Universal Acceptance

Reviewing programming languages and frameworks for compliance with Universal Acceptance good practice

21 August 2017
Version 1.0

Report UASG Programming Language Framework Review

TABLE OF CONTENTS

Document History

Version	Date	Authors	Notes
1.0	21 Aug 2017	Jim Hague Sara Dickinson	Review call 2017-08-09 updates.
0.99	4 Aug 2017	Jim Hague Sara Dickinson	Address comments on previous versions. Prepping for review call.
0.98	11 Jul 2017	Jim Hague Sara Dickinson	Add an Appendix in a separate document to describe the test sets in detail and link to documents containing example test data. Also some minor updates to the main document.
0.97	11 Apr 2017	Jim Hague Sara Dickinson John Dickinson	Incorporate received comments and corrections, review
0.96	10 Mar 2017	Jim Hague Sara Dickinson	Revised to follow test suite based approach
0.95	19 Jan 2017	Frank Michlick	Updated with feedback after first team review
0.90	24 Oct 2016	Frank Michlick	First version

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 (UASG) 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).

About this document

This document was created to provide a framework for the evaluation of popular programming packages and libraries and their usefulness in aiding Universal Acceptance good practice. It is a response to the description of work issued by UASG, available at https://uasg.tech/wp-content/uploads/2016/05/Help-Wanted-Open-Source-Software-Review-v201602111.pdf.

Where those packages or libraries do not provide the expected support, the follow-up project would create recommendations/patches to be submitted to add UA support and guidance for application developers on correct and effective use of the packages or libraries.

This document and the evaluation process it describes are expected to evolve together as experience is gained with evaluations.

Technical details required by those performing library evaluations, while forming Appendix C of this document, are presented in a separate Google Doc document. This separation of documents is purely due to technical restrictions in the document platform.

Target audience

The main part of this document serves as an overview of the evaluation framework presented in a form appropriate for all stakeholders.

Those performing the library evaluations require the technical details of the tests and the test data sets presented in Appendix C and accompanying documents. The test descriptions in this main document describe the categories of tests, omitting technical detail, and are accompanied by illustrative individual tests using data drawn from UASG004 - Use Cases for UA Readiness Evaluation.

Background

Software applications that make use of Internet services are built and used in a variety of ways. They exist at all points along a continuum ranging from embedded firmware in a connected device, through desktop/mobile/tablet applications, through to software that runs purely in a web browser environment, the latter often communicating with more software running on remote servers.

All these types make use of Internet identifiers which, while historically represented only in characters employed by US English (i.e. A-Z, 0-9 and '-'), can now, via the IDNA Protocol, be fully multilingual. These identifiers are:

- **Domain names, e.g. example.com or 普遍接受-测试.世界**
- **Email addresses, e.g. joe.bloggs@example.com or 测试3@普遍接受-测试.top**
- **Web addresses, or more precisely Uniform Resource Locators (URLs), e.g. https://uasg.tech/uasg-charter/ or https://普遍接受-测试.top/我的页面**

It is therefore important for all stakeholders in development of a software application to be aware what libraries are available for their chosen development environment to be used for processing Internet identifiers, and to have a clear basis for assessing those libraries, for technical and business suitability, with regard to the UA correctness and compliance.

Terminology

Libraries: All but the most unusual Internet applications today rely heavily on software components to perform much of their function. These components are variously termed packages, frameworks or libraries (which may or may not include various bindings); for brevity, they will all be referred to as libraries henceforth.

Functions: Similarly, the services offered by these libraries may be variously classed as methods, functions, APIs etc. but will be referred to simply as **functions** henceforth.

URLs: **URL** will be used henceforth as a generic term for both URLs and IRIs.

Identifier: Any of the Internet identifiers: domain names, email address or URLs.

References

A full list of references is given in Appendix D.

The illustrative test data presented in the main document is drawn from UASG004 - Use Cases for UA Readiness Evaluation. Other references list the relevant standards and related information in the following categories:

- **IDNA RFCs**
- **Unicode**
- **IANA Registries**
- **Special-use domain name RFCs**
- **Internationalized email RFCs**
- **Internationalized resource identifier RFCs**
- **Obsolete IDNA RFCs**

Initial list of libraries

This section lists a currently popular set of libraries that are good candidates for an initial set of evaluations.

- GNU Libidn. **Implementation of IDNA2003 in C. Bindings available for Perl and Ruby.**
- GNU Libidn2. **Implementation of IDNA2008 in C by the author of GNU Libidn.**
- International Components for Unicode. **Versions are available for Java and for C with C and C++ bindings.**
- Python encodings.idna. **Part of the Python standard library. Test in Python and Python3.**
- Python idna module. **A replacement for the Python standard library encodings.idna module that supports IDNA2008. Test in Python and Python3.**
- PHP IDN functions. **Part of the PHP standard library, supporting IDNA2003 and IDNA2008.**
- Go idna package. **Part of the Go standard library supporting IDNA2008.**
- Javascript idna-uts46 npm module. **Supports IDNA2003 and IDNA2008. Bundled with Node.js.**

The evaluation process will initially consider only Open Source libraries.

This evaluation process will consider libraries that offer IDNA-like API functions. However future consideration may also be given to evaluating libraries that implicitly require support for UA to function, e.g. TLS libraries that have a native implementation of domain name processing in order to validate X.509 certificates.

Basis of library evaluation

Measurement

The primary technical evaluation of all functions must be done by implementing and running a test suite consisting of distinct *test cases*. This evaluation is augmented with additional (rather more subjective) evaluations of other aspects of the library itself, which are *evaluation criteria*. The details of the evaluation are presented in the Library evaluation section.

Each test case or evaluation criteria is to be scored on the basis of a simple 0 to 5 point system:

0	1	2	3	4	5
Not Supported	Very Weak	Weak	Acceptable	Strong	Very Strong

These values are then summed (with a reduced 50% weighting for the evaluation criteria) to produce an overall score. An example scoring template is provided in Appendix A - Scoring template.

Test suite

This document outlines the test cases to be used in the test suite.
The scope of this document is to specify tests that provide specific evaluation criteria as a starting point for an evaluation of the overall library quality for typical use cases. A test suite providing comprehensive test coverage for all functions is large task, and is outside the scope of the current statement of work.

As noted, many libraries are available with bindings enabling them to be used by languages other than their implementing language. In this case, the test suite must be based on using the binding language.

It is strongly recommended that test suites generated as a result of these evaluations should be published under an Open Source license, to aid the growth over time of a comprehensive test resource.

To aid in evaluation, we classify library functions into two groups:

Low level functions: those that provide basic lower-level services, typically transformations defined in the IDNA RFCs.
High level functions: those directed at higher-level application tasks such as syntactic and semantic checks. These will typically include calls to lower-level functions to perform their tasks.

This classification allows the assessor and potential library user to judge whether functionality provided by a particular library is likely to be sufficient in and of itself, or whether further application code or other libraries will be necessary. It is highly preferable that higher level functions are implemented by libraries to avoid multiple application developers having to separately reproduce this functionality which would likely result in errors and inconsistencies.

Assumptions

We also make some basic assumptions about the programming languages being employed.

- **Unicode strings may be represented and manipulated by commonly available facilities. While this is universally true for all popular contemporary languages, the internal encoding a language will use to represent Unicode varies between languages; UTF-8 and UTF-16 are popular choices. When a Unicode string is required for a test, it is assumed it will be represented in the encoding native for the language.**

- **The language may make use of any available operating system services, including the network stack.**

These are both reasonable assumptions for all modern programming environments, and also serve to prevent the scope of this document expanding unreasonably.

