UASG

Criteria for evaluating programming languages and frameworks with regard to their UA compliance

Version 0.90
October 24th, 2016

Universal Acceptance

Background – Universal Acceptance

Universal Acceptance is a foundational requirement for a truly multilingual Internet, one in which users 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. To achieve Universal Acceptance, Internet applications and systems must treat all TLDs in a consistent manner, including new gTLDs and internationalized TLDs. Specifically, they must accept, validate, store, process and display all domain names.

The Universal Acceptance Steering Group is a community-based team working to share this vision for the Internet of the future with those who construct this space: coders. The group's primary objective is to help software developers and website owners understand how to update their systems to keep pace with an evolving domain name system (DNS).

Goal

The following criteria are to be used to evaluate whether a programming language or framework supports Universal Acceptance of all domain names and email addresses.

Glossary and abbreviations

PL – programming language or framework

Method – an explicitly or implicitly called function, method, subroutine or predicate (depending on the paradigm) used to perform operations on a data set, such as a domain name in a string

Data set – a domain name or email address

Core of (programming) language – the extent to which programming entities such as functions and methods are included in the scope of the default install of that language. Standard libraries are part of the core, non-standard or third party libraries or extensions are not part of the core.

Native – native in this context pertains to the distance of the device from the core of the language

Predefined – anything that is for the most part invariable, though maybe not immutable. Methods compiled into the interpreter (for script languages) are predefined, though they may be overridden.

Defined – anything variable, such as a device provided by a piece of code as part of a library to the language

Programming language / framework evaluation

An evaluation template can be found in the appendix of this document.

1. Initial observations & evaluation details

The following data is collected in free flow text in order to gain a general understanding of the language and systems used to evaluate.

- Name of programming language, version evaluated, Operating system used for Evaluation
- Which era is the programming language from?
- Has network enabled communication been a focus of the programming language?
- What are the primary domains of the programming language (web/online/offline etc), derived from its real-world use?

-

2. Scored evaluation criteria

The criteria in this section are scored according to the points listed with each criteria in the evaluation template in the appendix. A criteria that is not matched scores 0.

After judging all criteria questions, add up the results to an evaluation score for the programming language/framework. Please note details/background information in your report beyond the score.

In order to evaluate the criteria, the examples from US004 can be used.

2.1 a) Is TLD identification supported within the PL?

- TLD identification is the distinct and exact separation and lookup of the most significant part of the domain name.
- Splitting a string on a dot boundary is not identification, but segmentation. If a lookup is missing, this criteria is not fulfilled.
- Code splitting a string on a dot boundary and performing a lookup search on a table of some kind fulfills this criteria.

2.1 b) Is TLD identification performed using a predefined list or a defined list? (if 2.1a applies)

- Predefined lists could be compiled into the PL, or otherwise be constant. TLDs are dynamic, so predefined lists make this a mismatch.
- Defined lists are able to be updated, for example during time of operation. Defined lists make this a match.
- A DNS validation makes this valid as well, if the environment is intended for online applications only.

2.1 c) Are defined lists created from official feeds? (2.1a applies)

- Secondary providers (without integration into the ICANN processes) make this a mismatch.
- Consequently, primary providers (ICANN/IANA list or DNS validation) make this a match.

2.2 Does the PL offer native methods to process a TLD, a domain name, email address or URL?

- Implementations of such methods within the translation units of the interpreter (if present) are native by default.
- Implementations of such methods in the programming language itself are to be considered native unless they're not bundled with the programming languages as a run time environment.
- **Hint**: If a regular download of the programming language project's distribution files comes with a "standard library" containing such methods for use during run time, this criteria matches.
- **Hint**: Score this item 4 times, one time for TLD, domain name, email or URL each

2.3 Does the PL offer methods to inspect a TLD, a domain name, email address or URL in a structured manner?

- Is the user able to get structured bits of information from a data set?
- TLD, Domain name, email address or URL segmentation, such as splitting host and local part.
- Perhaps identifying a TLD, or retrieving the second level domain name.
- **Hint**: This is not about validating the data set, but about accessing parts of some meaning.

