Universal Acceptance Compliance of Some Programming Language Libraries and Frameworks

Cofomo inc

2021-09

# Introduction

This document describes the results of the third phase of the work done before. The results of the previous phase are available on the [UASG site](https://uasg.tech/software/). This new phase expands the previous work by adding mobile platforms libraries and PHP on the Linux & Windows platforms.

This second phase of the work verified the following languages and libraries for Universal Acceptance readiness on the linux platform exclusively:

This new third phase add 25 libraries-platforms to the set:

|  |  |  |  |
| --- | --- | --- | --- |
| **Language** | **Platform** | **Framework/Library** | **Versions** |
| PHP | Linux | cURL | PHP8.0 |
| PHP | Linux | emailValidator | 3.1.1 |
| PHP | Linux | Guzzle | 7.0 |
| PHP | Linux | intl | PHP8.0 |
| PHP | Linux | mailer | 6.5 |
| PHP | Linux | symfony | 5.3 |
| PHP | Windows | cURL | PHP8.0 |
| PHP | Windows | emailValidator | 3.1.1 |
| PHP | Windows | Guzzle | 7.0 |
| PHP | Windows | intl | PHP8.0 |
| PHP | Windows | mailer | 6.5 |
| PHP | Windows | symfony | 5.3 |
| PHP | Windows | mail (native) | PHP8.0 |
| Swift | iOS | AlamoFire | 5.4.4 |
| Swift | iOS | URLSession | iOS 14.4, Swift 5.3.2 |
| Swift | iOS | MessageUI | iOS 14.7.1, Swift 5.3.2 |
| Swift | iOS | IDNA-Cocoa | 870ba3e |
| Kotlin | Android | Apache | hc5-0.1.1 |
| Kotlin | Android | EmailIntent | Android 11 (API level 30) |
| Kotlin | Android | Fuel | 2.3.1 |
| Kotlin | Android | HttpUrlConnection | Android 11 (API level 30) |
| Kotlin | Android | JakartaMail | 2.0.1 |
| Kotlin | Android | okHttp | 4.9.1 |
| Kotlin | Android | retrofit | 2.9.0 |
| Kotlin | Android | volley | 1.2.1 |

# Methodology

In order to verify Universal Acceptance readiness, 5 datasets of sample internationalized domain names & email addresses were used. The next section gives a short description of each one. These datasets are described in detail in [UASG004](https://uasg.tech/wp-content/uploads/documents/UASG004-en-digital.pdf) and [UASG018](https://uasg.tech/wp-content/uploads/documents/UASG018-en-digital.pdf)

For email address internationalization (EAI), a dummy SMTP server, based on the popular [Mailhog SMTP server](https://github.com/mailhog/smtp), was used to verify the support of the SMTPUTF8 SMTP option by the mailer libraries and frameworks. However, Mailhog does not support SMTPUTF8 so we used a [fork](https://github.com/dcormier/smtp/tree/dc/ext) that enhance it to support SMTPUTF8. This dummy server, running within a docker, simulates communication with a real SMTP server from the library/framework perspective and checks if it behaves as expected.

For mobile platform, a fake POP3 from Greenmail was listening on the local network to completely simulate the mail delivering process on the real device.

For library like MessageUI, a crawler testing the emails list has been setup to interact with the interface on the iPhone as a real human would do. The script responsible to launch this crawler had the task to check if emails were sent or not.

## Datasets

### H\_DNS

Performs a syntactic check on a domain name. Determines whether the name appears to be correctly formed. If any part of the name already appears to be in ASCII form (an A-label), verify it can be converted to Unicode. Ref. RFC5891, RFC1035, SAC053

### H\_ES (to check EAI)

Performs a syntactic check on an email address. Determines whether the address appears to be correctly formed. Ref. RFC5891, RFC6531

### H\_ID

Compares the identifier stored in the system against the one used to authenticate by the user. The test cases aim to validate proper handling of internationalized identifiers by applications. Ref. RFC8264

### L\_A2U

Converts a domain name in ASCII to Unicode using the process described in RFC5891. If the domain name, or any constituent label, is already in Unicode or an ASCII label does not begin with the ACE prefix, the original label should not be altered. Ref. RFC5891

### L\_U2A

Converts a domain name in Unicode to ASCII using the process described in RFC5891 for domain name lookup. If the domain name, or any constituent label, is already in ASCII, the ASCII should not be altered. Ref. RFC5891, UTS#46

# Results

Here is the complete list of libraries with their compliance with the corresponding dataset, with a color indicating if they are UA ready or not. Yellow color indicates that some edge cases are not supported or the library needs to be used along with another one to be compliant.

| Legend |  |
| --- | --- |
|  | UA ready |
|  | UA ready but developer needs to be careful |
|  | UA not ready |

## Discussion

Detailed results are available at https://uasg.tech/software.