In other words, this framework is not designed for evaluating heavily restricted or specialised languages and runtimes.

Library evaluation

A library evaluation is a report with 5 sections:

1. Basic information
2. Auxiliary information
3. Implementation notes
4. Technical evaluation
   a. Test suite (based on behaviour described in RFCs)
   b. Report on 'Semantic Checks' functionality
5. Mitigation Actions

These sections are as follows.

1. Basic information

Basic information on the library. This should include:

Library name and version
Provider & how obtained
Test suite language, list of library implementation languages
Test environment:
    1. Processor architecture
    2. Operating system
List of notable library dependencies. Include here dependent libraries that do not form part of a standard operating system install.
What is the license?
Is there any financial cost to using the library? State items specified to obtain this figure such as country of use, hardware configuration etc.
Evaluator information. I.e. identity of evaluator, supporting documents and tests.
Any other comments, e.g. comments from the library maintainer related to the evaluation

SCORING: This section is not scored

2. Auxiliary information

Assess each criteria and include comments on:

1. Public bug tracker - is one available, are issues acknowledged and fixed?
2. Maintenance activity - does the library appear to be actively developed, what is the frequency of releases (at least 1 release in the last 12 months or clear indication that the project is still active even if the code has not needed an update)?

3. Documentation - what is the quality of the User guide, API reference, etc.
4. Availability of source code - is it publicly available, can patches/pull requests be easily submitted?
5. Other support channels - are there users mailing lists, forums, Stackoverflow articles, etc.?
6. Maturity and adoption - is there a large user base, is the library well established?
7. Standards support - does the library claim to support to most recent version of the standards (will aid with identifying libraries that only support IDNA2003)?
8. Portability - to what extent is the library available on multiple environments, and so useful across a range of applications or deployments?

SCORING: Each criteria may result in a score of between 0 and 5.

3. Implementation notes

Any applicable notes arising from implementing the test suite for the library. Assess each criteria and include notes on the following:

Overall assessment of ease of use of the library (5 is 'easy', ranging to 1 for 'difficult')
Were common pitfalls or difficulties in using the library identified - please describe in detail e.g., were specific combinations of flags required to produce IDNA compliant behaviour (5 if none found, ranging to 1 if several found).
Were other issues identified which prevent this library being widely used – please describe in detail e.g., is functionality only available in deprecated functions (5 if no issues identified, ranging to 1 if several issues identified).

SCORING: See specific criteria.

These criteria will be refined in future versions of this document as experience is gained with performing evaluations.

4. Technical evaluation

4.1. Test suite

The test suite is designed to evaluate compliance with behaviour described in RFCs. The great advantage of standards based evaluations is the relative ease with which compliance can measured and the general interoperability that results.

The way in which libraries present services varies significantly between libraries. It should not be assumed that the functions given below are matched by a single function in the library under test. Rather, they are an attempt to describe function at a high enough level that results from a test suite created for a particular library are comparable with results from other libraries.

Evaluators are encouraged to consider if the test suite can be written in a way that aligns with the current test framework of the library under adoption. Evaluators can then easily make library maintainers aware of the tests and maximise the chances of future adoption of the tests by the library maintainers.

A summary of the test cases is given below including a test case ID which is used later in the document. A more detailed description, notes and illustrative test data follow.

Low-level functions (see section 4.1.1)

L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form

L-R2A: IDNA2008 - Convert registration label to ASCII registry form

L-A2U: IDNA2008 - Convert ASCII domain name to Unicode

L-DNC: IDNA2008 - Domain name equivalence comparison
For libraries that only support IDNA2003, the tests should be implemented using the available IDNA2003 functions. Having only IDNA2003 support will give rise to test failures, and so produce a lower score.

High level functions (see section 4.1.2)

H-DNS: Domain name - syntactic check

H-DND: Domain name - decompose into components labels

H-ES: Email address - syntactic check

H-ED: Email address - decompose into components

H-US: URL - syntactic check

H-UD: URL- decompose into components

Appendix B provides some code samples for example test cases from the test suite.

Test cases

The full technical description of the proposed test cases is in Appendix C. Each test case has an input description, expected result, test purpose and standards reference. The document describes ~150 test cases in total, each of which should have at least one piece of test data, preferably multiple where applicable. *It is intended as a technical reference for evaluators.*

Example test data
The descriptions of the tests below are accompanied by a small number of example test data items with brief descriptions, largely drawn from UASG0004. *These are for illustrative purposes only (no more than 10 items are given per test) and cover a small subset of the test cases in the full technical description.*

SCORING: For a given library, it may or may not be possible to implement a particular function. If the library does not support the function, the test score is 0. Otherwise an overall score S is generated by multiplying the % pass rate by 0.05. This produces a relative score out of 5 for each test case.

-
- **4.1.1 Low-level functions**

Low-level functions provide basic transformation services required by the relevant IDNA RFCs.

- 4.1.1.1 L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form

Scenario: Convert 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.

References: RFC5891, UTS#46

Sample test data:

Input	Expected output	Comment
ua-test.link	ua-test.link	Verify ASCII passed unaltered.
普遍接受-测试.top	xn----f38am99bqvcd5liy1cxsg.top	Verify subdomain conversion.
ua-test.世界	ua-test.xn--rhqv96g	Verify TLD conversion.
普遍接受-测试.世界	xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	Verify all-Unicode conversion.
普遍接受-测试。世界	xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	Verify Open Dot is recognised as label separator.
ua-test.xn--rhqv96g	ua-test.xn--rhqv96g	Verify ACE encoded TLD is passed as ASCII.
xn----f38am99bqvcd5liy1cxsg.top	xn----f38am99bqvcd5liy1cxsg.top	Verify ACE encoded subdomain is passed as ASCII.
xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	Verify all-ACE encoded domain is passed as ASCII.
fußballplatz.de	xn--fuballplatz-w6a.de (non-transitional)	Verify IDNA2008 processing.

- 4.1.1.2 L-R2A: IDNA2008 - Convert registration label to ASCII registry form

Scenario: Convert a registration label to ASCII using the process described in RFC5891 Section 4. Input to this process must be a U-label, preferably accompanied by the expected A-label, or an A-label.

Note: Because it examines functionality as defined in RFC5891, this test is concerned specifically with labels, not domain names. A label is a component of a domain name; the domain name www.example.com consists of three labels, www, example and com.

References: RFC5891, RFC5893, UTS#46

Sample test data:

Input (U-label)	Valid?	Input (A-label) and Expected output	Comment
ua-test.link	N		Not single label.

ua-test	Y	ua-test	Input A-label.
普遍接受-测试.top	N		Not single label.
普遍接受-测试	Y	xn----f38am99bqvcd5liy1cxsg	Verify Unicode conversion.
普遍接受-测试。世界	N		Open Dot means not single label.
fußballplatz	Y	xn--fuballplatz-w6a	Verify IDNA2008.

- 4.1.1.3 L-A2U: IDNA2008 - Convert ASCII domain name to Unicode

Scenario: Convert 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.

References: RFC5891

Sample test data:

Input	**Expected output**	**Comment**
ua-test.link	ua-test.link	Verify ASCII passed unaltered.
xn----f38am99bqvcd5liy1cxsg.top	普遍接受-测试.top	Verify Unicode conversion in subdomain.
ua-test.xn--rhqv96g	ua-test.世界	Verify Unicode conversion in TLD.
xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	普遍接受-测试.世界	Verify all-Unicode conversion.
xn--fuballplatz-w6a.de	fußballplatz.de	Verify IDNA2008.

- 4.1.1.4 L-DNC: IDNA2008 - Domain name equivalence comparison

Scenario: Compare two Unicode domain names for equivalence. Comparison must be performed as specified in RFC5891, comparing either A-label or U-labels.

