Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 9, 2019

S. Leonard
Penango, Inc.
J. Hildebrand
Cisco Systems
T. Hansen
AT&T Laboratories

November 05, 2018

Regular Expressions for Internet Mail
draft-seantek-mail-regexen-03

Abstract

Internet Mail identifiers are used ubiquitously throughout computing systems as building blocks of online identity. Unfortunately, incomplete understandings of the syntaxes of these identifiers has led to interoperability problems and poor user experiences. Many users use specific characters in their addresses that are not properly accepted on various systems. This document prescribes normative regular expression (regex) patterns for all Internet-connected systems to use when validating or parsing Internet Mail identifiers, with special attention to regular expressions that work with popular languages and platforms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 9, 2019.

Copyright Notice

Table of Contents

1. Introduction

Internet Mail is everywhere. This fact of modern connected life is so self-evident that [RFC5598] says: "In practical terms, an email address string has become the common identifier for representing online identity." [MTECHDEV] acknowledges that email "has been one of the major technical and sociological developments of the past 40 years." Whether it is joining a social network, participating in a forum, blogging, paying taxes, buying products, conducting professional correspondence, or communicating with loved ones, one's email address forms the cornerstone or backstop (frequently both) to these methods of communication. Internet mail is not only ubiquitous: it is essentially free for all users connected to the Internet.

Yet it is surprising how fragile or cavalier many systems are with their treatment of Internet Mail identifiers, namely with email addresses. Prominent government agencies, financial institutions, and even major mail services reject a variety of forms that are in wide use by many user communities. [[NB: do a survey.]] For example, in the intervening time between IETF 94 and the submission of this Internet-Draft, the author interacted with not less than 25 different web or other Internet-connected services that rejected or mangled his perfectly valid email addresses. The result is a pernicious and creeping degradation of mail service and of the usability of the Internet Mail infrastructure, resulting in undelivered mail, misdelivered mail (which can constitute a security vulnerability), and denial of service.

The Internet Mail standards, like the mail system, have evolved over time and have been modified to accommodate volumes and scenarios far beyond their original design goals. Furthermore, some identifier forms have been restricted over time as certain syntaxes were determined to be harmful, arcane, or just plain useless. [[So while not "blame", some responsibility or causation lies with these standards, which go out of their way to balance backwards-compatibility, complexity, completeness, and flexibility at the expense of a simple and widely-implementable addressing format.]]

This document prescribes normative regular expression (regex) patterns for all Internet-connected systems to use when validating or parsing Internet Mail identifiers. Attention specifically focuses on "email address" (the specification for the string commonly associated

with a single mailbox at a single named entity), and Message-ID, which share nearly identical syntax, but have different use cases and semantics. First, the history of Internet Mail is traced to build a coherent data model for Internet Mail identifiers. Second, relevant expression formats are discussed. Third, expressions to fit the identifiers in a variety of computing contexts are developed and presented. The overall goal of this document is to establish cut-and-dried algorithms that developers can incorporate directly into their mail-using products (including web browsers, form validators, and software libraries), replacing current ad-hoc (and oftentimes atrociously inconsistent) approaches with standardized behavior.

1.1. Normative Effects

This document is proposed with either Informational or Best Current Practice status [RFC1818] for all users and systems of Internet Mail identifiers, which basically means everyone connected to the Internet, other than the mail infrastructure itself. The Internet Mail infrastructure has been standardized (and continues to be standardized by) other documents, most notably [RFC5321] and [RFC5322]. Therefore, implementers developing mail systems MUST rely on those standards when building interoperable mail systems. At the same time, the text of this specification has been [[NB: will be]] carefully vetted by [[the IETF]] so that implementers SHALL be able to rely on it as a normative reference. Whether designing a new standard or implementing a new system that uses Internet Mail identifiers for some other purpose (e.g., as usernames, security principals, or keys in a database), relying parties can "copy-and-paste" the expressions in this document to parse, validate, compose, and process Internet Mail identifiers, rather than relying on homegrown solutions.

Internet Mail has evolved over forty years, and will undoubtedly continue to evolve over time. This document does not constrain that development process. Actually, it is expected that expressions in this document will be updated to match changes in Internet Mail.

1.2. Definitions

The terms "email address" and "address" (without qualification) refer to the string commonly associated with a single mailbox at a single named entity. In this document, the prose text always qualifies these terms with the source document when using a different sense.

The term "Message-ID" (without qualification) refers to the globally unique string comprised of a left part, the at-sign (@), and a right-part, that is used to identify a single message. In this document, the prose-text always qualifies this term when using a different

sense. The term "Message-ID field", for example, refers to the Message-ID Header Field, which includes the characters "Message-ID:" and the surrounding "<" and ">" angle brackets.

Unquoted all-capital symbols in prose text have the meanings specified in [I-D.seantek-abnf-more-core-rules].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document provides expressions that can be used directly in popular computing platforms. An important subset of email address syntaxes, namely deliverable email addresses, can be described in a regular language [[CITE]]. A regular language is a language recognized by (and computable with) a finite automaton. [[CITE: Kleene's Theorem]] However, the full syntax of email addresses requires a context-free language (i.e., governed by ABNF) and a pushdown automaton.

