Universal Acceptance

UA-Readiness of Some Programming Language Libraries and Frameworks (Phase 3)

25 January 2022

TABLE OF CONTENTS

About Cofomo

Cofomo is an IT consulting firm based in Canada which joined forces with Viagénie, and its experts, in July 2021.

Viagénie has been involved with ICANN on multiple Universal Acceptance (UA) and Internationalized Domain Name (IDN) projects in the past, some of which include the development of the Label Generation Rules Toolset, implementation of a framework for testing languages and libraries against UA test cases, implementation of a test suite for RDAP compliance, and development of the Technical Compliance Monitoring system to monitor the technical infrastructure of generic top-level domain (gTLD) registry and registrars. The staff has been also involved in standards at IETF as co-author of the IDN RFCs (IDNA2003), IDN working group chair, and author of RDAP Bootstrap registry RFC.

Executive Summary

This study is conducted by the Universal Acceptance Steering Group (UASG) covering additional programming language testing beyond what has been published in UASG018A. The following table shows the summary of additional testing, with highlighting in green shows UA-ready, in yellow shows UA-ready but developer needs to be careful, and red shows Not UA-ready libraries across the different platforms.

Language	Lib Name		Platform	Compliance on dataset (%)	Datasets
Swift	MessageUI		IOS	97.8	HEs
Swift	URLSession		IOS	26.8	HDns
Swift	Alamofire		IOS	26.8	HDns
Swift	IDNA-Cocoa		IOS	80.0	LU2A ,LA2U
PHP	cURL*		Windows	92.9	HDns
			Linux	83.9	HDns
PHP	mail		Windows	83.7	HEs
PHP	emailValidator		Windows & Linux	100	HEs
PHP	Guzzle		Windows & Linux	92.9	HDns
PHP	intl		Windows & Linux	95.4	LU2A ,LA2U
PHP	PHPMailer		Windows & Linux	26.1	HEs (validation + sending)
PHP	Symfony	Http-client	Windows & Linux	92.9	HDns
		Polyfill-intl-idn	Windows & Linux	93.8	LU2A ,LA2U
		Mailer	Windows & Linux	33.7	HEs
Kotlin	okHttp		Android	85.7	HDns
Kotlin	HttpUrlConnection		Android	85.7	HDns
Kotlin	Retrofit		Android	85.7	HDns
Kotlin	fuel		Android	85.7	HDns
Kotlin	Volley		Android	85.7	HDns
Kotlin	Apache HttpClient		Android	10.7	HDns
Kotlin	Jakarta Mail		Android	70.7	HEs
Kotlin	Email Intent		Android	91.3	HEs

*PHP cURL compliance for Linux is below the one for Windows even though Linux is compliant because some of the failing tests on Linux are edge cases that do not reflect the overall compliance with UA and Windows has some false positive on invalid labels.

The table below contains a summary the results of the previous programming language testing reported in UASG018A.

Language	Lib Name	Compliance on dataset (%)	Datasets
c	libcurl	84.3	HEs
c	libidn2	95.2	LA2U ,LU2A
csharp	mailkit	84.3	HEs
csharp	microsoft	83.9	LA2U ,LU2A
go	idna	79	LA2U ,LU2A
go	mail	100	HEs
go	smtp	19.6	HEs
java	commons-validator	85.5	HEs ,HDns
java	guava	77.8	HDns
java	icu	93.5	LA2U ,LU2A
java	jakartamail	82.4	HEs
java	jre	71	LA2U ,LU2A
js	idna-uts46	85.5	LA2U ,LU2A
js	nodemailer	84.3	HEs
js	validator	94.2	HEs ,HDns
python3	django_auth	48.1	HEs ,HId
python3	email_validator	86.3	HEs
python3	encodings_idna	67.7	LU2A ,LA2U
python3	idna	100	LA2U ,LU2A
python3	smtplib	84.3	HEs
rust	idna	87.1	LA2U ,LU2A
rust	lettre	7.8	HEs

Introduction

This study is the third in a series that the UASG has conducted on programming languages and libraries; earlier results can be found in UASG018A. The earlier phases focused on 22 libraries and the Linux platform. This phase expands on the previous work by adding mobile platform libraries (Android and iOS) and PHP on the Linux and Windows platforms.