References: RFC5891

Sample test data:

Name 1	**Name 2**	**Equivalent?**	**Comment**
ua-test.link	ua-test.link	Yes	Verify ASCII.
xn----f38am99bqvcd5liy1cxsg.TOP	普遍接受-测试.top	Yes	Verify subdomain Unicode conversion and ASCII case insensitive.

普遍接受-测试.top	普遍接受-测试.top	Yes	Verify Unicode subdomain.
ua-test.世界	ua-test.世界	Yes	Verify Unicode TLD.
ua-test.xn--rhqv96g	ua-test.世界	Yes	Verify ASCII/Unicode TLD.
xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	普遍接受-测试.世界	Yes	Verify ASCII/Unicode.
xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g	普遍接受-测试。世界	Yes	Verify ASCII/Unicode with Open Dot.
普遍接受-测试.世界	普遍接受-测试。世界	Yes	Verify Unicode/Unicode with Open Dot
fußballplatz.de	xn--fuballplatz-w6a.de	Yes	Verify IDNA2008

- **4.1.2 High-level functions**

High-level functions reflect operations that an application is likely to want to perform but which are not directly detailed in an IDNA RFC, rather they either build on the operations specified therein or on separate RFCs relating to identifiers.

High-level functions provide basic syntactic checks and decomposition functions described by the relevant RFCs for the identifier. For the purposes of this document, we define a *syntactic check* as a check that a value obeys the rules of form (typically defined in a RFC) for that identifier. In other words, that the value is a potentially valid. So, for example, for a value to be a syntactically valid domain name it must pass all the rules laid down in the relevant RFCs for a domain name - overall length and individual label lengths must be within the prescribed limits, it must not contain any disallowed code points etc.

These functions may be provided directly by the library or implemented with standard library functions.

- 4.1.2.1 H-DNS: Domain name - syntactic check

Scenario: Perform a syntactic check on a domain name. Determine 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.

References: RFC5891, RFC1035, SAC053

Sample test data:

Name	**Syntactically correct?**	**Comment**
ua-test.link	Yes	Verify ASCII.

xn----f38am99bqvcd5liy1cxsg.TOP	Yes	Verify ACE plus ASCII.
普遍接受-测试.top	Yes	Verify Unicode subdomain.
ua-test.世界	Yes	Verify Unicode TLD.
ua-test.invalid	Yes	Verify non-existent domain,to ensure check is purely syntactic.
ua-test..invalid	No	Verify empty label prohibited.

- 4.1.2.2 H-DND: Domain name - decompose into components

Scenario: Split a domain name into its component labels.

References: RFC5891, UTS#46, SAC053

Sample test data:

Name	Components	Comment
ua-test.link	ua-test link	Verify ASCII.
xn----f38am99bqvcd5liy1cxsg.TOP	xn----f38am99bqvcd5liy1cxsg TOP	Verify ACE plus ASCII.
普遍接受-测试.top	普遍接受-测试 top	Verify Unicode subdomain.
ua-test.世界	ua-test 世界	Verify Unicode TLD.
普遍接受-测试。世界	普遍接受-测试 世界	Verify Open Dot as label separator.

- 4.1.2.3 H-ES: Email- syntactic check

Scenario: Perform a syntactic check on an email address. Determine whether the address appears to be correctly formed.

References: RFC5891, RFC6531

Sample test data:

Name	Syntactically correct?	Comment
info@ua-test.link	Yes	Verify ASCII.
info@普遍接受-测试.top	Yes	Verify ASCII with Unicode subdomain.

info@普遍接受-测试.世界	Yes	Verify ASCII mailbox, Unicode domain.
données@ua-test.link	Yes	Verify Unicode mailbox, ASCII domain.
info@ua-test.invalid	Yes	Verify non-existent domain.
info@@ua-test.technology	No	Verify single @.
info@ua-test..technology	No	Verify empty label disallowed.

- 4.1.2.4 H-ED: Email- decompose into components

Scenario: Split an email address into its component parts, the mailbox name and the domain name.

References: RFC5891, RFC6531

Sample test data:

Name	Components	Comment
info@ua-test.link	info ua-test.link	Verify ASCII.
info@普遍接受-测试.top	info 普遍接受-测试.top	Verify ASCII mailbox, Unicode subdomain.
info@普遍接受-测试.世界	info 普遍接受-测试.世界	Verify ASCII mailbox, Unicode domain.
données@ua-test.link	données ua-test.link	Verify Unicode mailbox.

- 4.1.2.5 H-US: URL - syntactic check

Scenario: Perform a syntactic check on a URL. Determine whether the URL appears to be correctly formed.

Verifying URL syntax is complex. These tests focus on UA aspects.

References: RFC3897

Sample test data:

Name	Syntactically correct?	Comment
http://ua-test.link/	Yes	Verify ASCII.
ftp://ua-test.technology:8125/resource.file	Yes	Verify ASCII with port and path.

https://普遍接受-测试.top/我的页面	Yes	Verify Unicode subdomain, Unicode path.
données://ua-test.link/	No	Verify ASCII scheme.
nosuchscheme://普遍接受-测试.世界/我的页面?arg=99	Yes	Verify Unicode domain, path, ASCII argument.
nosuchscheme://普遍接受-测试.世界/我的页面#données	Yes	Verify Unicode domain, path, Unicode fragment.
http://ua-test.invalid/	Yes	Verify non-existent domain.
http://ua-test..technology/	No	Verify empty label disallowed.
https://普遍接受-测试.top:bad/我的页面	No	Verify non-numeric port disallowed.

- 4.1.2.6 H-UD: URL- decompose into components

Scenario: Split a URL into its component parts. The degree of decomposition may vary; for the purposes of this document, the major concern is that the domain name is extracted correctly. The test case examples below do not decompose the path, parameters or named anchor components.

References: RFC3897

Sample test data:

Name	**Components**	**Comment**
http://ua-test.link/	http ua-test.link	Verify ASCII.
ftp://ua-test.technology:8125/resource.file	ftp ua-test.technology 8125 resource.file	Verify ASCII hostname, port and path.
https://普遍接受-测试.top/我的页面	https 普遍接受-测试 .top 我的页面	Verify
nosuchscheme://普遍接受-测试.世界/我的页面?arg=99	nosuchscheme 普遍接受-测试.世界 我的页面?arg=99	Verify Unicode domain, path, ASCII argument.
nosuchscheme://普遍接受-测试.世界/我的页面#données	nosuchscheme 普遍接受-测试.世界	Verify Unicode domain, path, Unicode fragment.

	我的页面 #données	

4.2 Report on "Semantic Checks" functionality

Whilst the test suite can evaluate whether or not an identifier is structurally correct, this section deals with semantic checks on identifiers, that is, whether the identifier is or can be meaningfully used.

- **Problem statement**

Semantic checks are much more challenging to describe and evaluate via a test suite for a number of reasons.

A semantic check can take a wide variety for forms, for example:

A check that the value for an identifier does currently 'exist'.
A check that part of an identifier does currently 'exist'.
A check that part of an identifier could 'exist' i.e. a check that a value (or part of a value) is not prohibited by some operational or administrative policy.

The definition of 'exists' is not always clear and is often identifier specific.

How such checks are performed are not typically described in RFCs. It may be possible to perform some of them using aspects of a protocol defined in an RFC however, more commonly they can often rely on:

Sources that are not authoritative to prove existence
Sources that are not updated in real-time
Sources that do not expose programmatic APIs

To evaluate **how** a check is done often requires inspection of the source code or a dependance on statements made in the libraries documentation.