The term "regular expression" is a sequence of characters that define a search pattern for string matching. Originally the term referred to expressions that described regular languages in formal language theory. Formal regular expressions are limited to:

o empty set and empty string o literal character o concatenation o alternation o repetition (Kleene star)

Modern-day libraries support expressions that far exceed the regular languages. Some libraries even support capabilities that exceed the context-free languages. However, this document limits itself to truly regular grammars where possible, and where not possible, to context-free grammars. Implementers can therefore implement (or compile) these specifications on computing-constrained devices.

The regular expressions in this document are intended to conform to the following de-jure or de-facto standards. Where expressions are given, they are annotated with single characters that refer to the standards to which they conform. [[NB: or, are intended to conform to, after further development.]]

1	4	Title	Ref
P	PCRE:	Perl Compatible Regular Expressions 2 (version 10.21, January 12, 2016)	[PCRE2]
E	(P)ERE:	POSIX Extended Regular Expressions (POSIX/IEEE Std 1003.1, Issue 7, 2013 Ed.) [[NB: could be ERE, PERE, or P-ERE]]	[POS-ERE]
J	JSRE:	JavaScript Regular Expressions (ECMAScript/ECMA-262, 6th Ed., 2015)	[JSRE6ED]

[[TODO: Need to do something with UTS #18: Unicode Regular Expressions.]]

Implementers should exercising caution when using a library that claims to be "Perl-Compatible" without actually being the bona-fide PCRE library: it may exhibit different or incomplete behavior. Implementers should also note that ERE and JSRE are fully implemented as alternative grammars in the std::regex library of C++11 and its successor, C++14. [[TODO: cite C++ standards.]] In the absence of a "live" regular expression library, the expressions in this document are easily compiled into automata (i.e., target language code) using well-studied algorithms.

Surrounding delimiters (i.e., slashes) are omitted unless relevant to the proffered usage.

2. History and Formal Models for Internet Mail Identifiers

2.1. The Core History

Internet Mail (also known as electronic mail, e-mail, email, or simply "mail") is an asynchronous, store-and-forward method of exchanging digital messages from an author to one or more recipients. [MTECHDEV] recounts the technical development of email, which is too voluminous to be repeated here. This section specifically focuses on the history of the identifiers used in email.

When people think of an email address, <user@example.com> is the quintessential example. However, addresses did not always look this way. Electronic mailing systems come from the earliest days of networking, before ARPANET. The specification for identifiers was defined by the networking project on a more-or-less ad-hoc basis. When ARPANET began, mail and file transfer were seen as important

founding services. In fact, FTP was a significant transport mechanism for mail.

By 1973, users of ARPANET came together to propose [RFC0524] and standardize [RFC0561] parts of the mail system. In these early specifications, the at-sign "@" was not used; instead, the term "AT" separated the left-side (user production) from the right-side (host production). Tokens (the word production, specifically) during that time were separated by SP, CR, and LF. The word production was defined at that time to be any ASCII character other than CR, LF, and SP. [RFC0524] only standardized the From header, not any recipient headers.

In 1975, [RFC0680] proposed a more formal format for for message fields beyond [RFC0561], including "receiver specification fields" (TO, CC, and BCC) and "reference specification fields" (MESSAGE-ID, IN-REPLY-TO, REFERENCES, and KEYWORDS) for the first time. The receiver fields used mailbox productions with user and host parts separated by "@" (without spaces), in contrast to the originator specification fields (specifically FROM and SENDER), which continued to use "AT" (with single spaces on either side). An [RFC0680] Message-ID was structured with a "Net Address" (presumably, a host name) in square brackets, followed by a line production (any characters other than CR or LF).

The first real email standards that resemble modern usage were published in 1977: [RFC0724] and its revision, [RFC0733]. Those documents are historic and their formats are incompatible with most modern mail systems, but understanding them provide important insights into the structure of identifiers starting with [RFC0821] and [RFC0822]. The "@" character gained greater prominence, although "AT" (then lowercased to "at" in the specifications) was still supported. Importantly, [RFC0733] Message-ID was standardized to match the [RFC0733] email address format, and a uniqueness guarantee was added: "The uniqueness of the message identifier is guaranteed by the host which generates it." Essentially, this text implies that the right-hand part of the Message-ID is to be a hostname, or operatively associated with a hostname (i.e., not just a random string, but a unique-possibly random-string assigned by a host). At this point, Message-ID and email address specifications converged in the host-phrase production.

[RFC0821] and [RFC0822] are the foundational RFCs for modern mail usage, and their revisions over the years retain the same basic structure and division between mail transfer (specifically, SMTP) and mail format. Jon Postel's invention of SMTP grew out of experience and disenchantment with the Mail Transfer Protocol (MTP) [RFC0772]. The Simple Mail Transfer Protocol is indeed simple: it is structured

like FTP but only has a limited set of commands: HELO, MAIL FROM, RCPT TO, DATA, QUIT, VRFY, and EXPN (and a few others not relevant for this discussion). The commands can take at most one argument. [RFC0821] describes a forward-path argument rather than an email address argument: the distinction is that after the username, a series of "@" host specifications designate the hops through which the message is supposed to travel before it reaches its destination.

The main [RFC0821] production for an email address is the <mailbox>, which is defined as <local-part> "@" <domain>. The <local-part> production can be a <dot-string> or a <quoted-string>. In both cases, the full range of ASCII characters is actually permitted, although different characters must be backslash-escaped in different productions. The <domain> production is extremely broad and can "stack" domain-element components with the period "."; a domain-element component can be a hostname, "#" and a number, or an IPv4 address enclosed in square brackets "[" and "]".