This third phase added 25 libraries-platforms to the set, as detailed in the following table:

Language	Platform	Framework/Library	Tested 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 UA-readiness, five datasets of sample Internationalized Domain Names (IDNs) and email addresses were used. The next section gives a short description of each one. These datasets are described in detail in UASG004 and UASG018.

For Email Address Internationalization (EAI), a dummy SMTP server, based on the Mailhog SMTP server, was used to verify the support of the SMTPUTF8 SMTP option by the mailer libraries and frameworks. However, the released Mailhog does not support SMTPUTF8 so we used a fork that enhances 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 the iOS platform, a fake POP3 from Greenmail was listening on the local network to completely simulate the mail delivering process on the real device.

For a library like MessageUI, a crawler testing the emails list was 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 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

This section lists the libraries and their UA compliance levels, and also shows which dataset was used to test the library. The different colors shows if the library is UA-ready or not. Yellow indicates that some edge cases are not supported or the library needs to be used along with another one to be UA compliant.

Legend	
🟩	UA-ready
🟨	UA-ready but developer needs to be careful
🟥	Not UA-ready

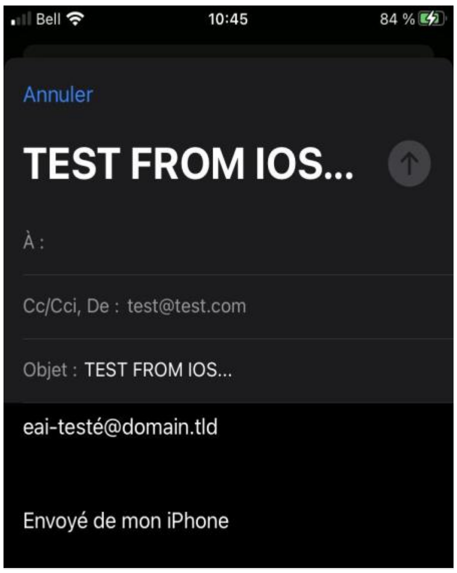
Discussion

Detailed results are available the links provided at the end of this document.

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 that communicates “Recipient addresses should be specified as per RFC5322” in its *setToRecipients* method documentation. This translates to a form that doesn't display the recipient's field whenever one inputs an email address including non-ASCII characters, like eai-testé@domain.tld email in the figure below:



TEST FROM IOS...

À :

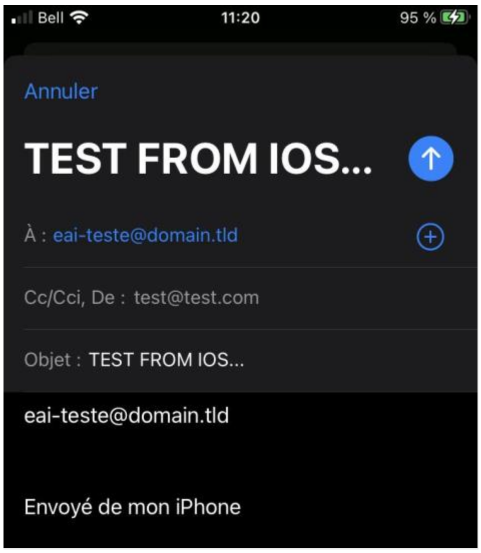
Cc/Cci, De : test@test.com

Objet : TEST FROM IOS...

eai-testé@domain.tld

Envoyé de mon iPhone

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 transforms it to an ASCII “e” the recipient shows up:


Bell 11:20 95 %

Annuler

TEST FROM IOS...

À : eai-teste@domain.tld

Cc/Cci, De : test@test.com

Objet : TEST FROM IOS...

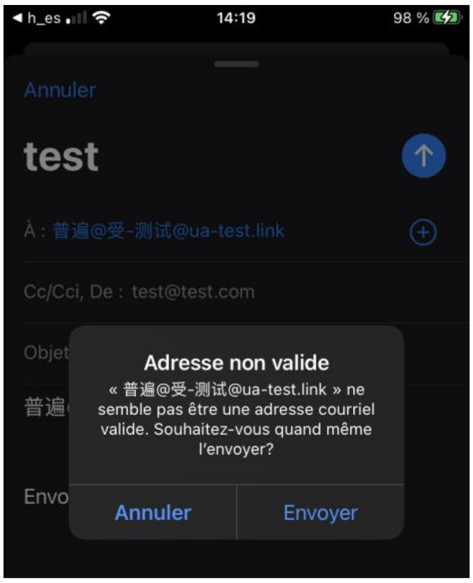
eai-teste@domain.tld

Envoyé de mon iPhone

Fortunately, there is a workaround for developers looking 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 they are 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. Lastly, it is worth noting that the app warns the user every time there is a suspicious character in the email (like two “@”):