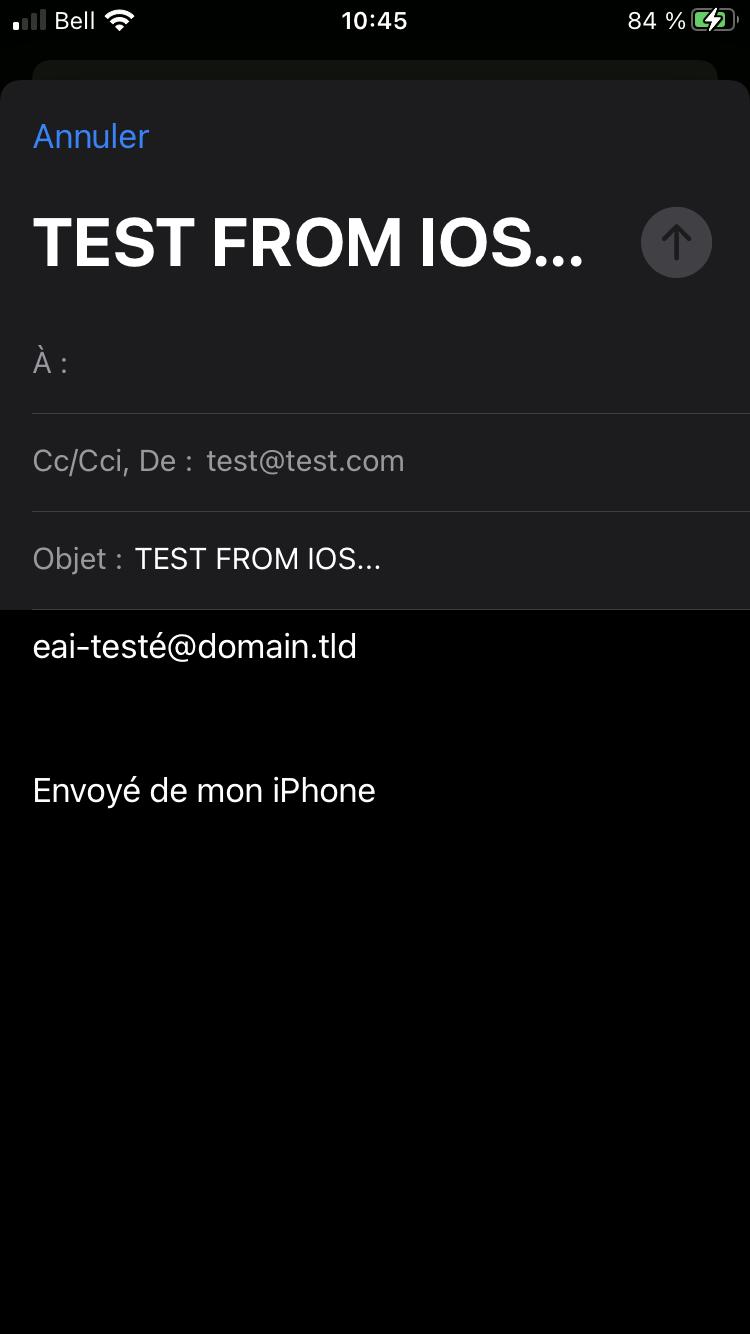
### IOS - Swift - MessageUI (EAI)