[RFC0822] introduced the "addr-spec" ABNF production, which is that series' term for an email address. While a [RFC0822] route-addr production can include a source route (aka forward-path with multiple hosts), addr-spec is noted to be global address, with the right side being a "domain" production. This definition presaged the first DNS standards [RFC0882] and [RFC0883], although it was clearly designed with DNS in mind. The [RFC0822] ABNF permits a domain to include multiple domain-literal productions (i.e., bracketed) separated by "."; however, the accompanying text basically obviates such productions. As [RFC0822] presaged the widespread implementation of DNS, various systems would spread routing information between the local-part and domain productions (see Section 6.2.5). [RFC0822] discusses local-part syntax extensively, including examples of comment productions that are supposed to be ignored semantically (see Section A.1).

Section 3.4.7 of [RFC0822] describes the constituent components of an address as requiring "preservation of case information", which is slightly different than saying "case-sensitive" outright (although the latter is strongly implied). The main historical point to glean is that intermediate mail systems were supposed to transit the local-part AS-IS without modification, so that the destination system--and only the destination system--would parse it.

[RFC0822] assigns specific semantics to Message-ID but is light on syntax: the msg-id production is just addr-spec enclosed in mandatory "<" and ">".

The widespread deployment of DNS [RFC0973] (later [RFC1034] and [RFC1035], with [RFC1123] relaxation) dramatically changed the

Internet messaging landscape of the late 1980s. [[TODO: complete.]] With respect to email addresses and Message-IDs, it became obvious that the right-hand side of the "@" was supposed to represent a fully-qualified domain name at which Mail eXchanger records are located [[TODO: cite]]. Other networks and host-naming formats became obsolete by the mid-1990s.

The 2001 standards [RFC2821] and [RFC2822] reflect the explosive growth and diverse deployments of Internet mail. The work was undertaken in the DRUMS (Detailed Revision/Update of Message Standards) working group between 1995 and 2001. [RFC2822] introduced the "obs-" prefix in its ABNF, along with Section 4, Obsolete Syntax. Essentially, [RFC2822] prescribes a generation format that is much stricter than the parsing format, but still demands that conforming implementations understand the parsing format. Therefore, the U+0000 NULL character that is considered obsolete in [RFC2822] must still be considered part of the data model of email addresses. The syntax of [RFC2821] is tighter than [RFC2822], and a careful read makes it apparent that the underlying address formats diverged in the intervening years. Specifically, [RFC2822] is much more concerned with historical forms than [RFC2821], which is more about contemporary transmission behavior. At the same time, [RFC2821] does not actually prohibit a wide range of C0 control characters, which still remain part of [RFC2821]'s data model. [[TODO: complete history of RFC 282x.]]

Revisions to the base mail standards most recently completed, [RFC5321] and [RFC5322], were worked on between 2005 and 2008. Email addresses saw further character restrictions, namely around the entire range of C0 control characters, which [RFC5321] explicitly prohibits. [[TODO: complete history of RFC 532x.]]

2.2. Multipurpose Internet Mail Extensions and Uniform Resource Identifiers

[[Message-ID and Content-ID. Therefore this document applies with equal normative force to Content-IDs, and to mid: and cid: URIs that use them.]]

2.3. Email Address Internationalization

As the Internet became a ubiquitous feature of modern life, and as email followed it, users in various countries called for identifiers that were usable in their native languages. The IETF [[worked on this]] in the email address internationalization (EAI) effort, culminating in [RFC6530], [RFC6531], and [RFC6532]. The key changes for identifiers were to expand the character repertoire of email addresses, and anything looking like email addresses (e.g., Message-

ID), to include UTF-8 octets. As UTF-8 can encode any Unicode scalar value (but not any Unicode code point), the practical result is that addresses and Message-IDs can contain (almost) any Unicode scalar value.

A practical result of EAI combined with MIME and MIME URIs, is that MIME URIs now are UTF-8 encoded in the pct-encoded production.

2.4. The Data Model

Parties that rely on this document SHALL interpret the semantically meaningful parts of Internet Mail identifiers as follows.

2.4.1. Email Address

An email address is comprised of a local-part and a domain, separated by "@". The parsed local-part is a sequence of Unicode scalar values, and can be represented by any well-formed Unicode encoding. [[NB: The following may be CONTROVERSIAL!!]] The parsed domain is a sequence of restricted Unicode scalar values that represent some identifier for some host on the network. The parsed domain can be:

1. a hostname of any kind, including (but not limited to) a DNS name;

2. an IPv4 address literal;

3. an IPv6 address literal;

4. an address literal, comprised of a standardized tag and a sequence of ASCII characters.

Conveniently, the domain string subtypes can be combined into a single well-formed Unicode string, discriminated as follows:

1. If the string begins with "IPv6:", it is a type 3 IPv6 address, and the remainder had better be a valid IPv6 address in textual form.

2. If the string begins with Ldh-str and a colon ":", it is a type 4 address literal, and the remainder had better be dcontent (which notably is not supposed to contain characters beyond ASCII).

3. If the string has four sets of digits 0-255 separated by dots, then it is an IPv4 address.

4. Otherwise, it had better be a domain name (i.e., comprised of NR-LDH labels and U-labels, separated by dots). [[NB: [RFC1912]

says a label can't be all-numeric, but then it catalogs some exceptions.]]