At this early stage it is expected that few libraries offer a comprehensive set of semantic checks (if any) and so comparisons will be particularly difficult and potentially misleading. It is anticipated that today many such checks are performed at the application level.

As an example, it could be stated that a domain name should only be considered semantically valid, if it is not only syntactically valid but it must also exist in DNS (Class IN and Type as relevant to the semantic check being carried out), as the standards are clear that DNS is the only authoritative source of information on the existence or otherwise of a domain.

If DNS is not available, it may be possible to make a partial determination of potential semantic validity by consulting other sources used within the industry. However these are frequently not databases, have no programmatic API, make no guarantee that their content is always completely up to date with current DNS, and cannot validate subdomains; any such check is not authoritative and so is of limited usefulness. The best options for such a static lookup are the IANA published root zone or IANA TLD list (updated every 24 hours). Both are available to download as text files.

To illustrate the complexity of the task, some test cases are suggested below but their scope is limited to a narrow subset of tests for existence in the DNS and additionally assume knowledge of the details of the implementation.

SCORING: Based on these considerations, this section is not scored. However evaluators are invited to describe what if any such functionality is offered by the library. After collecting initial data on this available functionality in libraries it is expected that this section will be significantly refined.

Semantic check functions

S-DN: Domain name - semantic check

S-EA: Email - semantic check

S-U: URL - semantic check

- **4.2.1 S-DN: Domain name- semantic check**

Scenario: Perform a syntactic then a semantic check on a domain name. This semantic check is a check that a DNS type ANY lookup returns a result for the domain.

References: RFC5891, RFC1035, SAC053

Sample test data:

Name	Semantically correct?	Comment
ua-test.link	Yes	Verify ASCII.
xn----f38am99bqvcd5liy1cxsg.TOP	Yes	Verify ACE plus ASCII.
普遍接受-测试.top	Yes	Verify Unicode subdomain.
ua-test.世界	Yes	Verify Unicode TLD.
ua-test.invalid	No	Verify non-existent domain disallowed.

- **4.2.2 S-EA: Email- semantic check**

Scenario: Perform a syntactic and then a semantic check on an email address. This semantic check is only a check that a Type MX resource record exists for the domain. The existence or otherwise of the mail server or the mailbox is not determined.

References: RFC5891, RFC6531

Sample test data:

Name	Semantically correct?	Comment
info@ua-test.link	Yes	Verify ASCII.
info@普遍接受-测试.top	Yes	Verify ASCII with Unicode subdomain.
info@普遍接受-测试.世界	Yes	Verify ASCII mailbox, Unicode domain.

données@ua-test.link	Yes	Verify Unicode mailbox, ASCII domain.
info@ua-test.invalid	No	Verify non-existent domain disallowed.

- **4.2.3 S-U: URL- semantic check**

Scenario: Perform a syntactic and then a semantic check on a URL. This semantic check is only a check that either a Type A or a Type AAAA resource record exists for the domain. Items not checked include:

The existence or otherwise of the server.
The validity of the protocol.
Whether the server accepts requests on the indicated port for the indicated protocol.
Whether a resource at the indicated path exists.

References: RFC3897

Sample test data:

Name	Semantically correct?	Comment
http://ua-test.link/	Yes	Verify ASCII.
ftp://ua-test.technology:8125/resource.file	Yes	Verify ASCII with port and path.
https://普遍接受-测试.top/我的页面	Yes	Verify Unicode subdomain, Unicode path.
nosuchscheme://普遍接受-测试.世界/我的页面?arg=99	No	Verify non-existant scheme disallowed.
http://ua-test.invalid/	No	Verify non-existent domain disallowed.

5. Mitigation Actions

In this section of the report recommended mitigation actions should be detailed. Ideally a numbered list of actions is given in each section for ease of tracking.

Major Actions

Major Actions would include requests (where appropriate) to implement missing functionality or direct contact with the library maintainer where significant test failures were seen for several test cases.

Minor Actions

Minor actions would be e.g., bug reports for a handful of failures or inaccurate/misleading documentation.

Appendix A - Scoring template

1. Basic information

Library name and version	
Provider/how obtained	
Test suite/library binding language	
Test environment - processor architecture	
Test environment - Operating system	
License	
Financial cost	
Evaluator information	
Comments	

2. Auxiliary information

Score items using this scale:

0	1	2	3	4	5
Not Supported	Very Weak	Weak	Acceptable	Strong	Very Strong

Item	Score 0-5	Comments
Public bug tracker		
Maintenance activity		
Documentation		
Source code availability		
Support channels		
Maturity and adoption		
Standards support		
Portability		
TOTAL		

3. Implementation notes

Item	Score 1-5	Comments
Overall ease of use 1=Difficult .. 5=Easy		
Common pitfalls 1=Several found .. 5=None found		
Other issues preventing wide use 1=Several identified .. 5=None identified		
TOTAL		

4. Test suite

ID	Description	Score 0-5
L-U2A	IDNA2008 - Convert Unicode domain name to ASCII lookup form	
L-R2A	IDNA2008 - Convert registration label to ASCII registry form	
L-A2U	IDNA2008 - Convert ASCII domain name to Unicode	
L-DNC	IDNA2008 - Domain name equivalence comparison	
H-DNS	Domain name - syntactic check	
H-DND	Domain name - decompose into components	
H-ES	Email address - syntactic check	
H-ED	Email address - decompose into components	
H-US	URL - syntactic check	
H-UD	URL- decompose into components	
	TOTAL	

Semantic check commentary

ID	Description	Comments
S-DN	Domain name semantic check	
S-EA	Email address semantic check	
S-U	URL semantic check	

-

5. Mitigation Actions

Major Actions	
Minor Actions	

Overall score

Auxiliary information total	A	
Implementation notes total	B	
Non-functional criteria total	**C=A+B**	
Test suite total	D	
25% of non-functional criteria total	E=C/4	
GRAND TOTAL	**F=D+E**	

Appendix B - Code examples

In practice, the low-level and high-level functions above are unlikely to be implemented in a consistent fashion across different libraries. This appendix gives examples of how some might be implemented as part of the test case using different libraries.

1. L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form

GNU Libidn2 (C)

```
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <idn2.h>

int main(int ac, char *av[])
{
  const char *name = u8"普遍接受-测试.世界";
  int rc;
  char *lookupname;

  setlocale(LC_ALL, "");

  rc = idn2_lookup_ul(name, &lookupname, 0);
  if ( rc != IDN2_OK )
  {
     fprintf(stderr,
           "error: %s (%s, %d)\n",
           idn2_strerror(rc),
```

```
            idn2_strerror_name(rc),
            rc);
      return 1;
  }
  printf("DNS lookup of %s: %s\n", name, lookupname);
  free(lookupname);
  return 0;
}
```

```
$ ./a.out
DNS lookup of 普遍接受-测试.世界: xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g
```

2. L-A2U: IDNA2008 - Convert ASCII domain name to Unicode

npm idna-uts46 (Javascript)

```
'use strict';

var uts46 = require('idna-uts46');
var ascii = "xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g";
var unicode = uts46.toUnicode(ascii);

console.log("DNS " + ascii + ": " + unicode);
```

```
$ js example.js
DNS xn----f38am99bqvcd5liy1cxsg.xn--rhqv96g: 普遍接受-测试.世界
```

Appendix C - Notes on development of test data sets

The text to this appendix is attached at the end of this document.