2.4 Do the PLs methods offer the ability to assess the validity of a data set in a structured manner?

- Can the user receive detailed information about the validation result?
- If the methods return structured information indicating states such as 'error in local part exists', or 'address not RFC compliant', this criteria matches.
- The PL reporting errors in the encoding of the data set DOES NOT match this criteria. Encoding errors are unstructured, hence they are handled elsewhere.

2.5 Do the methods handling TLD, a domain name, email address or URL accept input in Unicode?

- For this criteria, only the common UTF variants count (UTF-8, UTF16, UTF-32).
- **Hint**: Accepting either one of the UTFs will result in a criteria match.

2.6 a) Do the methods handling TLDs, domain names, email addresses or URLs allow for input data in Punycode?

- Punycode is detailed in RFC3492. It converts Unicode to ASCII for transport.
- If there are no such methods allowing for Punycode data, the score is 0.
- If there are methods accepting any number of possible encodings, Punycode must be one of these encodings.
- If there are versions of methods dedicated to only work with Punycode (such as a fictitious method "validate_domainname_Punycode" as a sibling to the equally fictitious standard "validate_domainname"), requirements are met.

2.6 b) Are methods handling TLDs, domain names, email addresses or URLs able to produce output in Punycode?

- Punycode is detailed in RFC3492. It converts Unicode to ASCII for transport.
- In some instances, it is desirable to generate Punycode from another representation.
- If Punycode cannot be generated on request, the score is 0.
- If Punycode can be generated on request, the criteria is fulfilled.

2.6 c) Can Punycode validation be performed during handling of Punycode encoded data?

- Punycode is detailed in RFC3492. It converts Unicode to ASCII for transport.
- Punycode allows for validation of Punycode encoded strings. Faulty Punycode encoded input must be identified. NOTE: This part concerns Punycode encoding, not the resulting Unicode.
- If it is not possible to detect faulty Punycode encoded input, the score is 0.
- If it is possible to detect faulty Punycode encoded input, the criteria is fulfilled.

2.7 Do the methods handling Punycode identify incompatible Unicode sequences?

- Punycode is not concerned with 'compliant' Unicode encodings.
- Punycode to Unicode conversion must thus be integrated into the methods.
- Rejection or dismissal of incompatible Unicode sequences make this a match.
- Anything else makes this a criteria mismatch.

2.8 Does the PL store domain names and email addresses in an endorsed Unicode representation format in memory during run time?

- Must be either UTF-8/16/32

2.9 Does the PL store domain names and email addresses in an endorsed Unicode representation format in persistent storage?

- Must be either UTF-8/16/32

2.10 Does the PL implement handling of TLDs, domain names, email addresses and URLs according to specifications and best practice?

- Segments have maximum lengths, but Punycode may exceed them (up to 255 characters due to DNS limit).
- TLDs can have more then three letters, and be internationalized

2.11 When processing domain names or email addresses for display, is the string representation of the data converted from an internal format to the display format?

2.12 Does the PL account for faulty Unicode encodings stemming from a Punycode conversion device?

- Invalid Unicode strings must not be processed further, especially in operations where several conversions happen sequentially.

3. Informational evaluation criteria

What is the internal storage format for TLDs, domain names, email addresses and URLs when used in string form?

- Please describe the internal storage format
- Name the of the common encoding used.
- The storage format is most likely a UTF type store.

4. Technical notes

Please note any uniqueness of the PL when it comes to working with TLDs, domains, email addresses or URLs.

5. Tips for implementing UA in PL

5.1 Native functions to use

Are there any built in functions available that fit the Universal Acceptance criteria and enable the PL to support these?

5.2 Recommended libraries

Are there any libraries available that fit the Universal Acceptance criteria and enable the PL to support these?

5.3 Storage functions to use

Which ways of storing TLDs, domain names, email addresses and URLs, do you recommend.

5.4 Conversion functions

Which conversion functions are readily available?

