Universal Acceptance

Evaluation of Web Sites for Acceptance of a Variety of Email Addresses

17 August 2017

Report UASG017

Introduction

Universal Acceptance is the concept that all domain names and all email addresses work in all applications.

Universal Acceptance (UA) is required for a truly multilingual Internet, one in which people around the world can navigate entirely in local languages. It is also the key to unlocking the potential of new generic top-level domains (gTLDs) to foster competition, consumer choice and innovation in the domain name industry. This provides consumers with a wider choice of identities to choose when choosing their own domain names. When an online system, such as a website or online form, is UA-ready, it means that it can accept ALL email addresses.

The Universal Acceptance Steering Group (UASG) is a community initiative supported by ICANN and dedicated to advancing awareness and adoption of UA worldwide.

The UASG conducted this first-of-its-kind study of more than 1000 websites to determine if they would accept a variety of email addresses based on new top-level domains (TLDs), including long TLDs and TLDs in non-English characters. The study also evaluated non-English mailbox names.

The results show that there is much work to be done before the world's websites are UA-ready. Longer top-level domains don't do as well as short ones, introducing non-English characters in the domain name markedly reduces the acceptance rate, and introducing non-English characters into the mailbox name further reduces the acceptance rate.

The Evaluation

Building on work started by domain name registry business Donuts, the UASG has, through ICANN's Global Support Center team, evaluated more than 1000 websites (based on Alexa ranking) to see if they allow registration with a variety of email structures:

ascii@ascii.newshort	info1@ua-test.link

ascii@ascii.newlong	info2@ua-test.technology
ascii@idn.ascii	info3@普遍接受-测试.top
ascii@ascii.idn	info4@ua-test.世界
Unicode@ascii.ascii	测试1@ua-test.link
Unicode@idn.idn	测试5@普遍接受-测试.世界
Arabic.arabic@arabic	دون@رسيل.السعودية

For each website tested, a page that allowed registration of an email address was found and attempts were made to register each of the evaluation cases.

Results

1262 websites[1] were considered for evaluation. Out of these, 749 websites included email fields that could be tested. Seven different email addresses were tested.

Fifty-four websites accepted all seven types of email addresses, meaning 7 percent are UA-ready. Forty-seven websites rejected all seven types of email addresses, meaning 6 percent do not accept IDNs or new gTLDs.

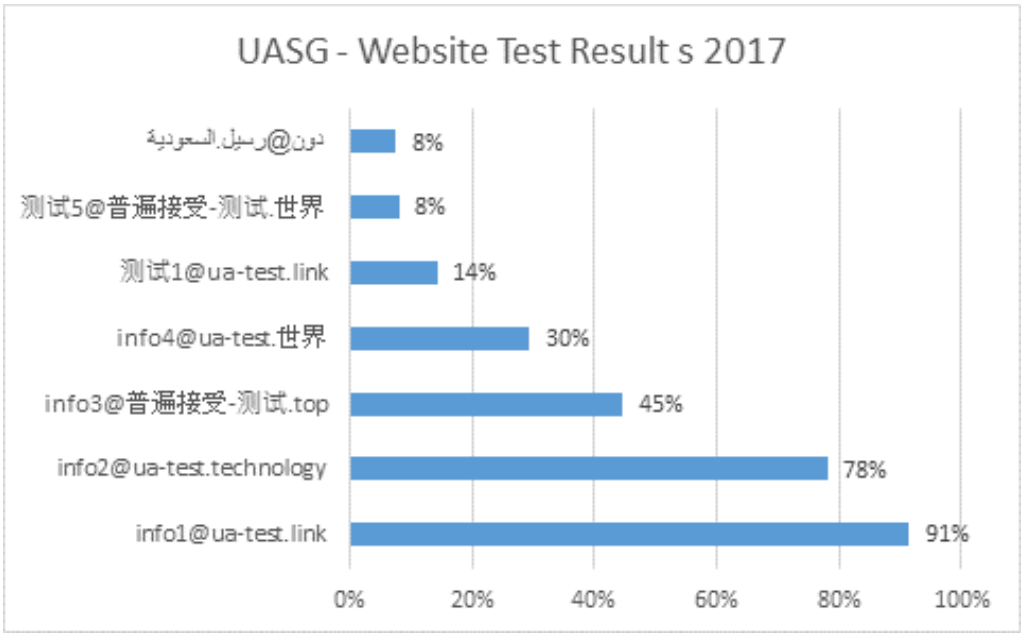
This table demonstrates how many websites accepted each type of email, and the right-most column shows the rate of acceptance out of all websites tested.

