.getProperties();
// enable UTF-8 support, mandatory for EAI support
props.put("mail.mime.allowutf8", true);

Session session = Session.getInstance(props, null);
```

Java: Jakarta Mail

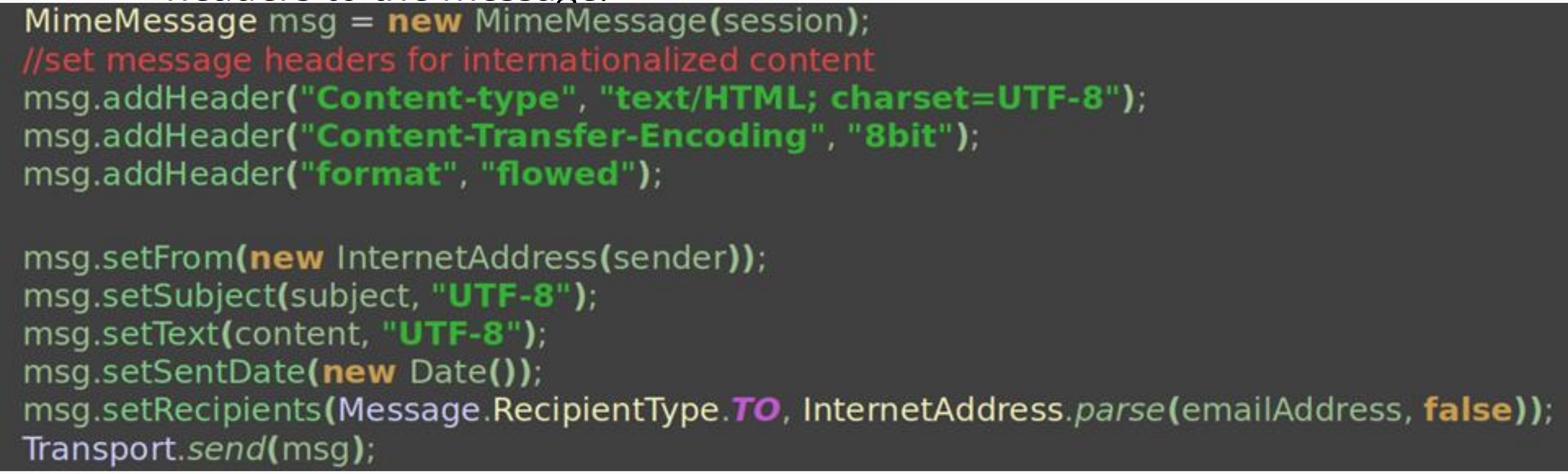
* EAI
    * In case the remote server does not support SMTPUTF8 extension and only the domain is a U-LABEL (localpart is ASCII), conversion to an A-LABEL becomes an appropriate way to send the email:



```java
if (transport instanceof SMTPTransport &&
    !((SMTPTransport) transport).supportsExtension("SMTPUTF8")) {
  // server does not support SMTPUTF8 extension, convert domain to A-label as a downgrading
  // measure
  // first check the local-part is ASCII-only
  String localPart = compliantTo.substring(0, compliantTo.lastIndexOf("@") - 1);
  if (!CharMatcher.ascii().matchesAllOf(localPart)) {
    // local-part contains non-ASCII characters
    return;  // stop here as email will likely be rejected if kept as-is
  }
  // domain conversion should not fail as domain is already validated
  emailAddress = convertDomainToAlabel(emailAddress).orElseThrow(/* should not happen */);
}
```

Java: Jakarta Mail

* EAI
    * Finally, when everything is validated and ready to send, add the proper headers to the message:



```
MimeMessage msg = new MimeMessage(session);
//set message headers for internationalized content
msg.addHeader("Content-type", "text/HTML; charset=UTF-8");
msg.addHeader("Content-Transfer-Encoding", "8bit");
msg.addHeader("format", "flowed");

msg.setFrom(new InternetAddress(sender));
msg.setSubject(subject, "UTF-8");
msg.setText(content, "UTF-8");
msg.setSentDate(new Date());
msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(emailAddress, false));
Transport.send(msg);
```

* Bug reports
    * [Some valid domains are considered invalid]()

Python

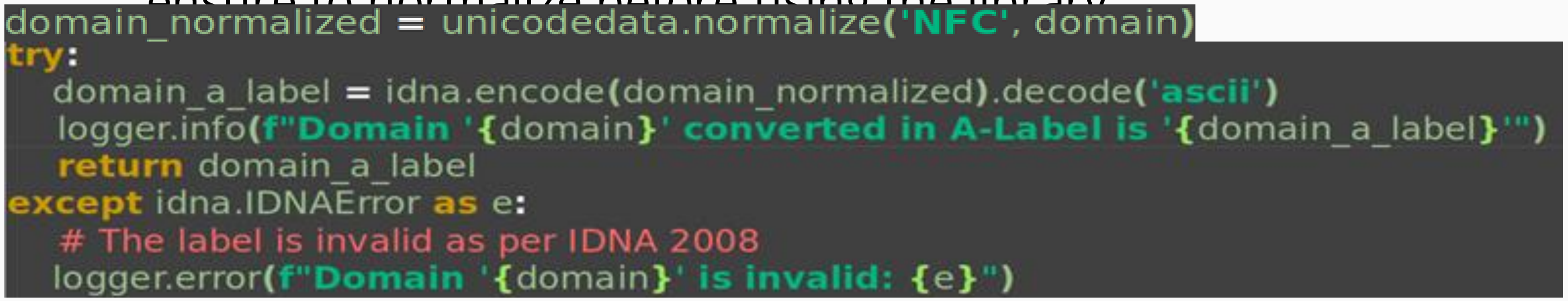
* IDN:
  * Avoid using Python native module encodings.idna which implement IDNA2003;
  * Prefer idna lib;

Next slides discussed possible and correct usage of the following libraries:

1. idna
2. email validator
3. smtplib

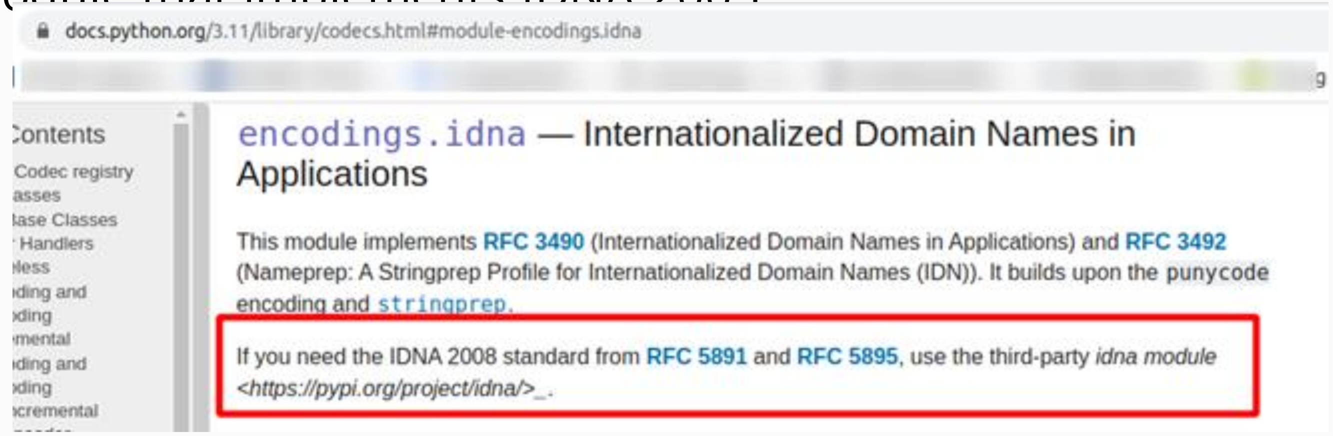
Python: IDNA

* IDNA
  * Fully IDNA 2008 compliant;
  * does not perform any normalization before converting the domain, ensure to normalize before using the library;