5. Finally, it is "some random Unicode string" that is syntactically valid under the most expansive rules, but is not useful for delivering or reporting on Internet mail.

2.4.2. Message-ID

A Message-ID is comprised of a left part (id-left) and a right part (id-right), separated by "@". The id-left is a sequence of Unicode scalar values, and can be represented by any well-formed Unicode encoding. [[NB: The following may be CONTROVERSIAL!!]] The id-right is a sequence of Unicode scalar values, and can be represented by ay well-formed Unicode encoding.

(A Content-ID [RFC2045] has the same composition as a Message-ID.)

2.5. Equivalence and Comparison

2.5.1. Email Address

Two email addresses are equivalent if the parsed local-part and parsed domain values are equivalent.

Two parsed local-parts are equivalent if their Unicode scalar values are equal.

The special values "postmaster", "abuse", and [TODO: fill out] SHALL be compared case-insensitively [TODO: full-width characters? Unicode case folding? etc.].

[Case sensitivity] In all other cases, two parsed local-parts may be equivalent if the [TODO: receiving MTA] delivers mail addressed to them to the same mailbox. There is no algorithmic comparison to determine said equivalence.

Two parsed domains are equivalent if both have the same type, and the values are equivalent. Additionally, an IPv6 address literal is equal to an IPv4 address literal when the IPv6 address is an "IPv4-mapped IPv6 address", and its IPv4 component equals the IPv4 address of the IPv4 address literal.

1. hostnames (domain names): equal [TODO: RFC-REF, IDNA, RFC1034, RFC1035, etc.].

2. IPv4 addresses: equal [TODO: RFC-REF].

3. IPv6 addresses: equal [TODO: RFC-REF].

4. address literal: the standardized tag is equal (case-insensitive), and the address value is octet-for-octet equal (case-sensitive), or is equal per the rules standardized by the standardized tag registration.

2.5.1.1. Case sensitivity of local-part

Of all equivalence issues, no issue generates more confusion and dissent in the email community than the case sensitivity of the local-part. Formally local-part is case-sensitive. A significant fraction of installed mail servers treat local-part as case-insensitive in the ASCII range. (At the time of this writing, EAI has not been implemented widely enough to make statements about case insensitivity for characters beyond ASCII.) [[TODO: survey and statistics.]] Furthermore, many systems [[TODO: quantify]] outside of the Internet Mail infrastructure compare the local-part of email addresses case-insensitively.

Historically, local-parts were case-sensitive because of MULTICS. However, as time went on they became case-preserving: receiving systems would not register additional mailbox names (i.e., local-parts) if the proposed mailbox name differed from an existing mailbox name only by case.

[[TODO: develop recommendation about how wise or unwise it is to go one way or another.]] One possible approach is to define an additional output, "conditionally equivalent" of the equivalence algorithm. Therefore an implementation conforming to this document SHALL output one of three states: equivalent, not equivalent, and "conditionally equivalent", that is, equivalent if and only if local-part is compared case-insensitively in the ASCII range [[TODO: should we say in the full Unicode range?]]. Applications implementing this algorithm SHOULD NOT treat such a state as "equivalent". For example, a user-facing application SHOULD treat this state as a "warning" that requires further intervention.

In modern times, email addresses tend to be emitted [[TODO: statistics]] in all-lowercase, when case is normalized. Therefore, applications implementing this algorithm [[TODO: document]] that are aware that the receiving MTA is case-insensitive, as well as applications implementing this algorithm that receive input that is case-ambiguous (such as voice input), SHOULD record the local-part in all-lowercase unless presented with evidence to the contrary.

The EAI standards (RFCs 6530-6532) make no mention of case sensitivity issues for characters beyond the ASCII range. Permitting

Unicode scalar values (i.e., UTF-8) opens up a whole range of comparison issues with potentially far-reaching identity and security implications. [[TODO: discuss case preservation and sensitivity issues in characters beyond the ASCII range. The bottom line is, use what you're given, don't mess with it, hope for the best.]]

2.5.2. Message-ID

Two Message-IDs are equivalent if the parsed id-left and parsed id-right values are equivalent.

Two parsed id-left values are equivalent if their Unicode scalar values are equal.

[[NB: controversial!]] Two parsed id-right values are equivalent if their Unicode scalar values are equal.

[[TODO: The case sensitivity of id-right has not been fully explored in any standard to-date. To the extent that id-right represents a domain name, there is a strong argument to treat id-right as case-insensitive in the ASCII range. Standardized tags are probably case-insensitive based on the ABNF of [RFC5321] relating to "IPv6". The rest is kind of up for grabs. The bottom line is, if you intend to match the Message-ID of an existing message, don't take chances: just copy it verbatim into the destination.]]

(Two Content-IDs are equivalent under the same rules. However, a Content-ID and a Message-ID are never equal to each other, and if such a thing occurs, it is not correct because both Content-ID and Message-ID are supposed to be "world-unique" [RFC2045] [RFC5322].)