Appendix D - References

UASG Documents

UASG004 - Use Cases for UA Readiness Evaluation

IDNA RFCs

RFC3492 - Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)
RFC5890 - Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework
RFC5891 - Internationalized Domain Names in Applications (IDNA): Protocol
RFC5892 - The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)
RFC5893 - Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)
RFC5894 - Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale

IANA Registries

IDNA Parameters (IDNA Contextual Rules and Derived Properties)

Unicode

Unicode Technical Standard #46 - Unicode IDNA Compatibility Processing

Special-use domain name RFCs

Some domain names are reserved for special use; that is, their use requires special handling at some point in the name resolution process. A full list of these names is given in IANA list of special-use domain names.
RFC6761 - Special-use domain names
RFC6762 - Multicast DNS
RFC7686 - The ".onion" special-use domain name

Internationalized email RFCs

RFC6530 - Overview and Framework for Internationalized Email
RFC6531 - SMTP Extension for Internationalized Email

Internationalized resource identifier RFCs

RFC3987 - Internationalized Resource Identifiers (IRIs)

Obsolete IDNA RFCs

These describe IDNA2003. They are kept here for reference when dealing with libraries that only support IDNA2003.
RFC3490 - Internationalizing Domain Names in Applications (IDNA)
RFC3491 - Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)

Appendix E - Future actions

The following are considerations that we think should be borne in mind for a future revision of this document after a sufficient number of evaluations have been performed to provide further input.

Review how to work with library maintainers to improve unit test coverage based on this work.
Reconsider the split of functions between high and low level, and the nomenclature used.
If evaluations show it is relevant, add details on test environment client/server/network hardware to the evaluation basic information.
Revise scoring system in light of experience with evaluations.
Consider adding tests of URLs embedded in URL parameters.
Revisit the question of Semantic Checks, what they should be and how they should be scored.
Review tracking of mitigation actions and potential updates to evaluation reports. Consider, for example, tracking all bugs files with library vendors, the latest evaluation reports etc.

Appendix C - Notes on development of test data sets

TABLE OF CONTENTS

Preamble

As noted in section 4 of the main document, the sample test data given in the document at present is illustrative and deliberately incomplete. We recognise that practical experience of performing evaluations is necessary to guide the development of test data that will be of practical benefit in delivering useful evaluation results. This appendix gives the design outline for test data sets for Universal Acceptance programming library assessment.

As noted, sharing of the test data sets developed will be highly beneficial to this process.

Requirements for test data sets

These tests are not designed to perform an exhaustive probe for all possible implementation problems in a library. Rather, the aim of the tests is to provide

- an indicator of the usefulness of the library;
- is the implementation generally correct, likely to be useful for most practical applications?
- are the common and some corner error cases correctly detected?

To that end, the tests are divided into *General* and *Specific* tests.

- *General* tests are tests that check overall common functionality works as expected; each test should be run several times with inputs reflecting cases likely to reflect real-world use, and each run counted as a different test result. To given a more concrete example, *general* domain name tests for Unicode base multilingual plane support should be run with input domain names from a variety of common scripts that might approximate expected real-world use, e.g. French, Chinese, Hindi, and Arabic. In this example, a single test will produce 4 test results.

- *Specific* tests check one particular piece of functionality works to specification; they have only the minimum input required to give an answer to the question posed by the test case. So, for example, a single instance of a domain with a Unicode combining mark as the first character is sufficient for test L-U2AS6 below. Similarly, some *specific* tests may in fact be covered by data from a *general* test; for example, test L-U2AS1 is likely to be covered by data for L-U2AG5. These test descriptions are included for completeness, but a test should be omitted, as it would be merely repeating an existing test.

The goal here is to ensure that the overall ratio of the tests passed to tests failed reflects the general usefulness and quality of the library.
The recommended balance of tests is to ensure that at least 50% of the test data for each test case are for *General* tests, such that a library with reasonable coverage of the basics will score at least 50%. The remaining test data will differentiate between such libraries and ones that also correctly handle the most common UA corner cases.

Test Data Sets

The following sections specify the different tests for each category. Each test includes a reference to the standards document that specifies the behaviour the test is examining. There are several relevant standards, and they are not always completely consistent; hence the need for a reference to guide the reader wishing to trace the requirement back to its source.

A set of test data for the low-level (L-) functions based on this document is under development and is intended for use by developers when implementing the evaluation. The data is currently divided into *valid* (expected to convert without error) and *invalid* (expected to give an error). These can be conveniently viewed in PDF form at valid-domains and invalid-domains. The authoritative original data is available in UTF-8 text files valid-domains.txt and invalid-domains.txt.

Low-level functions

For current data on UNASSIGNED, DISALLOWED, CONTEXTJ and CONTEXTO, see this IANA page.
A comprehensive set of test data for these functions is available from the Unicode Consortium. As befits a comprehensive test set, it contains a large number of tests probing for implementation weaknesses in the more obscure areas of the standards, and so lacks the balance of tests required. However, it provides a fruitful source of raw test data.

L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form

Convert a domain name in Unicode to ASCII using the process described in RFC5891 for domain name lookup.

General tests:

Test ID	Input: domain comprising the following, with expected ASCII output	Expected error	Test purpose	Reference
L-U2AG1	Plain ASCII	None	Verify that ASCII is passed through unaltered	RFC5891
L-U2AG2	Plain ASCII with >3 char TLD	None	Verify long TLDs are handled	RFC5891

L-U2AG3	Permitted non-ASCII from Unicode base multilingual plane with ASCII TLD	None	Verify basic Unicode support	RFC5891
L-U2AG4	Permitted non-ASCII TLD from Unicode base multilingual plane with ASCII rest of domain	None	Verify basic Unicode support	RFC5891
L-U2AG5	Permitted non-ASCII from Unicode base multilingual plane - entire domain	None	Verify basic Unicode support	RFC5891
L-U2AG6	Permitted non-ASCII from right to left script in Unicode base multilingual plane, complying with Bidi Rule (RFC5893)	None	Verify basic Unicode support	RFC5891
L-U2AG7	Permitted non-ASCII from Unicode supplementary multilingual plane - entire domain	None	Verify Unicode support for higher planes	RFC5891

Specific tests:

Test ID	Input: domain comprising the following, with expected ASCII output	Expected error	Test purpose	Reference
L-U2AS1	Permitted non-ASCII from Unicode base multilingual plane, labels separated with . FULL STOP (U+002E)	None	Verify basic Unicode support	UTS#46
L-U2AS2	Permitted non-ASCII from Unicode base multilingual plane, labels separated with . FULLWIDTH FULL STOP (U+FF0E)	None	Verify basic Unicode support	UTS#46

L-U2AS3	Permitted non-ASCII from Unicode base multilingual plane, labels separated with 。IDEOGRAPHIC FULL STOP (U+3002)	None	Verify basic Unicode support	UTS#46
L-U2AS4	Permitted non-ASCII from Unicode base multilingual plane, labels separated with 。HALFWIDTH IDEOGRAPHIC FULL STOP (U+FF61)	None	Verify basic Unicode support	UTS#46
L-U2AS5	Permitted non-ASCII from Unicode base multilingual plane with '–' (two consecutive hyphens) in the third and fourth character positions	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS6	Permitted non-ASCII from Unicode base multilingual plane with a combining mark as a first character	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS7	Permitted non-ASCII from Unicode base multilingual plane but containing a DISALLOWED character in a label	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS8	Permitted non-ASCII from Unicode base multilingual plane but containing a conforming CONTEXTJ character in a label	None	Verify CONTEXTJ support	RFC5891
L-U2AS9	Permitted non-ASCII from Unicode base multilingual plane but containing a non-conforming CONTEXTJ character in a label	Reject	Verify CONTEXTJ support	RFC5891
L-U2AS10	Permitted non-ASCII from Unicode base multilingual plane but containing a conforming CONTEXTO character in a label	None	Verify CONTEXTO support	RFC5891
L-U2AS11	Permitted non-ASCII from Unicode base multilingual plane but containing an UNASSIGNED character in a label	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS12	Permitted non-ASCII from Unicode base multilingual plane but containing a label that is 64 characters or longer in ACE form	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS13	Permitted non-ASCII from Unicode supplementary multilingual plane but containing a DISALLOWED character in a label	Reject	Ensure malformed Unicode is rejected	RFC5891
L-U2AS14	Permitted non-ASCII from Unicode supplementary multilingual plane but containing an UNASSIGNED character in a label	Reject	Ensure malformed Unicode is rejected	RFC5891

L-U2AS15	Permitted non-ASCII from Unicode base multilingual plane, not in Unicode Normalization Form C (NFC)	None	Ensure NFC processing happens before conversion	RFC5891

L-R2A: IDNA2008 - Convert registration label to ASCII registry form

Convert a registration label to ASCII using the process described in RFC5891 Section 4. Input to this process must be a U-label, preferably accompanied by the expected A-label, or an A-label.