```python
domain_normalized = unicodedata.normalize('NFC', domain)
try:
    domain_a_label = idna.encode(domain_normalized).decode('ascii')
    logger.info(f"Domain '{domain}' converted in A-Label is '{domain_a_label}'")
    return domain_a_label
except idna.IDNAError as e:
    # The label is invalid as per IDNA 2008
    logger.error(f"Domain '{domain}' is invalid: {e}")
```

  * ongoing discussion to make it as a replacement to the Python idn module that implements IDNA 2003



docs.python.org/3.11/library/codecs.html#module-encodings.idna

Contents
Codec registry
asses
ase Classes
Handlers
less
ding and
ding
mental
ding and
ding
cremental

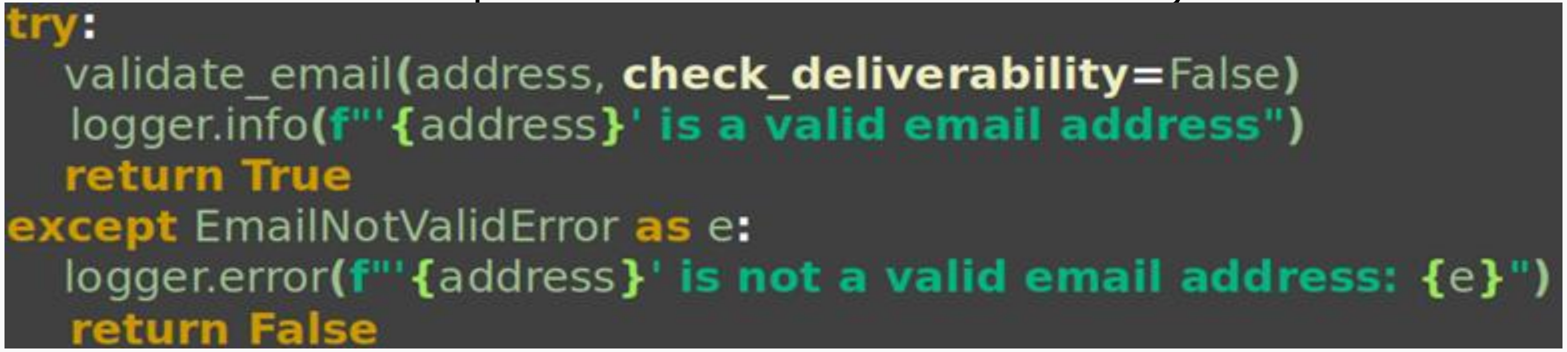
encodings.idna — Internationalized Domain Names in Applications

This module implements RFC 3490 (Internationalized Domain Names in Applications) and RFC 3492 (Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)). It builds upon the punycode encoding and stringprep.

If you need the IDNA 2008 standard from RFC 5891 and RFC 5895, use the third-party *idna module <https://pypi.org/project/idna/>_*.

Python: Email Validator

* EAI
  * Common email addresses validation (means that some valid email addresses per RFCs may be rejected, but only for edge cases);
  * EAI, however, is correctly supported and internationalized domain names validation is performed with the IDNA library

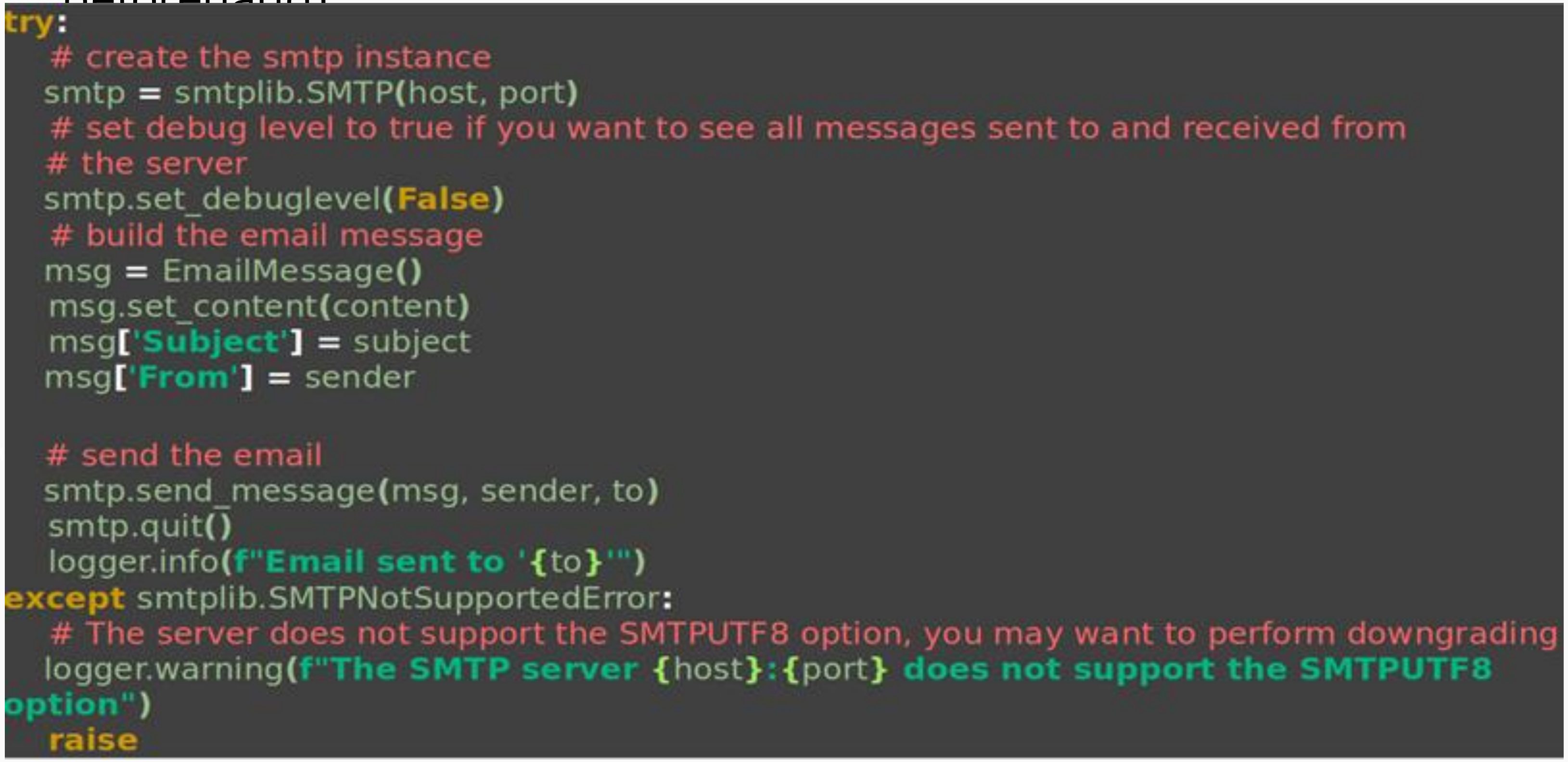