|  |
| --- |

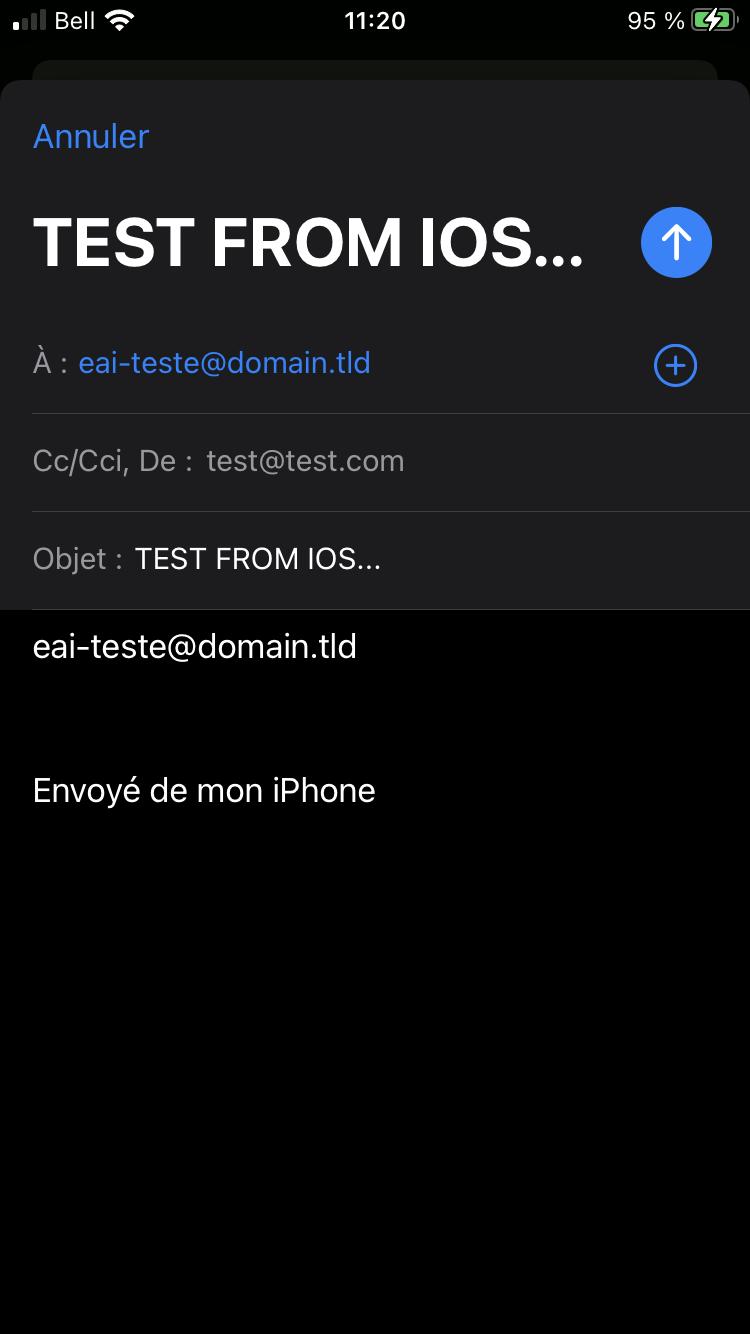
Even if the IOS native Mail app does support EAI, it is the programmatical API offered to developers to prefill an email message and send it that was tested. Unfortunately, it seems that Apple provides only a legacy programmatical API called [MFMailComposeViewController](https://developer.apple.com/documentation/messageui/mfmailcomposeviewcontroller) which tells that:

*Recipient addresses should be specified as per RFC5322.*

in its *setToRecipients* method documentation. This translates to a form that doesn’t display the recipients’ field whenever one inputs an email address including non-ASCII characters (like eai-testé@domain.tld email in the figure below):



The test makes sure that the email address under test was also set in the email’s body which does support non-ASCII characters. As soon as one removes the acute on the “é” and transform it to an ASCII “e” the recipient shows up:



Fortunately, there is a workaround for developers needing to support EAI in IOS. Since the native Mail application does support EAI, one can build a “mailto:” URL that, once clicked, will trigger the Mail app to open:

func sendEmail(subject: String, body: String, to: String) {

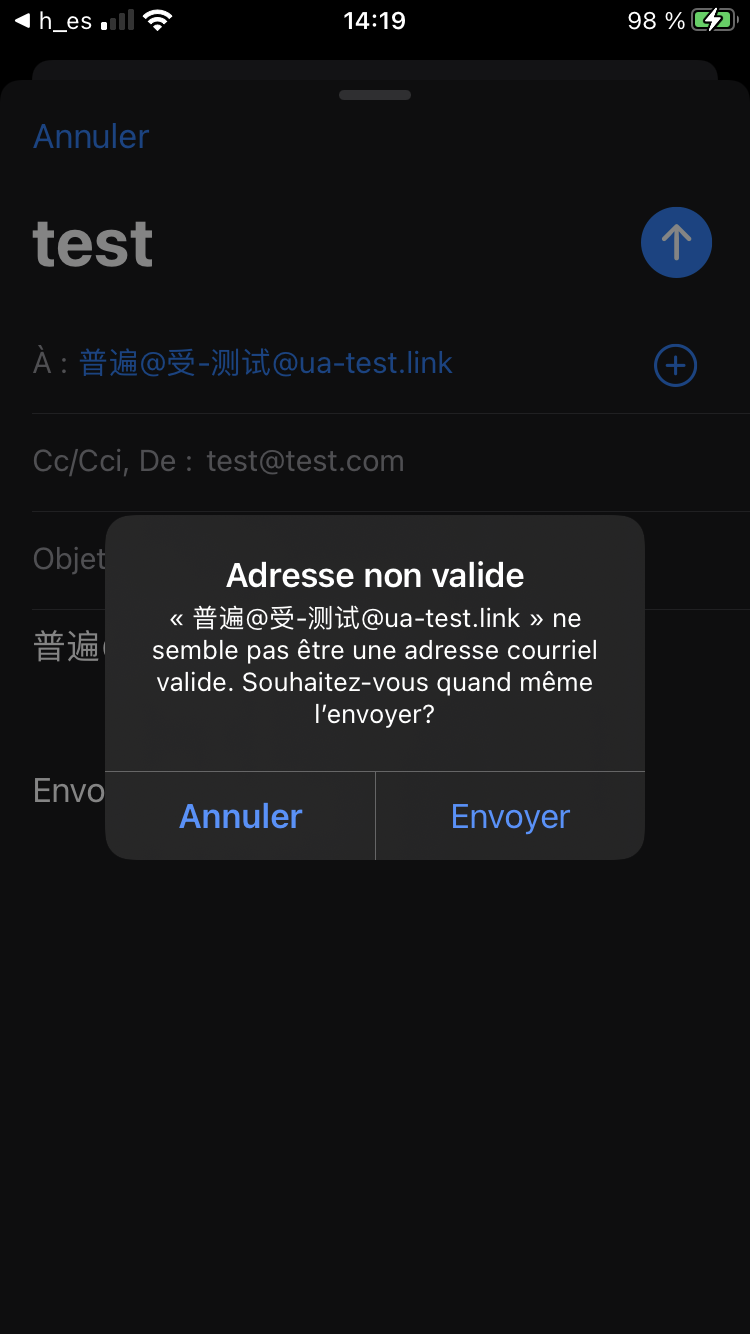
// THIS IS THE EAI WAY: GO THROUGH THE MAIL NATIVE APP:

var url = "mailto:\(to)?subject=test&body=\(to)"

url = url.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed).unsafelyUnwrapped

UIApplication.shared.open(URL(string: url).unsafelyUnwrapped)

This URL method has the consequence to exit the user from the application he/she is currently using instead of opening a modal like MFMailComposeViewController (from MessageUI) inside the current app. This will be a major issue for many app developers. At last, it is worth noting that the app warns the user every time there is a suspicious character in the email (like two “@”):



The Cofomo UA testing team opened a bug in the internal Apple feedback developers form and submit also a corresponding public [StackOverflow](https://stackoverflow.com/questions/69213585/mfmailcomposeviewcontroller-not-displaying-recipients-for-internationalized-emai) about the issue and this workaround.

### IOS - Swift - URLSession (IDNA2008)

|  |
| --- |

URLSession relies on the URL swift object. Unfortunately, this object is known to have issues (see <https://forums.swift.org/t/idn-punycode-in-url/35358> for instance) with what it calls in its internal documentation "illegal characters". Depending on what method the underlying framework uses, URL can unwrap a "nil" string if it considers there are "illegal characters" in it:

/// Initialize with string.

///

/// Returns `nil` if a `URL` cannot be formed with the string /// (for example, if the string contains characters that are /// illegal in a URL, or is an empty string).

public init?(string: String)

Testing raises many:  
  
 Error Domain=kCFErrorDomainCFNetwork Code=-1002 "(null)"}

errors for legal IDNs. The -1002 error code is described as

case cfurlErrorUnsupportedURL = -1002

It seems, like others has experienced, that the URLSession framework unwrap the url with a method non-compliant with IDNA2008 returning "null" or "nil". Despite multiple attempts & implementations, it seems to have no workaround forcing URLSession to unwrap the URL internal string differently.

Nevertheless, the URLSession has no issue with the tested punycode encoded URLs (starting with "xn--"). Thus, one can simply use a [converting library to make it work](https://github.com/Wevah/IDNA-Cocoa): URLSession.shared.dataTask(with: URL(**unicodeString**: "ua-testé.test")!)

The **unicodeString** is an extension constructor parameter added to the URLSession/URL Apple framework by the [IDNA-Cocoa](https://github.com/Wevah/IDNA-Cocoa) library here.