General tests:

Test ID	Input: label comprising the following, with expected ASCII output	Expected error	Test purpose	Reference
L-R2AG1	Permitted non-ASCII from Unicode base multilingual plane	None	Verify basic Unicode support	RFC5891
L-R2AG2	Permitted non-ASCII from right to left script in Unicode base multilingual plane, complying with Bidi Rule (RFC5893)	None	Verify basic Unicode support	RFC5891
L-R2AG3	Permitted non-ASCII from Unicode supplementary multilingual plane	None	Verify Unicode support for higher planes	RFC5891

Specific tests:

Test ID	Input: label comprising the following, with expected ASCII output	Expected error	Test purpose	Reference
L-R2AS1	Plain ASCII label	Accept	Verify that ASCII is passed through unaltered	RFC5891
L-R2AS2	Permitted non-ASCII from Unicode base multilingual plane containing . FULL STOP (U+002E)	Reject	Check input is label	UTS#46

L-R2AS3	Permitted non-ASCII from Unicode base multilingual plane containing . FULLWIDTH FULL STOP (U+FF0E)	Reject	Check input is label	UTS#46
L-R2AS4	Permitted non-ASCII from Unicode base multilingual plane containing 。 IDEOGRAPHIC FULL STOP (U+3002)	Reject	Check input is label	UTS#46
L-R2AS5	Permitted non-ASCII from Unicode base multilingual plane containing 。 HALFWIDTH IDEOGRAPHIC FULL STOP (U+FF61)	Reject	Check input is label	UTS#46
L-R2AS6	Permitted non-ASCII from Unicode base multilingual plane with '–' (two consecutive hyphens) in the third and fourth character positions	Reject	Ensure malformed Unicode is rejected	RFC5891
L-R2AS7	Permitted non-ASCII from Unicode base multilingual plane with a combining mark as a first character	Reject	Ensure malformed Unicode is rejected	RFC5891
L-R2AS8	Permitted non-ASCII from Unicode base multilingual plane but containing a DISALLOWED character	Reject	Ensure malformed Unicode is rejected	RFC5891
L-R2AS9	Permitted non-ASCII from Unicode base multilingual plane but containing a conforming CONTEXTJ character	None	Verify CONTEXTJ support	RFC5891
L-R2AS10	Permitted non-ASCII from Unicode base multilingual plane but containing a non-conforming CONTEXTJ character	Reject	Verify CONTEXTJ support	RFC5891
L-R2AS11	Permitted non-ASCII from Unicode base multilingual plane but containing a conforming CONTEXTO character	None	Verify CONTEXTO support	RFC5891
L-R2AS12	Permitted non-ASCII from Unicode base multilingual plane but containing a non-conforming CONTEXTO character	Reject	Verify CONTEXTO support	RFC5891
L-R2AS13	Permitted non-ASCII from Unicode base multilingual plane but containing an UNASSIGNED character	Reject	Ensure malformed Unicode is rejected	RFC5891
L-R2AS14	Permitted non-ASCII from Unicode base multilingual plane which is 64 characters or longer in ACE form	Reject	Ensure malformed Unicode is rejected	RFC5891

L-R2AS15	Permitted non-ASCII from Unicode base multilingual plane, but A-label not in all lowercase	Reject	A-label validation	RFC5891
L-R2AS16	Permitted non-ASCII from Unicode base multilingual plane, but A-label ending '-' (hyphen)	Reject	A-label validation	RFC5891
L-R2AS17	Permitted non-ASCII from Unicode base multilingual plane, but A-label->U-label does not match supplied U-label	Reject	U/A-label matching	RFC5891
L-R2AS18	Only A-label, not all in lowercase	Reject	A-label validation	RFC5891
L-R2AS19	Only A-label, that ends in '-' (hyphen)	Reject	A-label validation	RFC5891
L-R2AS20	Permitted non-ASCII from Unicode base multilingual plane, not in Unicode Normalization Form C (NFC)	Reject	Registry input must be NFC	RFC5891
L-R2AS21	Permitted non-ASCII from right to left script in Unicode base multilingual plane, not complying with Bidi Rule	Reject	See if Bidi checking happens.	RFC5891/5893

L-A2U: IDNA2008 - Convert ASCII domain name to Unicode

Convert a domain name in ASCII to Unicode using the process described in RFC5891.

General tests:

Test ID	Input: domain comprising the following ASCII, with expected Unicode output	Expected error	Test purpose	Reference
L-A2UG1	Plain ASCII	None	Verify that ASCII is passed through unaltered	RFC5891
L-A2UG2	Plain ASCII with >3 char TLD	None	Verify long TLDs are handled	RFC5891
L-A2UG3	ACE domain with ASCII TLD	None	Verify basic Unicode support	RFC5891
L-A2UG4	ACE TLD with ASCII rest of domain	None	Verify basic Unicode support	RFC5891

L-A2UG5	Permitted non-ASCII from Unicode base multilingual plane - entire domain	None	Verify basic Unicode support	RFC5891
L-A2UG6	Permitted non-ASCII from Unicode supplementary multilingual plane - entire domain	None	Verify basic Unicode support	RFC5891

Specific tests:

Test ID	Input: domain comprising the following ASCII, with expected Unicode output	Expected error	Test purpose	Reference
L-A2US1	A-label, not all in lowercase	Reject	A-label validation	RFC5891
L-A2US2	A-label that ends in '-' (hyphen)	Reject	A-label validation	RFC5891

L-DNC: IDNA2008 - Domain name equivalence comparison

Compare two Unicode domain names for equivalence. Comparison must be performed as specified in RFC5891, comparing either A-label or U-labels.

The input domains for the following tests should be those used in the tests described in L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form. Each input should be used to create two test cases; checking that comparing the domain succeeds, and a counter-example checking that comparing the domain to a different domain fails.