```
try:
    validate_email(address, check_deliverability=False)
    logger.info(f"'{address}' is a valid email address")
    return True
except EmailNotValidError as e:
    logger.error(f"'{address}' is not a valid email address: {e}")
    return False
```

  * *check_deliverability* option make a DNS query in order to ensure that the email address domain exists. Depending on the volume of email addresses to validate it can be interesting to enable it (here we disabled it)

Python: smtplib

* EAI
  * Used to send EAI compliant emails;
  * Supports EAI since Python 3.5.
  * Partial validation of the email address (please use email-validator beforehand):

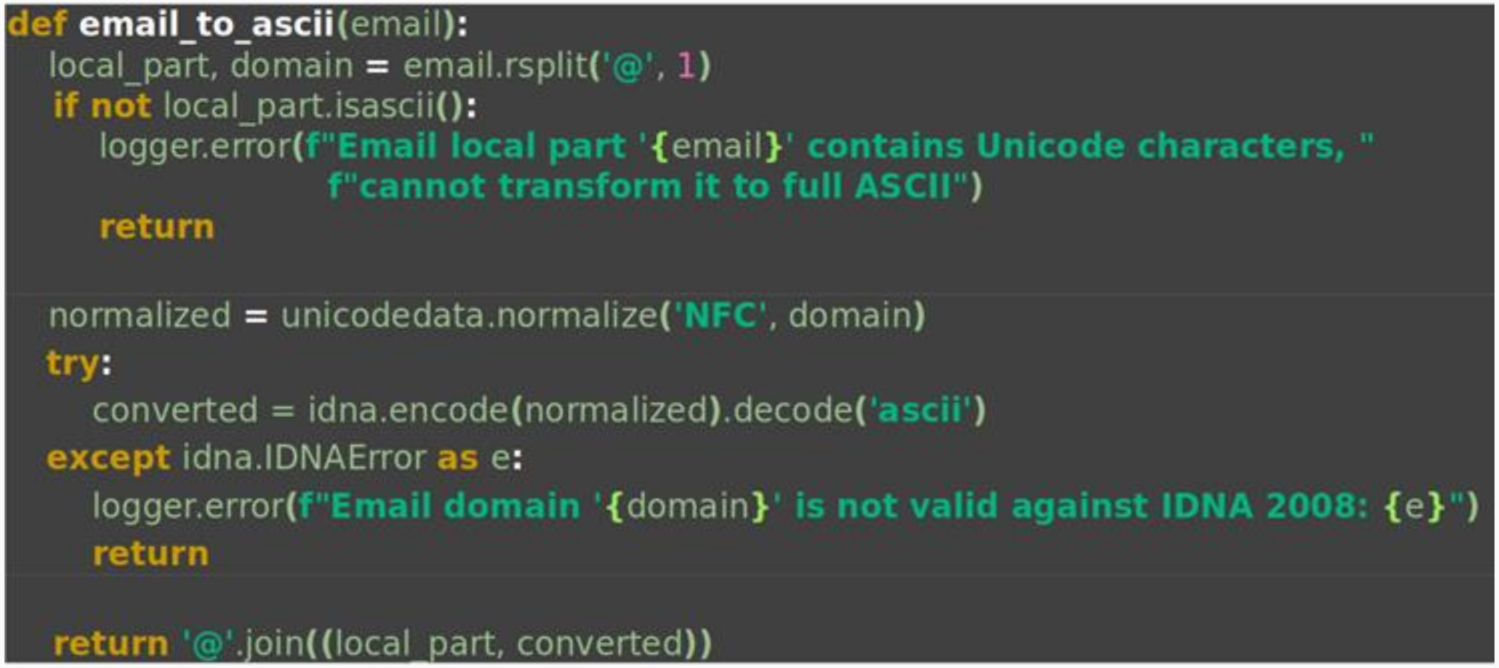