### IOS - Swift - Alamofire (IDNA2008)

|  |
| --- |

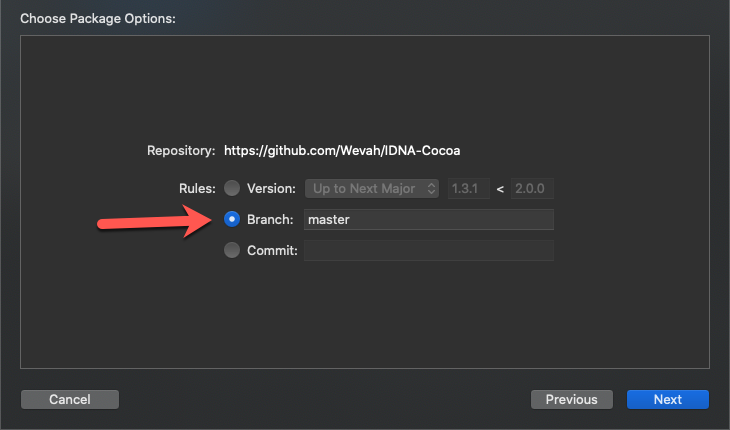
Alamofire is built on top of URLSession & URL objects. Plus, no conversion to an a-label is done before querying an URL containing a u-label. Benchmarks on tested URLs are the same as URLSession. This library must therefore be used with [IDNA-Cocoa](https://github.com/Wevah/IDNA-Cocoa) like URLSession:

AF.request("ua-testé.test".**idnaEncoded**)

Where **idnaEncoded** is an extended property added to the native swift String object by the [IDNA-Cocoa](https://github.com/Wevah/IDNA-Cocoa) library.

### IOS - Swift – IDNA-Cocoa (IDNA2008)

|  |
| --- |
|  |

IDNA-Cocoa seems to be the only Swift package available for IDNA2008. Official releases support only IDNA2003. However, the master branch latest commit supports IDNA2008; developers must thus be careful when importing the package:  


Compliance is good on our dataset except for BIDI rules management & DISALLOWED characters.

### PHP - cURL (IDNA2008)

In PHP, the first library to do an HTTP request one could think of is the cURL native extension.

#### Windows

|  |
| --- |

Developers don’t need to install anything, just enable it in their “php.ini” config file.

The cURL C library must be compiled with the flag --with-libidn2 to use IDNA2008. Unfortunately, the library embedded as a PHP 8.0 extension on Windows comes with libidn only, which supports the old protocol: IDNA2003.

One cannot just replace a “libidn.dll” with a “libidn2.dll” in the PHP lib folder, because everything is compiled and bundled inside a “php\_curl.dll” file.

#### Linux

|  |
| --- |
|  |

The curl extension may need to be installed, some distributions provide a php-curl package. On the distribution used for tests (Arch Linux), curl was included and compiled with libidn2 in the PHP installation.

The PHP curl extension is IDNA 2008 compliant according to our tests, the conversion is performed by libidn2 therefore any bug in libidn2 will impact it.

Some test fails though because curl is too permissives on some invalid URLs: empty label and no label separator.

It is noteworthy that in our tests, we struggled to get PHP to properly provide UTF-8 encoded strings for IDN conversion while all configurations were correct (locales on the Linux host and in php.ini file). The way to solve the problem was to provide the locale programmatically with the following PHP call:

setlocale(LC\_ALL, "en\_US.UTF-8");

### PHP - mail (EAI)

The native [mailing feature of PHP](https://www.php.net/manual/en/function.mail.php) is the first tool put available to developers using its language to send emails.

#### Windows



Unfortunately, PHP mail is not compliant: the SMTPUF8 flag is not sent to the SMTP server and the domain part of the email is not converted to an A-label beforehand.

Interestingly, all UTF-8 emails during testing were not dropped but sent “as-is” nevertheless. An intelligent SMTP server could thus process the emails by “being conservative in what it sends and be liberal in what it accepts from others”.

There is no pull request nor issue registered in the PHP bug report list to update PHP mail to support EAI.

A popular alternative to this native mail feature of PHP is [PHPMailer](https://github.com/PHPMailer/PHPMailer) that does have some issues recorded to support EAI, as confirmed by our results below.

#### Linux

PHP mail on Linux is using sendmail to actually send emails and was therefore not tested. Sendmail version 8.17.1 released the August 2021includes an experimental support for SMTPUTF8.

### PHP - emailValidator (EAI)

EmailValidator is an email address validator including many validation methods, including DNS validation. The validator tested here is called RFCValidator and validates email addresses against several RFCs.

This libraryis used in the Symfony PHP framework to validate email addresses.

#### Windows

Same compliance as Linux.

#### Linux

|  |
| --- |
|  |

EmailValidator is fully compliant with EAI according to our tests. It is highly recommended to use this library for validating email recipients before providing them to another library for email sending. However, this would only validates the email address compliance, whether the email is sent correctly or not will depend on the other library compliance with EAI.

### PHP – Guzzle (IDNA2008)

Guzzle is a popular PHP HTTP client.

It provides 2 handles to make HTTP requests, one with PHP curl and the other with PHP streams. Both methods have been tested and give the same results.

#### Windows

Same compliance as Linux.

#### Linux

|  |
| --- |
|  |

Guzzle is IDNA 2008 compliant, providing that the correct IDN flags are provided.

It relies on the PHP IDN conversion methods provided by the PHP internationalization module, tested below.