EMAILS TESTED		Rate of Acceptance (out of 749 websites)	
ascii@ascii.newshort	info1@ua-test.link	685	91%
ascii@ascii.newlong	info2@ua-test.technology	585	78%
ascii@idn.ascii	info3@普遍接受-测试.top	335	45%
ascii@ascii.idn	info4@ua-test.世界	221	30%
Unicode@ascii.ascii	测试1@ua-test.link	108	14%
Unicode@idn.idn	测试5@普遍接受-测试.世界	61	8%
Arabic.arabic@arabic	دون@رسيل.السعودية	57	8%

[1] Full results are available https://docs.google.com/spreadsheets/d/1T7sbUUBqDTsNWeUrkwXZLcpjlxt8qi-Ty6eH6DrOVdg/edit?usp=sharing

Analysis

Clearly an ascii@ascii.ascii had the highest acceptance rate and Arabic.arabic@arabic (as well as Unicode@idn.idn) the lowest.



When the UASG looked at the source code, they expected common approaches and common code. However, they found that this was not the case when they delved deeper into the code[2]. Very few called server-side libraries for validation. Most used a Regular Expression (RegEx) to provide first line validation. But the UASG did not find a consistent RegEx deployed. Instead, it appears as if developers would fetch a RegEx from GitHub, Stack Overflow or some other source code repository, and then apply their own version.

Conclusion

[2] See Appendix A for a report on the coding behind the web pages

There is much work to be done to get many of the world's websites UA and EAI-ready. Where the UASG thought they could address just a few applications and code repositories, that does not appear to be the case. Instead, the UASG will supplement its library evaluation and mitigation work with greater awareness-raising among the developer community.

Appendix A

Why do some websites reject internationalised email addresses that others accept?

Jim Hague, Sinodun Internet Technologies Ltd.
jim@sinodun.com
1 August 2017

TABLE OF CONTENT ▪

Introduction

Taking the results of the recent UA exercise Evaluation of Websites for Acceptance of a Variety of Email Addresses, we attempted to look a little further at why some websites reject addresses, and why some websites accept addresses others reject. The raw data is available here.[3]

We looked at three categories of websites:

[3] https://docs.google.com/spreadsheets/d/1T7sbUUBqDTsNWeUrkwXZLcpjlxt8qi-Ty6eH6DrOVdg/edit

- A random subset of sites that rejected some email address to see if there was any commonality in the underlying algorithm
- The set of websites that rejected all email addresses to understand the underlying cause
- The set of websites that accepted all email addresses to understand if they performed any validation at all

The following sections present the results for each category. Following those, we present an analysis of the results and suggest some mitigation actions.

Results

Sample validation failures

Here we inspected a random subset of sites to see if we could determine the algorithm responsible for rejecting the addresses. In most cases, we could not, either because the validation was performed on the server, or (in a few cases) simply because the location of validation was obscure and could not be found in a timely fashion. Table 1 below presents a sample of 10 cases where the algorithm could be identified i.e., where some validation was performed in the client which failed.

Table 1 - Sample validation failures

Website	Failing Email	Processing: example code and means of validation
twitter.com	info4@ua-test.世界	email:/^[\w-]+([^@,\s<>()]*[\w-]+)?@[\w-]+(\.[\w-]+)*\.[a-z]{2,}$/i Regular expression check in Javascript.
ibm.com	info4@ua-test.世界	emailFormat: /^(([^\.@"]+(\.[^<>()\[\]\\.*,;:\s@=/&"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/, email: /^[_A-Za-z0-9-!#$%'?^~`\{\}|\+]+(\.[_A-Za-z0-9-!#$%'?^~`\{\}\|\+]+)*@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}]com)\|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/, An email address must pass both of these Javascript regular expressions. The first supposedly checks for a valid email format, the second for invalid chars in the address.
meetup.com	info4@ua-test.世界	isEmail:function(){return this.value.match(/^([a-zA-Z0-9_.-])+@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/)?!1:{key:"isEmail", message:a("register.mobile.emailErrBadEm","Doesn't look like an email address")}}, isNotEmail:function(){return this.value.match(/^([a-zA-Z0-9_.-])+@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/)?{key:"isNotEmail",message:a("validation.error.emailNotAllowed","Can't be an email address")}:!1}, hasBrackets:function(){return this.value.match(/.*?(?:<|>).*/)?{key:"hasBrackets",message:a("validation.error.noBracketsAllowed","Should not have a < or a >")}:!1} Regular expression checks in Javascript.