```
try:
    # create the smtp instance
    smtp = smtplib.SMTP(host, port)
    # set debug level to true if you want to see all messages sent to and received from
    # the server
    smtp.set_debuglevel(False)
    # build the email message
    msg = EmailMessage()
    msg.set_content(content)
    msg['Subject'] = subject
    msg['From'] = sender

    # send the email
    smtp.send_message(msg, sender, to)
    smtp.quit()
    logger.info(f"Email sent to '{to}'")
except smtplib.SMTPNotSupportedError:
    # The server does not support the SMTPUTF8 option, you may want to perform downgrading
    logger.warning(f"The SMTP server {host}:{port} does not support the SMTPUTF8
option")
    raise
```

Python: smtplib

* EAI
  * If the remote server does not support SMTPUTF8 option and local-part is ASCII only, email address can be downgraded by transforming domain into A-label



```python
def email_to_ascii(email):
    local_part, domain = email.rsplit('@', 1)
    if not local_part.isascii():
        logger.error(f"Email local part '{email}' contains Unicode characters, "
                     f"cannot transform it to full ASCII")
        return

    normalized = unicodedata.normalize('NFC', domain)
    try:
        converted = idna.encode(normalized).decode('ascii')
    except idna.IDNAError as e:
        logger.error(f"Email domain '{domain}' is not valid against IDNA 2008: {e}")
        return