General tests:

Test ID	Input: domain comprising the following	Compare to	Result	Expected error	Test purpose	Reference
L-DNCG1	Plain ASCII	Plain ASCII	Equal	None	Verify that ASCII is passed through unaltered	RFC5891

L-DNCG2	Plain ASCII with >3 char TLD	Plain ASCII	Equal	None	Verify long TLDs are handled	RFC5891
L-DNCG3	Permitted non-ASCII from Unicode base multilingual plane with ASCII TLD	ACE encoding	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG4	Permitted non-ASCII from Unicode base multilingual plane with ASCII TLD	Unicode	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG5	Permitted non-ASCII TLD from Unicode base multilingual plane with ASCII rest of domain	ACE encoding	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG6	Permitted non-ASCII TLD from Unicode base multilingual plane with ASCII rest of domain	Unicode	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG7	Permitted non-ASCII from Unicode base multilingual plane - entire domain	ACE encoding	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG8	Permitted non-ASCII from Unicode base multilingual plane - entire domain	Unicode	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG9	Permitted non-ASCII from Unicode supplementary multilingual plane - entire domain	ACE encoding	Equal	None	Verify basic Unicode support	RFC5891
L-DNCG10	Permitted non-ASCII from Unicode supplementary multilingual plane - entire domain	Unicode	Equal	None	Verify basic Unicode support	RFC5891

Specific tests:

Test ID	Input: domain comprising the following	Compare to	Result	Expected error	Test purpose	Reference
L-DNCS1	Unicode that does not form a valid domain	Same Unicode		Reject	Check Unicode validation	RFC5891

L-DNCS2	Permitted non-ASCII from Unicode base multilingual plane - entire domain, label separator . FULL STOP (U+002E)	Unicode, but using alternate label separator e.g. 。IDEOGRAPHIC FULL STOP (U+3002)	Not equal	None	Check comparison is Unicode	RFC5891
L-DNCS3	Permitted non-ASCII from Unicode base multilingual plane with ASCII TLD, ACE encoded, TLD capitalised	ACE encoding, TLD not capitalised	Equal	None	Check comparison is case-insensitive ASCII	RFC5891

High-level functions

H-DNS: Domain name - syntactic check

Perform a syntactic check on a domain name. Determine 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.
This test should run all the tests described in L-U2A: IDNA2008 - Convert Unicode domain name to ASCII lookup form above and verify that the conversion does not produce an error. In addition, the following tests should also be run. These are all *specific* tests.

Test ID	Input: domain comprising the following	Correct?	Test purpose	Reference
H-DNSS1	Permitted non-ASCII from Unicode base multilingual plane with ASCII '.invalid' TLD	Yes	Verify Unicode support	RFC5891
H-DNSS2	Permitted non-ASCII from Unicode base multilingual plane with empty label ('..')	No	Check domain composition	RFC1035

H-DNSS3	Permitted non-ASCII from Unicode base multilingual plane with no label separator character, i.e. none of the following: . FULL STOP (U+002E) . FULLWIDTH FULL STOP (U+FF0E) 。 IDEOGRAPHIC FULL STOP (U+3002) 。 HALFWIDTH IDEOGRAPHIC FULL STOP (U+FF61)	No	Check domain composition	SAC053

H-DND: Domain name - decompose into components

Split a domain name into its component labels.
Tests for this function do not test for domain name validity.

General tests:

Test ID	Input: domain comprising the following	Expected error	Test purpose	Reference
H-DNDG1	Plain ASCII	None	Verify basic support	RFC5891
H-DNDG2	Plain ASCII with >3 char TLD	None	Verify long TLDs are handled	RFC5891
H-DNDG3	Permitted non-ASCII from Unicode base multilingual plane with ASCII TLD, labels separated with . FULL STOP (U+002E)	None	Verify basic support	UTS#46
H-DNDG4	Permitted non-ASCII TLD from Unicode base multilingual plane with ASCII rest of domain, labels separated with . FULL STOP (U+002E)	None	Verify basic support	UTS#46
H-DNDG5	Permitted non-ASCII from Unicode base multilingual plane - entire domain, labels separated with . FULL STOP (U+002E)	None	Verify basic support	UTS#46
H-DNDG6	Permitted non-ASCII from Unicode base multilingual plane - entire domain, single label	Reject	Check domain composition	SAC053

Specific tests:

Test ID	Input: domain comprising the following	Expected error	Test purpose	Reference
H-DNDS1	Permitted non-ASCII from Unicode base multilingual plane - entire domain, labels separated with . FULLWIDTH FULL STOP (U+FF0E)	None	Verify basic support	UTS#46
H-DNDS2	Permitted non-ASCII from Unicode base multilingual plane - entire domain, labels separated with 。IDEOGRAPHIC FULL STOP (U+3002)	None	Verify basic support	UTS#46
H-DNDS3	Permitted non-ASCII from Unicode base multilingual plane - entire domain, labels separated with ｡HALFWIDTH IDEOGRAPHIC FULL STOP (U+FF61)	None	Verify basic support	UTS#46

H-ES: Email address - syntactic check

Perform a syntactic check on an email address. Determine whether the address appears to be correctly formed.

General tests:
The *general* test email addresses should include all domain test cases from the *general* tests from Domain name: syntactic check.

Test ID	Input: email address comprising the following	Expected error	Test purpose	Reference
H-ESG1	Plain ASCII local part, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verify Unicode support	RFC6531
H-ESG2	Unicode local part from base multilingual plane, '@' plain ASCII domain	None	Verify Unicode support	RFC6531
H-ESG3	Unicode local part from base multilingual plane, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verify Unicode support	RFC6531
H-ESG4	Unicode local part from base multilingual plane including Bidi text, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verify Unicode support	RFC6531
H-ESG5	Unicode local part from supplementary multilingual plane, '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verifying local part handling	RFC6531

Specific tests:

Test ID	Input: email address comprising the following	Expected error	Test purpose	Reference
H-ESS1	Plain ASCII local part including '@', '@' plain ASCII domain	Reject	Verifying local part handling	RFC6531
H-ESS2	Quoted plain ASCII string local part including '@', '@' plain ASCII domain	None	Verifying local part handling	RFC6531
H-ESS3	Unicode local part from base multilingual plane including '@', '@' plain ASCII domain	Reject	Verifying local part handling	RFC6531
H-ESS4	Quoted Unicode string local part from base multilingual plane including '@', '@' plain ASCII domain	None	Verifying local part handling	RFC6531
H-ESS5	Unicode local part from supplementary multilingual plane including '@', '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	Reject	Verifying local part handling	RFC6531
H-ESS6	Quoted Unicode string local part from supplementary multilingual plane, '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verify Unicode support	RFC6531
H-ESS7	Quoted Unicode string local part from supplementary multilingual plane plus '@', '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verifying local part handling	RFC6531

H-ED: Email address - decompose into components

Decompose email addresses into mailbox plus domain. These tests are not syntactic checks, but checks that the decomposition is correct.

General tests:
The *general* test email addresses should include all domain test cases from the *general* tests from Domain name: syntactic check.

Test ID	Input: email address comprising the following	Expected error	Test purpose	Reference
H-EDG1	Plain ASCII local part, '@' plain ASCII domain	None	Verify basic support	RFC6531
H-EDG2	Plain ASCII local part, '@' plain ASCII domain with >3 char TLD	None	Verify long TLDs are handled	RFC6531
H-EDG3	Plain ASCII local part, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verify Unicode support	RFC6531
H-EDG4	Unicode local part from base multilingual plane, '@' plain ASCII domain	None	Verify Unicode support	RFC6531
H-EDG5	Unicode local part from base multilingual plane, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verify Unicode support	RFC6531
H-EDG6	Unicode local part from base multilingual plane including Bidi text, '@' permitted non-ASCII from Unicode base multilingual plane domain	None	Verifying local part handling	RFC6531
H-EDG7	Unicode local part from supplementary multilingual plane, '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verifying local part handling	RFC6531

Specific tests:

Test ID	Input: email address comprising the following	Expected error	Test purpose	Reference
H-EDS1	Plain ASCII local part including '@', '@' plain ASCII domain	Reject	Verifying local part handling	RFC6531
H-EDS2	Quoted plain ASCII string local part including '@', '@' plain ASCII domain	None	Verifying local part handling	RFC6531

H-EDS3	Unicode local part from base multilingual plane including '@', '@' plain ASCII domain	Reject	Verifying local part handling	RFC6531
H-EDS4	Quoted Unicode string local part from base multilingual plane including '@', '@' plain ASCII domain	None	Verifying local part handling	RFC6531
H-EDS5	Unicode local part from supplementary multilingual plane including '@', '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	Reject	Verifying local part handling	RFC6531
H-EDS6	Quoted Unicode string local part from supplementary multilingual plane, '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verify Unicode support	RFC6531
H-EDS7	Quoted Unicode string local part from supplementary multilingual plane including '@', '@' permitted non-ASCII from Unicode supplementary multilingual plane domain	None	Verifying local part handling	RFC6531

H-US: URL - syntactic check

Perform a syntactic check on a URL. Determine whether the URL appears to be correctly formed.
A complete set of tests verifying IRI syntax is complex. These tests focus on checking the UA aspects of IRI syntax.