3. Regular Expressions for Email Addresses

[[Valid email address vs. deliverable email address]] A "deliverable email address" complies with the modern production rules of [RFC5321] and [RFC6531]. A deliverable email address SHALL have a domain part that is a domain name (Section 2.3.5 of [RFC5321]); it SHALL NOT have a domain part that is an address literal (Section 4.1.3 of [RFC5321]) or a host name that does not comply with domain name rules (see [RFC1034], [RFC1035], [RFC5890] et. seq.). [[TODO: justify. Basically experiments have borne out that many mail systems will not accept user@[3.3.3.3] as a RCPT TO. Technically it is valid per RFC 5321, but practically any receiving MTA that handles more than one MX/domain will have difficulty in figuring out what domain-specific mailbox to which to deliver the mail. The author tried a couple of popular MTAs.]] Systems that use email addresses with an expectation of SMTP delivery SHALL accept productions that comply with this document.

Email addresses that do not meet these modern production rules may nevertheless be valid under the other modern (e.g., [RFC5322]) or legacy (e.g., [RFC0821] and [RFC0822]) production rules.

Systems that recognize "modern email addresses" in new corpa (e.g., in text editors) SHALL accept productions that comply with this document.

Systems that recognize "legacy email addresses" in existing corpa (e.g., in email messages or documents predating this document) SHALL accept productions that comply with this document.

3.1. Deliverable Email Address

A deliverable email address is an email address that can be used to deliver messages over the modern SMTP infrastructure. This has important implications for the domain part, because the domain part MUST represent a domain name that complies with contemporary rules and regulations, such as [RFC5890].

3.1.1. ASCII Building Blocks

The following rules are amalgamated from the SMTP and Internet message format standards [RFC5321] [RFC5322]. All expressions are PCRE2-compatible.

```
(?(DEFINE)
 (?#local)
 (?<atext>[0-9A-Za-z!#-'*+\-/=?^_`{-~])
 (?<dot_string>(?&atext)+(?:\.(?&atext)+)*)
 (?<qtext>[ !#-\[\]-~])
 (?<quoted_pair>\\[ -~])
 (?<qcontent>(?&qtext)|(?&quoted_pair))
 (?<quoted_string>"(?&qcontent)*")
 (?<local_part>(?&dot_string)|(?&quoted_string))
 (?#domain)
 (?<oct>0*(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]?))
 (?<IPv4>(?&oct)(?:\.(?&oct)){0,3})
 (?<sub_domain>[0-9A-Za-z](?:[\-0-9A-Za-z]{0,61}[0-9A-Za-z])?)
 (?<domain>(?!(?&IPv4)[^\-.0-9A-Za-z])(?&sub_domain)
  (?:\\.(?&sub_domain))*(?![\-.0-9A-Za-z]))
)
```

3.1.2. Deliverable Email Address

A deliverable email address matches the <Mailbox> production of [RFC5321].

```
(?&local_part)@(?&domain)

(?&local_part)@(?&domain)(?<![\-.0-9A-Za-z]{254})

^(?&local_part)@(?&domain)$

^(?&local_part)@(?&domain)(?<![\-.0-9A-Za-z]{254})$
```

The aforementioned patterns take into account the limitations of DNS [RFC1034], namely, that a label can only have 63 characters, and that a domain name can only have 253 characters.

The negative-lookbehind on the domain (with {254}) essentially ensures that the "@" symbol is present no earlier than 253 preceding characters: PCRE2 cannot process variable-length lookbehinds.

The repeater {0,61} in <sub_domain> is not sufficient to ensure a safe <domain> production, because what will happen is that the domain production might end after 63 consumed characters in a putative sub-domain, leaving overlong characters that are still part of the putative sub-domain hanging at the end. (This is not an issue when the entire string is an address, in which case $ definitively terminates the string.) The way to deal with this it the negative-lookahead on the end of <domain> that fails to match when additional domain characters are present. This negative-lookahead also deals with some unpleasant corner cases when the <domain> is a partial IPv4 address.

The aforementioned patterns also take into account [RFC1912], namely, "[l]abels may not be all numbers". Actually the way that software works is that if the putative domain name can be parsed as an IPv4 address, then it is treated as an IPv4 address; otherwise, it is treated as a DNS name. Therefore, character sequences that are valid IPv4 addresses need to be restricted out. For example: "1.3.4.255" is invalid; but, "1.3.4.256" and "1.3.4.255.1" are valid, because they do not parse to IPv4 addresses. (411.org is valid for obvious reasons.)

Borderline cases are partial IPv4 addresses, such as "411" (could be 0.0.1.155) and "1.411" (could be 1.0.1.155). The regular expressions above accept these domain productions, but they may not be safe. The regular expressions above also accept hex form, such as "0xef" (could be 0.0.0.239).

3.1.3. (Leftover from draft-00) Basic Rules of Derivation (Unicode)

The following rules are amalgamated from the SMTP standards [RFC5321] and [RFC6531], the foundational DNS standards [RFC1034], [RFC1035], and [RFC1123], and the modern IDNA standards [RFC5890], [RFC5891], and [RFC5892].

[[NB: The syntax below assumes that Perl Compatible Regular Expressions 2 [PCRE2] is being used, such that \xnn and \x{...} refer to valid Unicode scalar values, i.e., well-formed UTF-8 sequences. Specifically, the surrogate range U+D800-U+DFFF is omitted. Although listed as JSRE and ERE-compatible, these expressions will need to be massaged somewhat to handle the Unicode-referencing differences.]]