    return '@'.join((local_part, converted))
```

Javascript

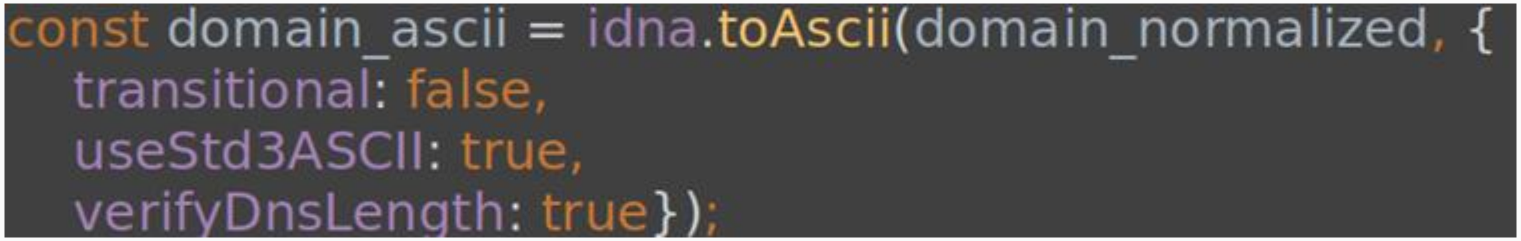
- IDN:
  - URL WHATWG native javascript object automatically converts to ASCII. This conversion is sometimes using IDNA2003 or IDNA2008 depending on the browser and the installation of NodeJS if server-side.
  - Please use a RFC's compliant library like *uri-js* instead of the URL object, especially when running code client-side.
  - Bug reports
    - Illegal characters in URL do not raise an exception (LDH rule)
    - IDNA2008 DISALLOWED characters are permitted in URL

Next slides discussed possible and correct usage of the following libraries:

1. Idna-uts46
2. validator
3. nodemailer

Javascript: idna-uts46

* IDNA
  * Known issues: doesn't support BIDI & contextual Rules;
  * Project no longer maintained. CentralNic fork (called *idna-uts46-hx*) took over the maintenance and ensure new Unicode versions are supported;

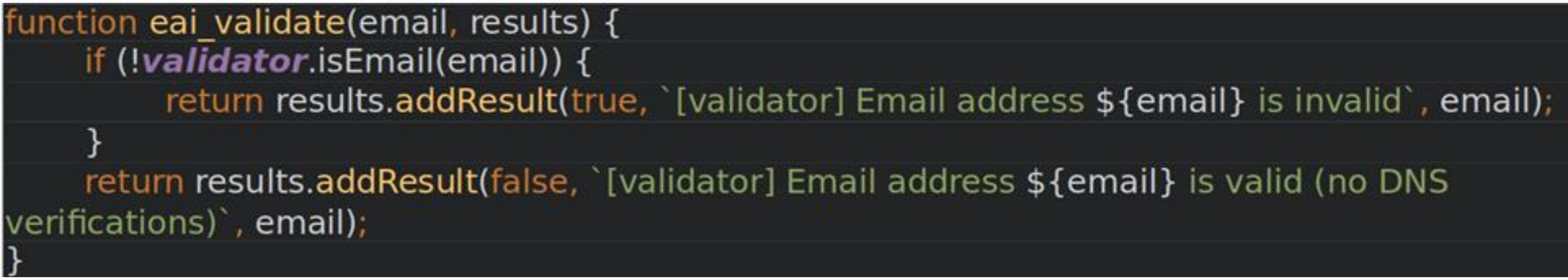