The client should be instantiated with the following IDN option:

'idn\_conversion' => IDNA\_DEFAULT

**| IDNA\_USE\_STD3\_RULES**

**| IDNA\_CHECK\_BIDI**

**| IDNA\_CHECK\_CONTEXTJ**

**| IDNA\_NONTRANSITIONAL\_TO\_ASCII**

### PHP – intl (IDNA2008)

PHP intl extension contains the internationalization methods for PHP, including IDN conversion from and to A-label.

#### Windows

Same compliance as Linux.

#### Linux

|  |
| --- |
|  |

Altough some DISALLOWED characters are not detected and would need to be corrected upstream, PHP intl is IDNA 2008 compliant providing that it is used with the right flags:

idn\_to\_utf8($url, IDNA\_DEFAULT

**| IDNA\_USE\_STD3\_RULES**

**| IDNA\_CHECK\_BIDI**

**| IDNA\_CHECK\_CONTEXTJ**

**| IDNA\_NONTRANSITIONAL\_TO\_UNICODE**,

**INTL\_IDNA\_VARIANT\_UTS46**, $idnaInfo);

idn\_to\_ascii ($url, IDNA\_DEFAULT

**| IDNA\_USE\_STD3\_RULES**

**| IDNA\_CHECK\_BIDI**

**| IDNA\_CHECK\_CONTEXTJ**

**| IDNA\_NONTRANSITIONAL\_TO\_ASCII**,

**INTL\_IDNA\_VARIANT\_UTS46**, $idnaInfo);

### PHP – PHPMailer (EAI)

PHPMailer is a popular PHP library for sending emails. It also provides an email address validator.

#### Windows

Same compliance as Linux.

#### Linux

|  |
| --- |
|  |

This library is not UA compliant, its validator even rejects unicode. It supports unicode domain but its conversion is not IDNA 2008 compliant.

It is highly recommended not to use it.

According to a [response from the developer on Stack Overflow](https://stackoverflow.com/questions/56516188/how-to-send-php-mail-with-international-recipient) support for EAI is planned.

### PHP – Symfony (IDNA 2008, EAI)

Symfony is a well-known PHP framework, providing many reusable components, some of which were tested for UA.

#### Windows

Same compliance as Linux.

#### Linux

##### Http-client (IDNA 2008)

|  |  |
| --- | --- |
|  |  |

Symfony HTTP client uses either its own IDN converter or the one provided in PHP intl extension, however it does not provides the flags that would make it IDNA 2008 compliant and its API does not offer a way to provide those flags.

It is recommended to make the conversion with the right flags before providing the URL to Symfony HTTP client.

##### Polyfill-intl-idn (IDN 2008)

|  |  |
| --- | --- |
|  |  |

This is the own version of IDN conversion methods provided by Symfony to replace PHP intl when it is not installed or enabled.

They should be called exactly as the one of intl (with the same flags) and provide the same results.

##### Mailer (EAI)

|  |  |
| --- | --- |
|  |  |

The Symfony component for sending email is EAI compliant but may be too permissive on email address validation as it is using emailValidator PHP library with another validator than RFCValidator and doesn’t allow to change it.

Moreover, domain is converted in A-label and conversion is not IDNA 2008 compliant.

It is thus recommended to make a preliminary email address validation with emailValidator, providing RFCValidator and to convert the domain in A-label with IDNA 2008 before providing it.

### Android - Introduction

Android SDK 30 was used for this testing and the test Android application was developed using Kotlin, Java would have led to equivalent results.

All libraries were used in their last version at the date of testing (september 2021).

### Android - Kotlin - okHttp (IDNA2008)

|  |
| --- |

okHttp is a very popular HTTP client in the Java and Android environments. However, it is only compatible with IDNA2003 as it relies on java.net.IDN. A [bug report](https://github.com/square/okhttp/issues/1615) was closed in 2020 showing they are not willing to support IDNA2008.

Starting from Android 4.4, this library is used by Android to implement java.net.HttpUrlConnection on Android.

Though Android implementation of the java.net.IDN package uses ICU4j that is IDNA 2008 compliant, 2 major issues prevent it to fulfill IDNA 2008 compliance:

* some specific flags have to be used for a fully compliant label conversions and are not set in their package rewriting,
* they are using ICU4j static functions that are only IDNA 2003 compliant.

Therefore, okHttp and all implementations relying on Android network stack would be tied to an IDNA 2003 compliance except if they perform their own IDN conversion.

### Android - Kotlin - HttpUrlConnection (IDNA2008)

|  |
| --- |

HttpUrlConnection is a HTTP client interface in Java, many HTTP libraries are using it to make actual HTTP connections. On Android, starting from Android 4.4, its default implementation is using okHttp (see above). Therefore, HttpUrlConnection is only IDNA 2003 compliant.