[[TODO: There need to be two forms: a form that matches a complete string, so the regular expression can start with ^ and end with $. This will make the execution pretty fast. A second form can match any string in a block of text. This will be much more intensive because ^ and $ cannot be used; instead the boundary could be delimited by any of a HUGE yet noncontiguous quantity of characters beyond ASCII, such as fullwidth punctuation, spaces of various kinds, etc.]]

```
P E J atext = [A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}]

P E J dot-string = [A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}]+
                   (?:\.[A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}])

P E J qtext = [ !#-\[\]-~\xA0-\x{10FFFF}]

P E J quoted-pair = \\[ -~]

P E J qcontent = [ !#-\[\]-~\xA0-\x{10FFFF}]|\\[ -~]

P E J quoted-string = "(?:[ !#-\[\]-~\xA0-\x{10FFFF}]|\\[ -~])*"

P E J local-part = [A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}]+
             (?:\.[A-Za-z0-9!#-'*+-/=?^_`{-~\xA0-\x{10FFFF}])|
             "(?:[ !#-\[\]-~\xA0-\x{10FFFF}]|\\[ -~])*"

; RFC 5890, must contain at least one non-ASCII character
; TODO: express other constraints such as in Protocol document [RFC5891]
; and Tables document [RFC5892]
; WARNING: intentional omission: dot . is included in this production--
; it should not be
      U-label = [[TOO COMPLEX TO DO RIGHT NOW...]]
P E J U-label = [\x00-\x{10FFFF}]*[\x80-\x{10FFFF}]+[\x00-\x{10FFFF}]*

P E J sub-domain = [A-Za-z0-9](?:[A-Za-z0-9\-]*[A-Za-z0-9])?|
                   [\x00-\x{10FFFF}]*[\x80-\x{10FFFF}]+
                   [\x00-\x{10FFFF}]*

P E J domain = (?:[A-Za-z0-9](?:[A-Za-z0-9\-]*[A-Za-z0-9])?|
               [\x00-\x{10FFFF}]*[\x80-\x{10FFFF}]+[\x00-\x{10FFFF}]*)
               (?:\.(?:[A-Za-z0-9](?:[A-Za-z0-9\-]*
               [A-Za-z0-9])?|[\x00-\x{10FFFF}]*
               [\x80-\x{10FFFF}]+[\x00-\x{10FFFF}]*))*
```

3.1.4. Complete Expression for Deliverable Email Address

The following regular expression is a deliverable email address:

```
; Mailbox from RFC 5321, as amended
P E J DEA     = ([A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}]+
                 (?:\.[A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}])|
                 "(?:[ !#-\[\]-~\xA0-\x{10FFFF}]|\\[ -~])*")@
                ((?:[A-Za-z0-9](?:[A-Za-z0-9\-]*[A-Za-z0-9])?|
                 [\x00-\x{10FFFF}]*[\x80-\x{10FFFF}]+[\x00-\x{10FFFF}]*)
                 (?:\.(?:[A-Za-z0-9](?:[A-Za-z0-9\-]*
                 [A-Za-z0-9])?|[\x00-\x{10FFFF}]*
                 [\x80-\x{10FFFF}]+[\x00-\x{10FFFF}]*))*)
```

In the regular expression DEA, capturing group 1 is the local-part production, and capturing group 2 is the domain production.

3.1.5. Using Character Classes

[[TODO: provide expressions that use character classes, and explain the benefits and tradeoffs.]]

3.1.6. "Flotsam" and "Jetsam" Beyond ASCII

As mail usage is international in scope, modern mail and mail identifier-using systems MUST support Unicode EAI identifiers. Unfortunately, rigorously following the EAI specifications [RFC6530], [RFC6531], and [RFC6532] will lead to (possibly) unforeseen text parsing problems, where naive (or strictly conforming) parsers will tend both to overconsume and underconsume non-ASCII text surrounding an otherwise "obvious" e-mail address. The problem is described in this section, while the following Section 3.1.3 provides at least some partial mitigations.

The right-hand side of a deliverable email address is a domain name. A conforming parser may well overconsume text on the right-hand side, aka "jetsam", that cannot possibly be in a domain name, such as non-ASCII punctuation, spaces, control characters, and noncharacter code points. On the left-hand side (local-part), for better or for worse, the atext production of local-part has been extended in both [RFC6531] and [RFC6532] to accept any Unicode character beyond the ASCII range. Therefore, a whole slew of "flotsam" can get validly prepended to an internationalized email address. Apparently the only way to "stop" this is to quote the local-part.

Characters will get inconsistent treatment depending on which end the characters appear. For example: the characters U+FF1C FULLWIDTH LESS-THAN SIGN and U+FF1E FULLWIDTH GREATER-THAN SIGN are classified as [Sm] aka "Symbol, Math", which are DISALLOWED under [RFC5892]. Therefore the presence of U+FF1E on the domain end will terminate the address (for a properly implemented regular expression), but the corresponding presence of U+FF1C on the local-part end will NOT terminate the address! A conforming parser would actually start much earlier in a blob of text (possibly as early as the beginning of a new line) and match all characters up to the @ delimiter, blowing straight through U+FF1C.

3.1.7. Certain Expressions for Restrictions

Email addresses incorporate internationalized domain names, so the complex and confusing rules of IDNs apply directly to the right-hand side of deliverable email addresses.

[[TODO: integrate these expressions into the main expressions of 3.1.2.]] The following expressions apply to an individual sub-domain production:

[[NB: open issue if length should be restricted. Author believes it should be length-restricted, because overlong labels in domain names mean the address can't be looked up, and therefore, the address is not deliverable.]]

