DRAFT

Programming Language Hacks
UA103
Version 17-08-09

Millions of users may be denied access to your services because your application has not kept up with changing standards. That's not fair to you or to them. **Universal Acceptance** is the concept that all domain names and all email addresses work in all applications. Addressing the issue is in the Bug Fix range of efforts.

The characters available in Top Level Domain Names and email address have changed markedly since 2010 when non-ASCII characters were first introduced into the Root Zone. In 2013 the Top Level Domain (TLD) names expanded dramatically as hundreds of new choices became available and the contents of the Root Zone remain very dynamic. In 2012 non-ASCII characters were also available in mailbox portion of email addresses.[1]

Examples include:

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

In a recent study of 1000 popular websites just 7% accepted a full range of email addresses to be used as unique identifiers. When we looked into the code we found no consistency in the approach, in the RegEx's used to validate email addresses, and very little use of server side libraries for validation. There is clearly much to do.

The Universal Acceptance Steering Group (www.uasg.tech) was established in 2015 to raise awareness of these issues and to facilitate resolution. It is an initiative of the Internet community and is supported by ICANN. The UASG has developed a range of documentation, some aimed at management and some aimed at developers. (www.uasg.tech/documents)

It's time for developers to update their code to accommodate these new domain names and email addresses. So here are some tips and tricks to use when updating code:

Input
Data fields that accept domain names or email addresses should be able to accept ASCII and non-ASCII characters. As the next billion Internet users come on line (and existing users use addresses that better reflect their sense of identity) most of them will be from

[1] RFCs 6530 – 6533 cover Email Address Internationalisation

countries that don’t use ASCII and failing to support them will mean missed opportunities. UTF-8 is the key here. This will affect both programs that accept data from a keyboard or other data sources and the database where it’s stored. The good news is that most modern databases will have no problems with this.

Validation

The easiest way to deal with this is to use syntactic validation against the latest specifications in the RFCs[1]. There are other ways of making sure the data entered is what the user meant, such as requiring entry of the field twice and doing a compare.
If you need to validate further, use a DNS lookup – that’s the most certain. Or if you’re going to use a local table of Top Level Domains, make sure that it’s from an authoritative source[2] and that your local table is updated at least daily.

In a recent study there were very few common validation routines in play. It looks like a Regular Expression is fetched from GitHub or StackOverlow and then tweaked. The UASG is developing a recommended validation routine.

Storage

The easiest way to deal with storage is to support Unicode. This ensures that the data is reproducible exactly as received. But for applications or systems that can’t, there is an algorithm (Punycode)[2] that allows transformation of domain names between ASCII and non-ASCII strings. Note: The Punycode algorithm is only intended for domain names. For mailbox names, an alternative encoding schemes may be needed.

Processing

There are times when two different representations (e.g. Unicode and Punycode) of a domain name are not the same but are equivalent. There are some instances where variants within a non-ASCII script may exist or when a label uses multiple scripts. When processing or sorting, it’s important that equivalent names are treated as equivalent. This will require some policies for the application or indeed the organization as to how domain names and email addresses are dealt with.

Display

Because domain names in non-ASCII characters (and mailbox names too) are growing more popular, you’ll need to make sure that you’re able to display them in a way that works for your community. Public facing applications should certainly display in native scripts and appropriate fonts and not in ASCII resulting from a Punycode transformation (e.g. A-label).

Check Libraries

A growing number of libraries, particularly Open Source Programming Language Libraries, will be creating or correcting validation routines, so being able to be UA Ready may be as simple as re-compiling the code using the latest version of the library. The UASG is encouraging remediation work in many libraries.

[2] There are a few options for the authoritative list of TLDs. The first option is the DNS root zone itself. It is DNSSEC-signed, so the list is properly authenticated. You can obtain the root zone from any of the following links:

- http://www.internic.net/domain/root.zone
- http://www.dns.icann.org/services/authoritative-dns/index.html
- http://data.iana.org/TLD/tlds-alpha-by-domain.txt

Github and SourceForge are also two good places to look to find working code.
The UASG also publishes some good reference material at www.uasg.tech/documents.

Don't forget Log Files

Application Log Files are important when solving problems and developers should not forget to ensure that these too are UA ready.

Most efforts to get applications UA Ready will fall into the 'Bug Fix' level of effort. It's time to get applications up to scratch.