◀ h_es 14:19 98 %

Annuler

test

À : 普遍@受-测试@ua-test.link

Cc/Cci, De : test@test.com

Objet

普遍

Envo

Adresse non valide

« 普遍@受-测试@ua-test.link » ne semble pas être une adresse courriel valide. Souhaitez-vous quand même l'envoyer?

Annuler Envoyer

The UA testing team opened a bug in the internal Apple feedback developers form and submitted a corresponding public StackOverflow 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 that 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 errors for legal IDNs like below:

```
Error Domain=kCFErrorDomainCFNetwork Code=-1002 "(null)"}
```

The -1002 error code is described as:

```
case cfurlErrorUnsupportedURL = -1002
```

It seems that the URLSession framework unwraps the URL with a method non-compliant with IDNA2008 returning "null" or "nil". Despite multiple attempts and 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 U-Label-to-A-Label converting library to make it work prior to calling URLSession:

```
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 library.

The bug is reported in

- Stackoverflow
- Swift's issue tracker Jira
- Apple Feedback Assistant (URL not public)

IOS - Swift - Alamofire (IDNA2008)

🟥

Alamofire is built on top of URLSession and 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 like URLSession:

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

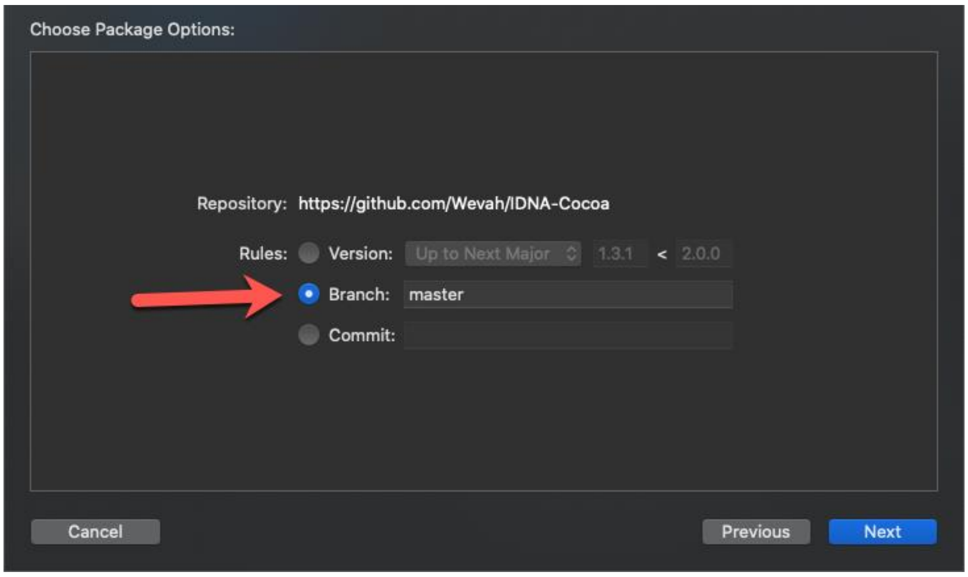
Where **`idnaEncoded`** is an extended property added to the native swift String object by the IDNA-Cocoa library.

Since Alamofire uses the same URL framework as URLSession, we didn't duplicate the bug report.

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 until now (hash= 870ba3ee80ca3555f7ee0da611895311441d10145 of May 22th 2021) supports IDNA2008; developers must be careful when importing the package:


Choose Package Options:

Repository: https://github.com/Wevah/IDNA-Cocoa

Rules: ○ Version: Up to Next Major 1.3.1 < 2.0.0

[✓ Branch: master]

○ Commit:

Using the latest commit, the library had a very good compliance level on our dataset except for BIDI rules management and DISALLOWED characters.

PHP - cURL (IDNA2008)

In PHP, the first library to do an HTTP request one could think of is the cURL native extension. PHP cURL has been tested on Windows and Linux as described below.

Windows

🟥

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

Unfortunately, The cURL C library embedded as a PHP 8.0 extension on Windows uses “Windows.h: IdnToUnicode/IdnToAscii” functions which are known to be only IDNA2003 compliant.

The PHP bug is reported here, but since it is more related to Windows internal libraries, we opened a bug in the Windows’s Feedback Hub as well.

Linux

🟩

The curl extension may need to be installed on some distributions as some provide a php-curl package. On the distribution used for our 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 tests failed however because curl is too permissive 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 is the first tool made available to developers using its language to send emails. PHP mail has been tested on Windows only as described below.

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

An alternative to this native mail feature of PHP is PHPMailer, which does have some issues recorded to support EAI, as confirmed by our results below.

We logged a new bug report here.

Linux

PHP mail on Linux is using sendmail to actually send emails, thus PHP is not involved in the email processing part. It was therefore not tested.

Sendmail version 8.17.1 released in August 2021 includes an experimental support for SMTPUTF8.

PHP - emailValidator (EAI)

EmailValidator is an email address validator that includes many validation methods such as DNS validation. The validator tested here is called RFCValidator and validates email addresses against several RFCs.

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

EmailValidator has been tested on Windows and Linux as described below.

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 sending emails. However, this would only validate the email address compliance, whether the email is sent correctly or not will depend on the EAI compliance of the underlying email sending library.

Windows

Same compliance as Linux.

PHP – Guzzle (IDNA2008)

Guzzle is a PHP HTTP client. It provides two handles to make HTTP requests, one with PHP curl and the other with PHP streams. Both methods were tested and gave the same results. Guzzle has been tested on Windows and Linux as described below.

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
```