The following lookahead regular expressions apply on a per-label (i.e., per-sub-domain) basis.

P E J (?=[^.]{1,63}\.|$)
: restricts to 63 characters

P E J (?=[0-z]{1,63}\.|$)
: restricts to 63 LDH characters

P E J (?=[0-z\xA0-\x{10FFFF}]{1,56}\.|$)(?!..--)
(?=[0-z\xA0-\x{10FFFF}]{0,55}[\xA0-\x{10FFFF}][0-z]{0,55}\.|$)
or
(?=[0-z\xA0-\x{10FFFF}]{1,56}\.|$)(?!..--)(?![0-z]{1,56}\.|$)
: Enforces that for U-labels (where at least one non-ASCII char is present), there cannot be more than 55 chars. The equivalent ACE will include xn-- PLUS -2rh, which means 7 extra characters (8 chars minus 1 Unicode char). Did a test with U+00DE. Also not allowed to have any any HYPHEN HYPHEN (Section 4.2.3.1 of [RFC5891]).

[[NB: Should we permit fullwidth and other dots (not just ASCII dot)?]]

P E J (?![0-9]{1,63}\.|$)
: restricts out all-numeric labels [RFC1912]
[[TODO: Would it be more accurate to say that the all-numeric labels 0-255 are prohibited, but 256+ are permissible?]]

[[NB: The end-of-address production \.|$ must also include the possibility of any number of non-domain name characters, when searching through an arbitrary block of text.]]

[[TODO: the flotsam on the local-part end is potentially a big problem, unless we say that deliverable email addresses MUST be delimited on the left-hand side by the ASCII character < or other well-established characters that are not in dot-atom-text/dot-string.]]

3.1.8. Unquoting Local-Part

The following regular expressions can be used to unquote the local-part production.

```
P sed s/^"(.*)"$/$1/
        Applies when the local-part is isolated from @domain.
        Removes surrounding quotations.

P sed s/\\(.)/$1/g
        Applies when the local-part is isolated from the surrounding
        quotations. (Safe to use with dot-string aka dot-atom-text,
        since backslashes are not present.) Unquotes quoted-pairs.

P   J /^"|(?=.+"$)\\(.)|"$/g  (with "$1" replacement string)
P sed s/^"|(?=.+"$)\\(.)|"$/$1/g
        Applies when local-part is isolated from @domain.
        Removes surrounding quotations and unquotes quoted-pairs
        in one loop by using lookahead.
```

[[TODO: add more detailed descriptions of operations.]]

3.1.9. Quoting Local-Part

Given a Unicode string that represents the unquoted local-part, the following regular expressions can be used to create a quoted production.

```
J /(?=[^"\]*["\])(["\])/\$1/g
        Quote each and every " and \.
J /^(?![A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}]+
      (?:\.[A-Za-z0-9!#-'*+\-/=?^_`{-~\xA0-\x{10FFFF}])$).*$/"$&"/
        If the string does not conform to dot-string (including,
        e.g., the presence of consecutive dots), surround the
        entire string with quotations.
```

3.2. Modern Email Address

A modern email address is an email address that conforms to [RFC5322], except for the ABNF productions in that standard that are marked as "obsolete". For example, control characters are excluded. Modern email addresses are a superset of deliverable email addresses [RFC5321]. Since modern email addresses are not necessarily deliverable by SMTP, the domain production does not need to conform to DNS rules. This relaxation makes the regular expressions much simpler. On the other hand, modern email addresses permit embedded comments and folding whitespace, requiring the use of a pushdown automaton.

```
(?(DEFINE)
 (?#whitespace)
 (?<FWS>(?:[\t ]*\r\n)?[\t ]+)
 (?<CFWS>(?:(?&FWS)?(?&comment))+(?&FWS)?|(?&FWS))
 (?#local)
 (?<atext>[0-9A-Za-z!#-'*+\-/=?^_`{-~])
 (?<dot_atom_text>(?&atext)+(?:\.(?&atext)+)*)
 (?<ctext>[!-'*-\[\]-~])
 (?<ccontent>(?&ctext)|(?&quoted_pair)|(?&comment))
 (?<comment>\((?:(?&FWS)?(?&ccontent))*(?&FWS)?\))
 (?<qtext>[ !#-\[\]-~])
 (?<quoted_pair>\\[ -~])
 (?<qcontent>(?&qtext)|(?&quoted_pair))
 (?<quoted_string>(?&CFWS)?"(?:(?&FWS)?(?&qcontent))*(?&FWS)?"(?&CFWS)?)
 (?<local_part>(?&CFWS)?(?&dot_atom_text)(?&CFWS)?|(?&quoted_string))
 (?#domain)
 (?<dtext>[!-Z^-~])
 (?<domain>(?&CFWS)?(?:(?&dot_atom_text)|
  \[(?:(?&FWS)?(?&dtext))*(?&FWS)?\])(?&CFWS)?)
)
```

A modern email address matches the <addr-spec> production of [RFC5322], without any obsolete parts.

```
(?&local_part)@(?&domain)
```

```
^(?&local_part)@(?&domain)$
```

3.3. Legacy Email Address

[[TODO: expand regular expressions. Arguably, characters beyond ASCII need not be included. Therefore domain should be MUCH simpler: the name form should be restricted to LDH-strings.]]