```
const domain_ascii = idna.toAscii(domain_normalized, {
  transitional: false,
  useStd3ASCII: true,
  verifyDnsLength: true});
```

* Transitional=false ensures we are on the IDNA2008 protocol;
* useStd3ASCII=true ensures we allow only LDH domains;
* verifyDnsLength=true ensures we do not allow a domain label to exceed 63 characters

Javascript: validator

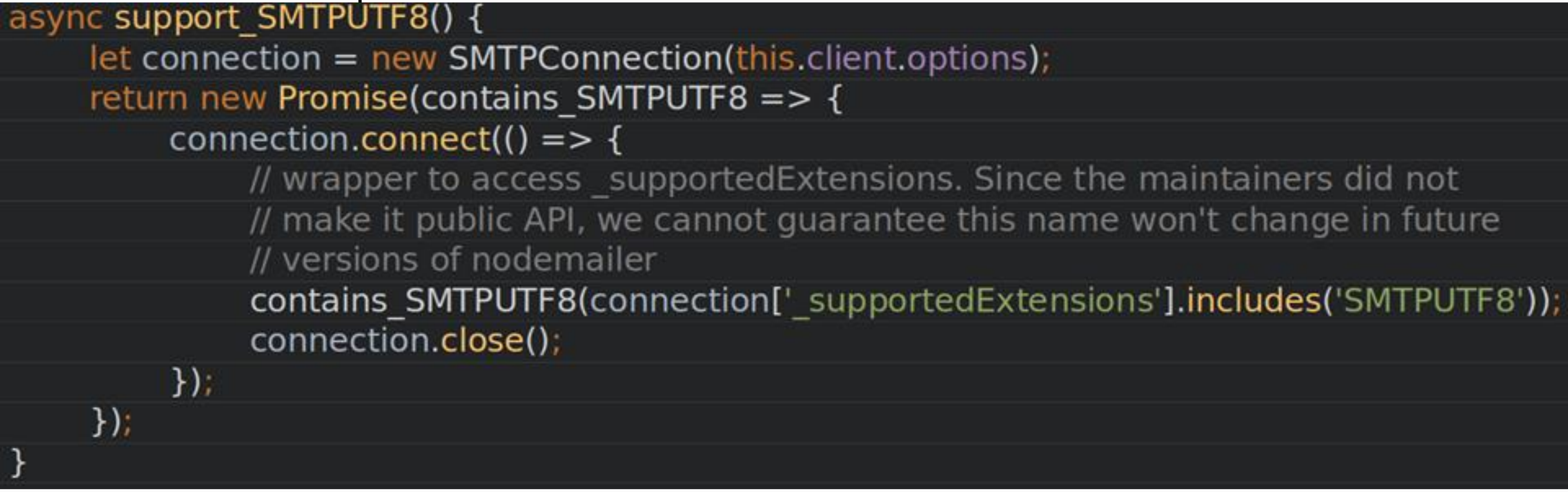
* EAI
  * Validation EAI compliant;
  * Can be use client-side and server-side as-is