Windows

Same compliance as Linux.

PHP – intl (IDNA2008)

PHP intl extension contains the internationalization methods for PHP, including IDN conversion from and to A-label. PHP intl has been tested on Windows and Linux as described below.

Linux

🟩

PHP intl is IDNA 2008 compliant but the compliance is implemented according to UTS #46 in ICU, therefore, some DISALLOWED characters are not detected and would need to be corrected upstream. To get the IDNA 2008 compliance right, flags have to be used:

```
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);
```

A bug report was filled on the PHP bug tracking system but was suspended as it needs to be fixed in the ICU C library and not in PHP source code.

Windows
Same compliance as Linux.

PHP – PHPMailer (EAI)

PHPMailer is a popular PHP library for sending emails. It also provides an email address validator. PHPMailer has been tested on Windows and Linux as described below.

Linux
🟥

This library is not UA compliant; its validator even rejects Unicode. It supports Unicode domains 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, support for EAI is planned.

Some bugs are already reported for RFC6531 support therefore no new bug was filled on that topic. However, a bug on domain part conversion was filled to use the correct IDN flags and we provided a pull request that was merged upstream and released in version 6.5.2 (25 November 2021).

Windows
Same compliance as Linux.

PHP – Symfony (IDNA 2008, EAI)

Symfony is a well-known PHP framework that provides 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 provide 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.

A bug report was submitted along with a pull request that was merged upstream and released in versions 4.4.34, 5.3.11, 5.4.0-RC1 and 6.0.0-RC1.

Polyfill-intl-idn (IDN 2008)

🟩

This is their 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 not EAI compliant. It allows sending email with addresses containing non-ASCII local parts but does not send the SMTPUTF8 flag. It also converts domains in A-label but without full IDNA 2008 compliance.

Finally, it may be too permissive on email address validation as it uses the emailValidator PHP library with another validator other than RFCValidator and doesn't allow it to be changed. It is 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. However, if your email address contains a non-ASCII local-part, the mail server will certainly end the connection with an error.

Three bug reports were filled:

- Email domain conversion to A-label: a fix was provided along with the fix for HTTP client and merged upstream.
- Correctly fail when @ is provided in local-part unquoted
- Implements RFC6531

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 as the libraries and frameworks used for our testing are the same for both languages, Java being able to use Kotlin libraries and conversely.

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 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.

Although Android implementation of the java.net.IDN package uses ICU4j that is IDNA 2008 compliant, two major issues prevent it from fulfilling IDNA2008 compliance:

- Some specific flags have to be used for 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 IDNA2003 compliance except if they perform their own IDN conversion.