indiatimes.com	info4@ua-test.世界	var reg = /^([A-Za-z0-9_\-\.])+\@([A-Za-z0-9_\-\.])+\.([A-Za-z]{2,5})$/; Regular expression check in Javascript.
in.bookmyshow.com	info4@ua-test.世界	<input type="text" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$" class="email-input _error" placeholder="Enter your Email ID" id="iUserName" required="" minlength="1"> HTML5 input field with regular expression.
choicehotels.com	info4@ua-test.世界	<input type="email" aria-describedby="membershipEmailError" class="form-control ng-invalid ng-valid-minlength ng-dirty ng-touched ng-valid-email ng-valid-maxlength ng-not-empty ng-valid-required ng-invalid-pattern" id="membershipEmail" name="email" ch-focus-if="missingPartnerHubField === 'email'" ng-class="{'rentals-input text-left text-mondo text-bold': $root.featureFlags.VACATION_RENTALS_NEW_INPUTS}" ng-focus="clearGuestInfoError('email', guestInfoForm.email)" ng-maxlength="60" ng-minlength="5" ng-model="guestInfo.email" ng-pattern="/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$/i" ng-required="true" required="required" aria-invalid="true"> Regular expression validation on a HTML input field using Angular JS.
fodors.com	info4@ua-test.世界	var emailregex = /\b[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}\b/; Regular expression check in Javascript. Checking the Javascript, we found another 2 email validation routines in the Javascript files loaded by the page, each with a different regular expression and/or other processing.
ft.com	info3@普遍接受-测试.top	function(e){return/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/.test(e)} Regular expression check in Javascript.
sears.com	info4@ua-test.世界	reEmail = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i, reEmailUser = /^(root@|abuse@|spam@)/i, Regular expression checks in Javascript.
telegraph.co.uk	info4@ua-test.世界	email:/^[a-zA-Z0-9.!#$%&'*+=?^_`{\|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/i Regular expression check in Javascript.

Rejection of all email addresses

Next we looked specifically at websites that rejected all forms of email addresses used in the tests. There are not many of these (roughly 7 percent of the test samples) and usually the rejection was not, as far as

we could determine, performed in the client. We found three sites which rejected all tested email addresses in the client; these are detailed in Table 2.

Table 2 - Sites which reject all addresses

Website	Processing: example code and means of processing
oomall.com	result=str.match(/^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)((com)|(net)|(cn))+)$/) Regular expression check in Javascript. This rejects all email addresses with non-ASCII domains, and further rejects any TLD that is not .com, .net or .cn. By a narrow margin this site beats some strong competition and wins our personal award for bone-headedness.
cdc.gov	<input autocomplete="on" class="form-control input-xxlarge input-validation-error" data-val="true" data-val-maxlength="Email address must be under 256 characters." data-val-maxlength-max="255" data-val-regex="Email does not appear to be a valid format." data-val-regex-pattern="^[\w-\.]{1,}\@([\da-zA-Z-]{1,}\.){1,}[\da-zA-Z-]{2,3}$" data-val-required="Please enter your email address." id="Email" maxlength="255" name="Email" placeholder="Enter your e-mail address" title="Please enter your e-mail address (required)" type="text" value=""> HTML5 input field with regular expression. This rejects non-ASCII domains, and further rejects any TLD that is not 2 or 3 characters long.
ajc.com	/^[-a-z0-9~!$%^&*_=+}{\'?]+(\.[-a-z0-9~!$%^&*_=+}{\'?]+)*@([a-z0-9_][-a-z0-9_]*(\.[-a-z0-9_]+)*\.(aero|arpa|biz|com|coop|edu|gov|info|int|mil|museum|name|net|org|pro|travel|mobi|[a-z][a-z])|([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(:[0-9]{1,5})?$/i Regular expression check in Javascript. This rejects all non-ASCII addresses, and further rejects any TLD that is more than 2 characters long and which is not in a hardcoded list of TLDs. It also appears that someone has attempted to support IPv4 domain literals, though without the required enclosing [] and allowing a trailing colon and a HTTP-like port number which is not permitted by RFC5321.

Accepting all email addresses

Finally, we looked specifically at websites that accepted all forms of email address used in the tests. Again, there are not many of these (as with sites that rejected all forms of addresses, roughly 7 percent of the test samples), and again there was usually no client validation performed as far as we could determine. Table 3 presents 3 sites that perform client validation and accepted all test addresses.