### Android - Kotlin - Retrofit (IDNA2008)

|  |
| --- |

Retrofit has the same maintainers as okHttp and is largely using okHttp for its stack, therefore it is IDNA 2003 compliant only.

### Android - Kotlin - fuel (IDNA2008)

|  |
| --- |

Fuel performs percent encoding on URLs but it uses java.net.HttpUrlConnection that correctly performs the conversion in A-label. However, this goes with IDNA2003 compliance only.

### Android - Kotlin - Volley (IDNA2008)

|  |
| --- |

Volley uses java.net.HttpUrlConnection therefore it is only IDNA 2003 compliant.

### Android - Kotlin - Apache HttpClient (IDNA2008)

|  |
| --- |

Android first versions were using a fork of Apache HttpClient then Google abandoned it.

Development still continues but the library is not IDN compliant and should not be used without proper validation and transformation to A-label.

NB: Multiple [Apache HttpClient versions](https://hc.apache.org/httpcomponents-client-4.5.x/android.html) exists. We’ve tested the version from [Android extensions for Apache HttpClient](https://ok2c.github.io/httpclient-android-ext/), targetting the most recent Android versions. That is basically the stock version with utilities for Android.

### Android - Kotlin - Jakarta Mail (EAI)

|  |
| --- |

Although it seems to consider some scripts invalid in domains and fails when the domain is not in normalization form NFC, Jakarta mail correctly validates email addresses and supports EAI.

It is a good solution to directly send email on the Android ecosystem, but expect errors with some domains.

### Android - Kotlin - Email Intent (EAI)

|  |
| --- |

From [Android documentation](https://developer.android.com/guide/components/intents-filters), an Intent is a messaging object you can use to request an action from another app component. Email is one of the common intents provided by Android.

In practicality, an email intent is a way to provide some application pre-filled information for an email to be sent (recipient, subject, body).

Intents are designed to be generic, therefore the email intent is only a generic interface to provide data to another application and it does not perform any validation on email. The data is however correctly transmitted to the email application that would have to perform all the required validation and support EAI.

Thus, it won’t be fair to say email intent is not EAI ready as it is not meant to be EAI ready, but it provides data as-is to other applications therefore, evaluating email intent compliance would mean evaluating all Android email applications.

# Conclusion

Detailed results of the tests are available at: <https://cofomo.github.io/universal-acceptance/report/>

Android

Most tested HTTP libraries are using the same base code therefore the results are quite similar, except for Apache HTTPClient that should not be used.

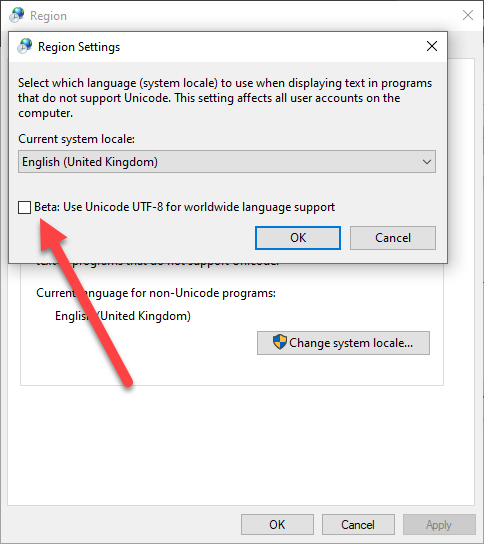
Starting from Android 4.4 okHttp is used by Android to implement java.net.HttpUrlConnection on android but okHttp relies on java.net.IDN which is IDNA2003 compliant only, therefore no library that uses the base Android network stack would be IDNA2008 compliant.

As okHttp seems to refuse to solve the problem, a solution on Android would be to replace java.net.IDN as they replace the java.net.HttpUrlConnection. An IDNA2008 compliant solution could be easily achieved as Android already contains icu.text.IDNA that offers that compliance and contains very similar functions.