We created another bug report for IDNA2008, providing a more appropriate way to solve the issue than the old one. The maintainer was responsive and willing to fix it, at least for Android that provides packages for IDNA directly in the SDK, but the fix broke some of their tests and the fact that IDNA2008 is not implemented in the most used web browser (Chrome) made them stop implementing that support.

It is therefore highly recommended to make Google implement IDNA2008 instead of IDNA2003 in their products if we want to encourage the community to follow.

Android - Kotlin - HttpUrlConnection (IDNA2008)

🟥

HttpUrlConnection is a HTTP client interface in Java and 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 IDNA2003 compliant.

We made a bug report on the Android bug tracker that has been transferred to the engineering team. We also mentioned to the team that other Google products are affected. Resolving those issues at Google is a major step in IDNA2008 adoption.

Android - Kotlin - Retrofit (IDNA2008)

🟥

Retrofit has the same maintainers as okHttp and is largely using okHttp for its stack. Therefore, it is IDNA2003 compliant only.

No bug report has been filled as Retrofit is using okHttp stack and is maintained by the same company.

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.

We submitted a bug report and a pull request to perform conversion to A-label instead of percent encoding on domains.

Android - Kotlin - Volley (IDNA2008)

🟥

Volley uses java.net.HttpUrlConnection, therefore, it is only IDNA2003 compliant.

No bug has been reported as it is developed by Google and is using HttpUrlConnection. The HttpUrlConnection bug report is sufficient to cover Volley compliance.

Android - Kotlin - Apache HttpClient (IDNA2008)

🟥

Android's first versions were using a fork of Apache HttpClient but 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 exists. We've tested the version from Android extensions for Apache HttpClient, targeting the most recent Android versions. That is basically the stock version with utilities for Android.

We filled an issue on their issue tracker.

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.

We submitted a bug report to the Jakarta maintainers.

Android - Kotlin - Email Intent (EAI)

🟨

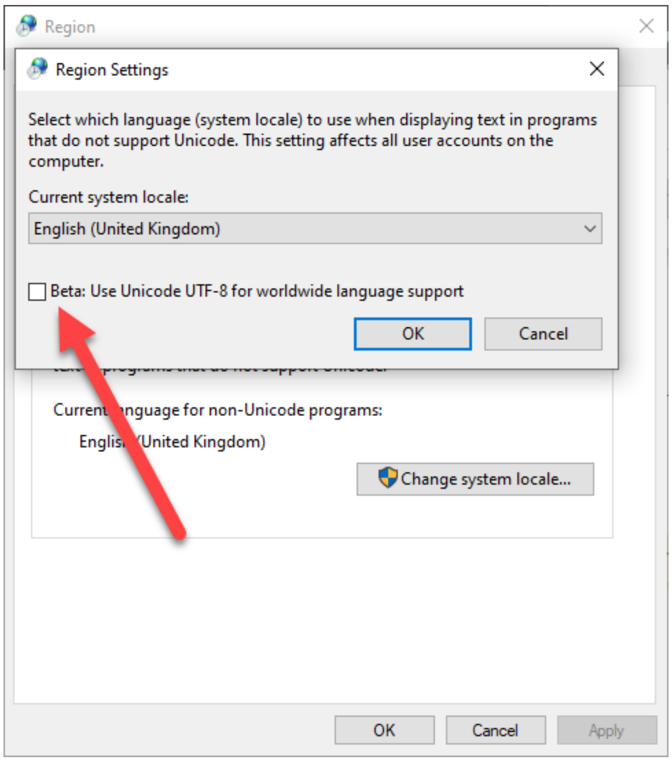
From Android documentation, an Intent is a messaging object you can use to request an action from another app component. Email is one of the common intents provided by Android. In practicality, an email intent is a way to provide some application pre-filled information for an email to be sent (recipient, 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.

As a result, it would not be fair to say email intent is not EAI-ready as it is not meant to be, but it provides data as-is to other applications. Therefore, evaluating email intent compliance would mean evaluating all Android email applications.