5.5 Proposed changes for better UA support

Please list potential enhancements that could be made to the PL and/or libraries for better Universal Acceptance support.

Appendix: Programming language / framework evaluation template

1. Initial observations & evaluation details	
Name of PL	
Version evaluated	
Operating system used for evaluation	
Which era is the PL from?	
Has network enabled communication been a focus of the PL?	
What are the primary domains of the PL (web/online/offline etc), derived from its real-world use?	

2. Scored evaluation criteria

The criteria in this section are scored according to the points listed with each criteria. A criteria that is not matched scores 0.

After judging all criteria questions, add up the results to an evaluation score for the programming language/framework. Please note details/background information in your report beyond the score.

#	**Topic**	**Evaluations**	**Score**
2.1a	TLD identification	TLD identification provided = 1 no TLD identification provided = 0	
2.1b		(only if 2.1a applies) TLD identification through a list (w/ updates) = 1 TLD identification through DNS = 2	
2.1c		(only if 2.1b applies) Official feeds used for lists = 2 Other unofficial sources = 0	
2.2	Native methods (process)		
2.2a	TLD	Native method for processing TLDs = 1 No native method for processing TLDs = 0	
2.2b	Domain name	Native method for processing domains = 1 No native method for processing domains = 0	
2.2c	Email address	Native method for processing email addresses = 1 No native method for processing email addresses = 0	
2.2d	URL	Native method for processing URLs = 1 No native method for processing URLs= 0	
2.3	Native methods (inspect)		

Universal Acceptance

2.3a	TLD	Native method for inspecting TLDs = 1 No native method for inspecting TLDs = 0	
2.3b	Domain name	Native method for inspecting domains = 1 No native method for inspecting domains = 0	
2.3c	Email address	Native method for inspecting email addresses = 1 No native method for inspecting email addresses = 0	
2.3d	URL	Native method for inspecting URLs = 1 No native method for inspecting URLs= 0	
2.4	Native methods (validate)		
2.4a	TLD	Native method for validating TLDs = 1 No native method for validating TLDs = 0	
2.4b	Domain name	Native method for validating domains = 1 No native method for validating domains = 0	
2.4c	Email address	Native method for validating email addresses = 1 No native method for validating email addresses = 0	
2.4d	URL	Native method for validating URLs = 1 No native method for validating URLs= 0	
2.5	Unicode input	Unicode (UTF8/UTF16/UTF32) is accepted =1 Unicode is not accepted = 0	
2.6a	Punycode input and conversion	Punycode is accepted = 1 Punycode is not accepted = 0 PL has facility for Punycode → Unicode conversion = 2	
2.6b	Punycode conversion and output	Punycode cannot be generated on request = 0 Punycode can be generated on request = 1	
2.6c	Punycode fault detection	Not possible to detect faulty Punycode encoded input = 0 Possible to detect faulty Punycode encoded input = 1	
2.7	Punycode in Unicode (identification of incompatible sequences)	Validation of Punycode input encoding = 1 No validation of Punycode input encoding = 0	
2.8	Unicode storage at runtime	UTF8/16/32 used = 1 UTF8/16/32 not used = 1	

2.9	Unicode (persistent storage)	UTF8/16/32 used = 1 UTF8/16/32 not used = 0	
2.10	Best practices	Validation of segments with proper (Punycode!) max length = 1 TLD can be longer than three characters = 1	
2.11	String/Display data conversion?	Conversion between internal and display format = 1 else = 0	
2.12	Faulty Unicode encodings	Punycode → faulty Unicode conversions caught = 1 Punycode → faulty Unicode conversions not caught = 0	
Total Score			

3. Informational Evaluation Criteria	
Describe internal storage format	
Encoding(s) used	

4. Technical Notes	
Please note any uniqueness of the PL when it comes to working with TLDs, domains, email addresses or URLs.	

5. **Tips for implementing UA in PL**	
5.1 Native functions to use	
5.2 Recommended libraries	
5.3 Storage functions to use	
5.4 Conversion functions	
5.5 Proposed changes for better UA support	