General tests:
The *general* test IRIs should include all domain test cases from the *general* tests from Domain name: syntactic check.

Test ID	Input: IRI comprising the following	Expected error	Test purpose	Reference
H-USG1	Plain ASCII IRI	None	Verify basic support	RFC3897
H-USG2	Plain ASCII IRI, with username from permitted Unicode and port	None	Verify basic support	RFC3987
H-USG3	IRI with path of permitted Unicode from base multilingual plane	None	Verify Unicode support	RFC3987

H-USG4	IRI with path of permitted Unicode from base multilingual plane containing permitted Bidi text	None	Verify Unicode support	RFC3987
H-USG5	IRI with path of permitted Unicode from supplementary multilingual plane	None	Verify Unicode support	RFC3987

Specific tests:

Test ID	Input: IRI comprising the following	Expected error	Test purpose	Reference
H-USS1	Plain ASCII IRI, with username and port	None	Verify basic support	RFC3987
H-USS2	Plain ASCII IRI without scheme, otherwise plain ASCII	Reject	Verify scheme checking	RFC3987
H-USS3	Unicode scheme, otherwise plain ASCII	Reject	Verify scheme checking	RFC3987
H-USS4	Plain ASCII IRI, with username and port	None	Verify basic support	RFC3987
H-USS5	Plain ASCII IRI, with username from non-permitted Unicode and port	Reject	Verify username checking	RFC3987
H-USS6	Plain ASCII IRI, with username and non-numeric port	Reject	Verify port checking	RFC3987
H-USS7	IRI with path containing non-permitted Unicode from base multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS8	IRI with path of permitted Unicode from base multilingual plane containing non-permitted Bidi text (direction formatting characters)	Reject	Verify Unicode support	RFC3987
H-USS9	IRI with path containing non-permitted Unicode from supplementary multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS10	IRI with path of permitted Unicode from base multilingual plane and '#' fragment of permitted Unicode from base multilingual plane	None	Verify Unicode support	RFC3987

H-USS11	IRI with path of permitted Unicode from base multilingual plane and '#' fragment of permitted plus private Unicode from base multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS12	IRI with path of permitted Unicode from base multilingual plane and '#' fragment including non-permitted Unicode from base multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS13	IRI with path of permitted Unicode from base multilingual plane and '#' fragment of permitted Unicode from base multilingual plane containing permitted Bidi text	None	Verify Unicode support	RFC3987
H-USS14	IRI with path of permitted Unicode from base multilingual plane and '#' fragment of permitted Unicode from base multilingual plane containing non-permitted Bidi text	Reject	Verify Unicode support	RFC3987
H-USS15	IRI with path of permitted Unicode from base multilingual plane and '#' fragment of permitted Unicode from supplementary multilingual plane	None	Verify Unicode support	RFC3987
H-USS16	IRI with path of permitted Unicode from base multilingual plane and '#' fragment including non-permitted Unicode from supplementary multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS17	IRI with path of permitted Unicode from base multilingual plane and '?' query of permitted Unicode from base multilingual plane	None	Verify Unicode support	RFC3987
H-USS18	IRI with path of permitted Unicode from base multilingual plane and '?' query of permitted including private Unicode from base multilingual plane	None	Verify Unicode support	RFC3987
H-USS19	IRI with path of permitted Unicode from base multilingual plane and '?' query including non-permitted Unicode from base multilingual plane	Reject	Verify Unicode support	RFC3987
H-USS20	IRI with path of permitted Unicode from base multilingual plane and '?' query of permitted Unicode from base multilingual plane containing permitted Bidi text	None	Verify Unicode support	RFC3987

H-USS21	IRI with path of permitted Unicode from base multilingual plane and '?' query of permitted Unicode from base multilingual plane containing non-permitted Bidi text	Reject	Verify Unicode support	RFC3987
H-USS22	IRI with path of permitted Unicode from base multilingual plane and '?' query of permitted Unicode from supplementary multilingual plane	None	Verify Unicode support	RFC3987
H-USS23	IRI with path of permitted Unicode from base multilingual plane and '?' query including non-permitted Unicode from supplementary multilingual plane	Reject	Verify Unicode support	RFC3987

H-UD: URL (IRI): decompose into components

Decompose IRIs into scheme, username, host, port, path, query and fragment. Tests for this function do not test for domain name validity.

General tests:

Test ID	Input: IRI comprising the following	Expected error	Test purpose	Reference
H-UDG1	Plain ASCII IRI	None	Verify basic support	RFC3987
H-UDG2	Plain ASCII IRI, plain ASCII domain with >3 char TLD	None	Verify long TLDs are handled	RFC3987

H-UDG3	IRI with domain and path in Unicode from base multilingual plane	None	Verifying Unicode support	RFC6531
H-UDG4	IRI with domain and path in Unicode from base multilingual plane containing Bidi text	None	Verifying Unicode support	RFC6531
H-UDG5	IRI with domain and path in Unicode from supplementary multilingual plane	None	Verifying Unicode support	RFC6531

Specific tests:

Test ID	Input: IRI comprising the following	Expected error	Test purpose	Reference
H-UDS1	Plain ASCII IRI without scheme	Reject	Verifying basic support	RFC6531
H-UDS2	Plain ASCII IRI with username	None	Verifying basic support	RFC6531
H-UDS3	Plain ASCII IRI with port	None	Verifying basic support	RFC6531
H-UDS4	Plain ASCII IRI with username and port	None	Verifying basic support	RFC6531
H-UDS5	IRI with Unicode scheme	Reject	Verifying Unicode support	RFC6531
H-UDS6	IRI with port and with username and domain in Unicode from base multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS7	IRI with port and with username and domain in Unicode from supplementary multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS8	IRI with domain and path and '#' fragment in Unicode from base multilingual plane	None	Verifying Unicode support	RFC6531

H-UDS9	IRI with domain and path and '#' fragment in Unicode from base multilingual plane containing permitted Bidi text	None	Verifying Unicode support	RFC6531
H-UDS10	IRI with domain and path and '?' query in Unicode from base multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS11	IRI with domain and path and '?' query in Unicode from base multilingual plane containing permitted Bidi text	None	Verifying Unicode support	RFC6531
H-UDS11	IRI with domain, path, '#' fragment and '?' query in Unicode from base multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS12	IRI with domain, path, '#' fragment and '?' query in Unicode from base multilingual plane, all containing permitted Bidi text	None	Verifying Unicode support	RFC6531
H-UDS13	IRI with domain and path and '#' fragment in Unicode from supplementary multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS14	IRI with domain and path and '?' query in Unicode from supplementary multilingual plane	None	Verifying Unicode support	RFC6531
H-UDS15	IRI with domain, path, '#' fragment and '?' query in Unicode from supplementary multilingual plane	None	Verifying Unicode support	RFC6531