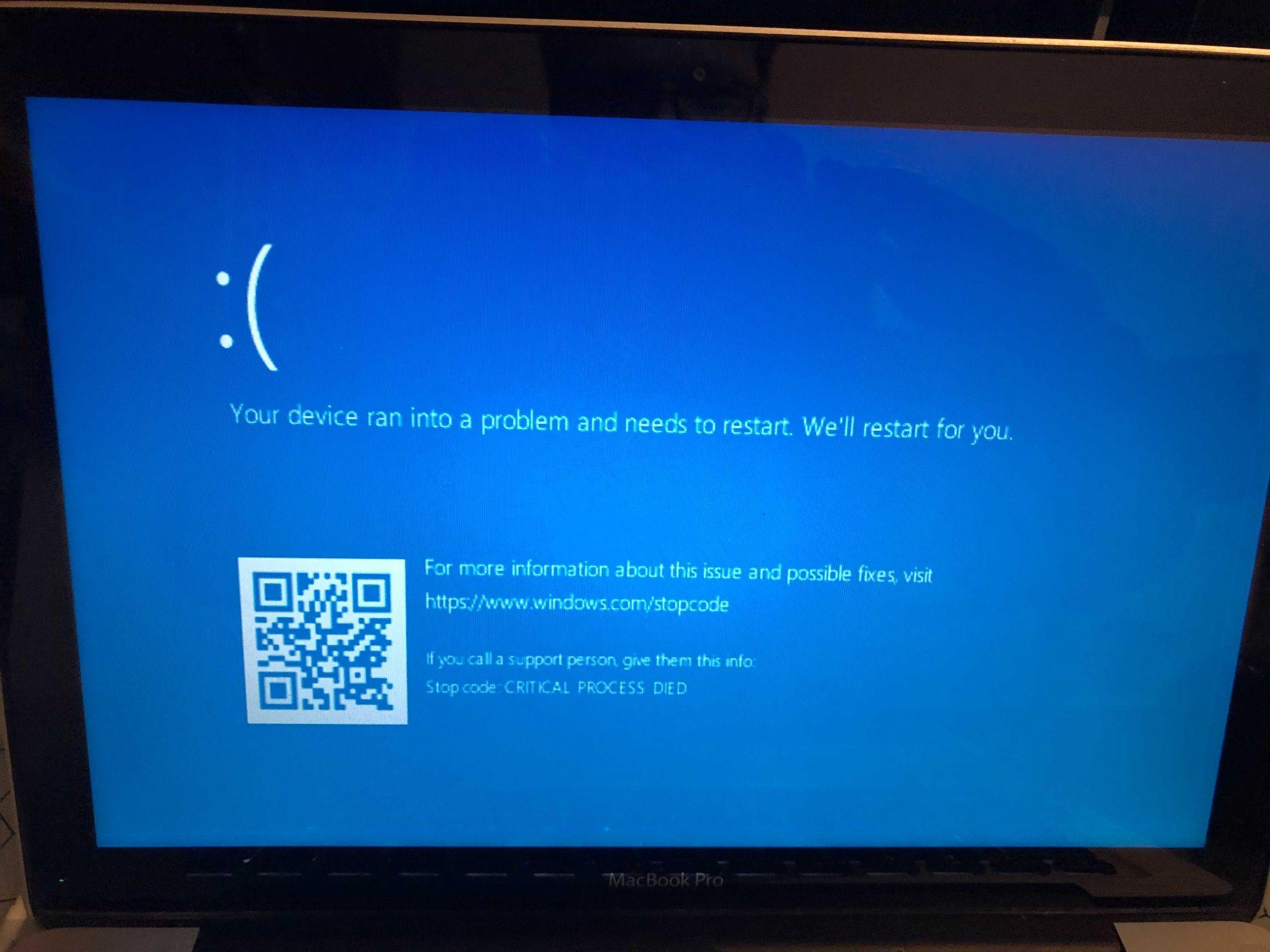
The most used SMTP library on Android offers a good compliance with EAI, however, it often makes more sense for Android developers to use email Intent and then delegate email sending to an application selected by the user. Developers should then be aware that by using email intent they are relying on other applications compliance with EAI.

Windows

There was a surprise during testing on the Windows platform. The shell testing script responsible to pass IDN test cases to the PHP CLI program was not working until this “Beta” Windows feature has been enabled:



By enabling this checkbox, a restart is triggered. This restart has provoked a “blue screen” during testing on a virgin Windows installation, maybe due to the “Beta” nature of the UTF-8 support on Windows (see <https://en.wikipedia.org/wiki/Unicode_in_Microsoft_Windows#UTF-8> for more details):



This checkbox should be checked by any PHP or non PHP developers worried about internationalization & universal acceptance because even if Windows is provided in many languages, these languages are supported through a codepage under the hood that is not UTF-8. This can trigger many problems for all sorts of communications with the outside world (emails, file sharing, etc.) and between programs internally.

IOS

Despite good compliance with EAI and IDNA2008 from native IOS apps like Mail or Safari, Apple doesn't seem to provide libraries with the same level of acceptance for developers. This is rather counter-intuitive since one expects these libraries to power Safari or Mail. Perhaps Apple takes care of always converting to an A-Label before using their HTTP libraries. For email libraries, we found a workaround that goes through their Mail native app, bypassing the standard way of poping an email composition modal (obsolete as we noticed), and everything works fine from there.