A Note on Windows

There was a surprise encountered 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:



Region

Region Settings

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

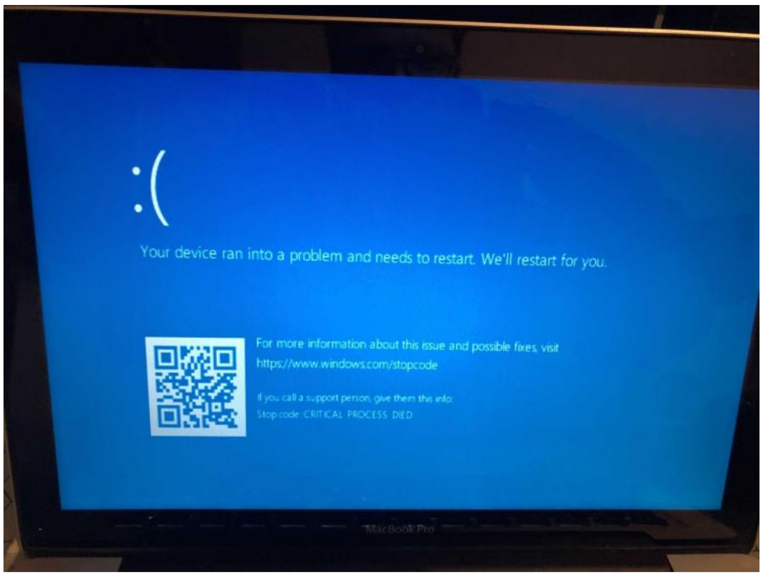
programs that do not support Unicode.

Current language for non-Unicode programs:

English (United Kingdom)

By enabling this checkbox, a restart is triggered. This restart provoked a “blue screen” during testing on a virgin Windows installation, possibly 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.



:(

Your device ran into a problem and needs to restart. We'll restart for you.

For more information about this issue and possible fixes, visit
https://www.windows.com/stopcode

If you call a support person, give them this info:
Stop code: CRITICAL PROCESS DIED

This checkbox should be checked by any PHP or non PHP developers worried about internationalization and UA 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.

Conclusion

Detailed results of the tests are available in the “Test Reports” section of this report.

Android

Most of the tested HTTP libraries are using the same base code, therefore the results are quite similar except for Apache HTTPClient, which should not be used. Starting from Android 4.4, okHttp is used by Android to implement java.net.HttpUrlConnection 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 refuses to solve the problem, a solution on Android would be to replace java.net.IDN in an IDNA2008 compliant way which is not the case. An IDNA2008 compliant solution could be easily achieved as Android already contains and uses icu.text.IDNA that offers that compliance.

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 be aware that by using email Intent, they are relying on other applications’ compliance with EAI.

Windows

As related in the previous phase 2 report, Microsoft supports IDNA2008 in its core .NET Framework. Testing on the PHP cURL extension revealed that this support has not been translated into their C API from "windows.h" and is the cause of the non-compliance reported. Besides that, libraries are behaving as on Linux. Nevertheless, from a UA standpoint, the "Beta" feature using UTF-8 on Windows could lead to subtle errors like the one we encountered for passing non-ASCII parameters between programs.

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 counterintuitive since one expects these libraries to power Safari or Mail. Perhaps Apple takes care of this by 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 popping an email composition modal (obsolete as we noticed), and everything works fine from there.

Bug Reports

As described in the results, bugs have been reported and many got answered. In some cases, we even provided patches by way of git pull requests. At the time of this report being published, three were already merged upstream and one is pending.

We noticed some important elements regarding bug reports:

- The bug report has to be very clear and precise, follow the rules of the project, and suggest appropriate ways to resolve the problem whenever possible.
  - For example, in okHttp, there was already a bug report on IDNA2008 compliance but it was not clear enough and suggested a resolution that was not appropriate, therefore it was immediately rejected.
- Watch and provide answers in the bug report.
  - It is likely that the maintainers will ask questions related to the bug report. If they get no answer then they will discard it.
- If possible, provide new code to solve the issue in the form of a patch in a pull request.
  - In many cases, the maintainers asked for a pull request anyway. Providing a pull request immediately will increase the chances that the problem is solved upstream.
  - The previous elements are also relevant for pull requests.

Test Reports

Kotlin (.html)

PHP Linux (.html)

PHP Windows (.html)

Swift (.html)

Bug Reports

Bug reports for each of the non-compliant tested libraries are available here.