```
function eai_validate(email, results) {
    if (!validator.isEmail(email)) {
        return results.addResult(true, `[validator] Email address ${email} is invalid`, email);
    }
    return results.addResult(false, `[validator] Email address ${email} is valid (no DNS
verifications)`, email);
}
```

Javascript: nodemailer

* EAI
  * Transform automatically the domain part into a A-LABEL when possible;
  * Checking compliance of remote with SMTPUTF8 is on the shoulder of the developer:

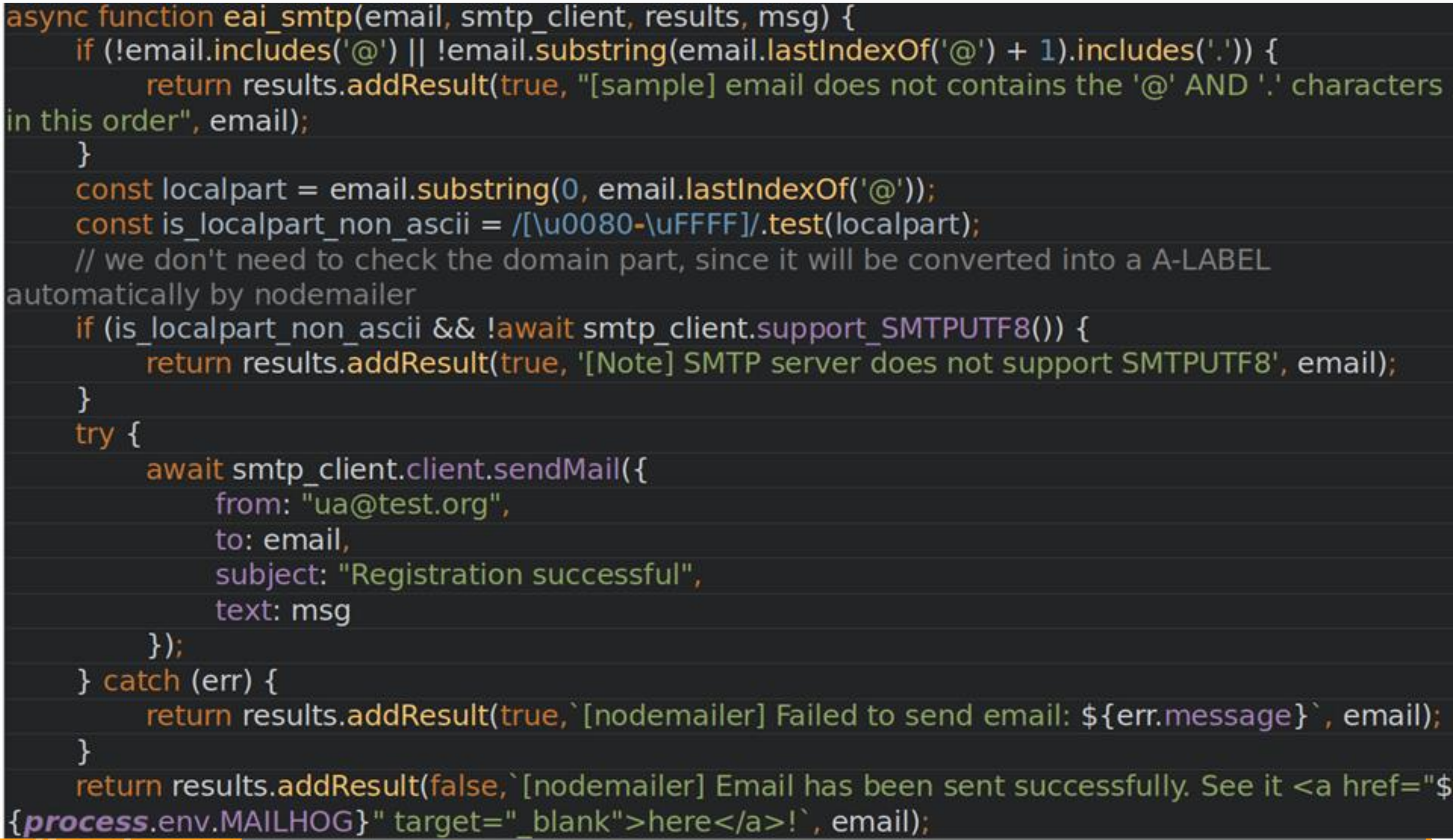


```
async support_SMTPUTF8() {
    let connection = new SMTPConnection(this.client.options);
    return new Promise(contains_SMTPUTF8 => {
        connection.connect(() => {
            // wrapper to access _supportedExtensions. Since the maintainers did not
            // make it public API, we cannot guarantee this name won't change in future
            // versions of nodemailer
            contains_SMTPUTF8(connection['_supportedExtensions'].includes('SMTPUTF8'));
            connection.close();
        });
    });
}
```

* Bug report:
  * No error for internationalized emails when the SMTP server doesn't support SMTPUTF8

Javascript: nodemailer

* EAI (complete example):



```
async function eai_smtp(email, smtp_client, results, msg) {
	if (!email.includes('@') || !email.substring(email.lastIndexOf('@') + 1).includes('.')) {
		return results.addResult(true, "[sample] email does not contains the '@' AND '.' characters
in this order", email);
	}
	const localpart = email.substring(0, email.lastIndexOf('@'));
	const is_localpart_non_ascii = /[\u0080-\uFFFF]/.test(localpart);
	// we don't need to check the domain part, since it will be converted into a A-LABEL
automatically by nodemailer
	if (is_localpart_non_ascii && !await smtp_client.support_SMTPUTF8()) {
		return results.addResult(true, '[Note] SMTP server does not support SMTPUTF8', email);
	}
	try {
		await smtp_client.client.sendMail({
			from: "ua@test.org",
			to: email,
			subject: "Registration successful",
			text: msg
		});
	} catch (err) {
		return results.addResult(true,`[nodemailer] Failed to send email: ${err.message}`, email);
	}
	return results.addResult(false,`[nodemailer] Email has been sent successfully. See it <a href="$
{process.env.MAILHOG}" target="_blank">here</a>!`, email);
```

Resources

- All resources, bug reports, detailed results and the likes are available at this url:

  https://cofomo.github.io/universal-acceptance/