Table 3 - Sites which accept all addresses

Website	Processing: example code and means of processing

beenverified.com	/^((([a-z]|\d|[!#\$%&'*\+\-\/=\?\^_`{|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+(\.([a-z]|\d|[!#\$%&'\*\+\-\/=\?\^_`{|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+)*)|((\x22)((((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(([\x01-\x08\x0b\x0c\x0e-\x1f\x7f]|\x21|[\x23-\x5b]|[\x5d-\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(\\([\x01-\x09\x0b\x0c\x0d-\x7f]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))))*(((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(\x22)))@((([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))\.)+(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))$/i Regular expression check in Javascript. This appears to accept allowed Unicode characters, though from the base multilingual plane only. This website uses the JQuery validation plugin. This is not an official part of JQuery, but is written by a core JQuery developer. However, the plugin regular expression, which in the original source rejects all non-ASCII addresses, has been replaced by this site with the more flexible regular expression above.
Foxnews.com Account sign in form.	/^.+@(?:[^.]+\.)+(?:[^.]{2,})$ Regular expression check in Javascript. This accepts any Unicode characters, only insisting that the domain must have more than one label and the TLD is 2 characters or longer. This clearly wins our personal award for best use of a regular expression.
intuit.com	/^((([a-z]|\d|[!#\$%&'*\+\-\/=\?\^_`{|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+(\.([a-z]|\d|[!#\$%&'\*\+\-\/=\?\^_`{|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+)*)|((\x22)((((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(([\x01-\x08\x0b\x0c\x0e-\x1f\x7f]|\x21|[\x23-\x5b]|[\x5d-\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(\\([\x01-\x09\x0b\x0c\x0d-\x7f]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))))*(((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(\x22)))@((([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))\.)+(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])){2,}/i Regular expression check in Javascript. This appears to accept allowed Unicode characters, though from the base multilingual plane only, and further insists the TLD must be at least 2 characters long.

Analysis of client validation

Key findings

In all cases the client validation of the email address was done using a regular expression embedded in the client code. The regular expressions typically had the following characteristics:

- In all cases (except those that accepted all test addresses), the regular expression prohibited the use of non-ASCII domain names in email addresses.
- Several regular expressions prohibited TLDs longer than three, four (or, in one case, six) characters.
- Two regular expressions prohibited TLDs not on a hardcoded list (one of these two did allow any two ASCII character TLD).
- In all but two cases (other than those that accepted all test addresses), non-ASCII mailbox identifiers were also prohibited.

Sites that accepted all the test email addresses generally performed only minimal validation via a regular expression (with two of the three specifically accepting Unicode characters from the base multilingual plane), with further validation presumably being done server-side.

Sites that rejected all the test email addresses generally did so because of a combination of a restriction on non-ASCII characters and a restriction on the accepted TLD.

Although the regular expressions used show a certain number of common features, they are all unique. Also, there was absolutely no commonality in the way in which each site used JavaScript to validate email.

None of the sites inspected directly used a library to perform client-side validation.

Implementation details

- In two cases the HTML5 facility for adding a regular expression pattern to an input field was used. In all other cases, JavaScript was used directly for the validation.
- Three attempt to explicitlys match an IPv4 literal address domain (i.e. jim@[192.168.0.1] - nobody attempts IPv6 literals), though one messes it up by not allowing for the enclosing [].

Global vs local sites

Neither the size or technical focus of the company seem to make a difference to the implementation. We find it ironic that the Times of India will not accept a Hindi email address, and neither will IBM (for their global ID sign-up), despite, according to the comments, their code being written by IBM India. And while both of these organisations might be perceived as rather staid and not really in tune with the modern Internet zeitgeist, neither Meetup or Twitter fares any better.

Mitigation actions for client validation

Discussion

From the above (limited) data, it appears that any modifications to enable full UA will have to be per-site; there is no evidence in this sample of any use of common client-side libraries that might be fixed to leverage UA acceptance.

Reviewing recent developer forum and blog posts, it is depressingly clear that the vast majority of developers, when tasked with 'validate this email', ask on a forum, have another user give them a regular expression saying 'I use this' and happily plug that regular expression (or some small variant) in and mark the job done. Comments from the odd enlightened developer pointing out that the regular expression will reject valid email addresses do not seem to create much concern.

Happily there are signs that, among thinking developers, recognition is slowly dawning. **Regular expressions cannot fully validate an email address.**

Partial mitigation

We note that in all the sites checked, the client-side validation is employed as a basic input check. The address is invariably submitted for server-side processing, for example ensuring that no account with that email address has been registered at the site. Any server-side software should be validating all data it is passed from the client.
Practically speaking, therefore, for many systems we think that amending the regular expression to check only that the address contains '@' (or possibly following the Foxnews example above and also checking for a 2-character or more TLD), and thus ensuring that UA email addresses are accepted by the user interface and passed to the server for further validation, could be a first meaningful step to UA.

Full mitigation

Searching the NPM JavaScript package repository for email validation shows isemail to be the most popular package for email address validation by a significant margin. From release 3.0.0 of 22nd June 2017, this supports UA email addresses.

Encouraging its adoption would seem, therefore, to be a promising recommendation to move towards full mitigation for client-side email address validation (note, however, that no formal evaluation of the library has been performed).

Unfortunately, it may well be the case that organisations will be reluctant to deploy more sophisticated client-side checking, as this will increase the amount of Javascript that must be downloaded before a page is ready for input. In this case, partial mitigation may be the only option.

Caveat

Both partial and full mitigation proposals above deal only with client-side validation. As noted, it is to be expected that further server-side validation is also being performed. Judging by the results of *Evaluation of Websites for Acceptance of a Variety of Email Addresses*, it is probable that further mitigation work will be required to ensure that valid UA addresses are not rejected by this server-side validation.