3.4. Algorithms for Detecting Email Addresses

As Section 3.1 indicates, compiling and executing a true and correct regular expression for an email address (deliverable, valid, historic) will be complicated and time-consuming. More efficient algorithms are desirable.

- Scan text buffer for "@".
- Evaluate characters after "@" for domain production.
- Evaluate character prior to "@".

- If prior character is <">, scan backwards for initial (unescaped) <">; evaluate characters in between to match quoted-string production.

- Otherwise if prior character is a valid atext, consume characters backwards while evaluating for dot-string. (The dot-string production is palindromic.)

[[TODO: add other suggested algorithms.]]

[[Splitting valid email address into local-part and right-hand-side.]]

3.5. Handling Domain Names

[[TODO: discuss the issues with handling domain names.]]

4. Regular Expressions for Message-IDs

Message-ID values form a disjoint set from email address values, i.e., a Message-ID that also happens to be an email address is just a coincidence.

The productions that comprise Message-ID are called id-left and id-right.

4.1. Modern Message-ID

A modern Message-ID is one that complies with the strict generation rules of [RFC5322]. In particular: id-left is only dot-atom-text (as amended by [RFC6532], and id-right is dot-atom-text or no-fold-literal (as amended by [RFC6532]). Notably, virtually any Unicode scalar value is permissible in id-right, because [RFC6532] does not import U-label (unlike [RFC6531]). The resulting regular expressions will therefore be more expansive, at the cost of accepting characters, such as fullwidth punctuation, that would otherwise delimit Message-IDs on both ends in text.

The regular expressions reuse many of the subroutines of Section 3.1. [[POINTER: obs-id-left and obs-id-right are supersets of their modern forms, so deliverable email address regular expressions may well be reused directly.]]

```
(?(DEFINE)
 (?#id-left)
 (?<atext>[0-9A-Za-z!#-'*+\-/=?^_`{-~])
 (?<dot_atom_text>(?&atext)+(?:\.(?&atext)+)*)
 (?<id_left>(?&dot_atom_text))
 (?#id-right)
 (?<dtext>[!-Z^-~])
 (?<id_right>(?&dot_atom_text)|\[(?&dtext)*\])
)
```

A modern Message-ID matches the <addr-spec> production of [RFC5322], without any obsolete parts.

```
(?&id_left)@(?&id_right)
```

```
^(?&id_left)@(?&id_right)$
```

4.2. General Message-ID

A "general Message-ID" is one that complies with any of the mail rules.

[[TODO: complete.]]

5. Security Considerations

Internet Mail identifiers are important for identifying users and other principals in security systems. While a user's login token and an email address are formally separate entities, many common Internet-connected systems conflate the two. Systems that accept email addresses as login tokens (in particular, other systems' email addresses, rather than their own) SHALL accept the full range of valid email addresses. To do otherwise is to act as a denial of service against legitimate users with legitimate mailbox names.

When a user forgets his or her password or other login credentials, the most common recovery method on the Internet is to send a recovery message to the user's registered [[email address]]. Preventing users from using their chosen or assigned addresses acts as a denial of service.

Because a local-part can contain almost any Unicode scalar value, security issues are essentially pushed from clients to servers and to registration processes. I.e., it is up to a server implementation to decide whether to accept an arbitrary Unicode string for registration or for delivery with SMTP, folding or normalizing input at its discretion. A robust server implementation needs to handle arbitrary input gracefully.

In contrast, when a domain part represents a domain name, the string is severely restricted by the IDNA documents [RFC5890] et. seq. A server is perfectly within its rights to reject input that is not in NFC or contains disallowed characters. Specifically, since the domain part is the key to retrieving the MX resource record, the Internet Mail standards hardly get involved. These restrictions put more onus on clients to validate strings. However, integrating the entire static list of [RFC5892] into regular expressions would be unduly burdensome on many implementations. While an implementation can consider using character classes, the risks and benefits of using character classes need to be carefully considered.

Character classes represent a shorthand for certain ranges of characters based on Unicode properties. Effectively the total system's state table remains the same, but the complexity is pushed from one component (the regular expression definition) to another component (the table representing the character classes). Ultimately, the regular expression character classes in popular formulations will derive from the Unicode Standard [UNICODE] definitions such as UnicodeData.txt. Since a proper IDNA-enabled DNS library needs to keep track of these character classes anyway, referencing these ranges by character classes should not add much to the image size. However, now the regular expression (which previously was self-contained) now has an external dependency to data that may-and probably will-frequently change, including (for example) the ranges of unassigned code points. What could have been a valid email address one month may be invalid the next month, or vice-versa, simply depending on the version of the regular expression or DNS library that an implementation depends on. Implementers should therefore evaluate their own needs for security and stability in picking particular regular expression forms.

6. References

6.1. Normative References

[I-D.seantek-abnf-more-core-rules]
Leonard, S., "Comprehensive Core Rules and References for ABNF", draft-seantek-abnf-more-core-rules-07 (work in progress), September 2016.

[JSRE6ED] Ecma International, "ECMAScript 2015 Language Specification", Standard ECMA-262, 6th Edition , June 2015, <http://www.ecma-international.org/ecma-262/6.0/>.

[PCRE2] Hazel, P., "Perl Compatible Regular Expressions 2, version 10.21", January 2016, <http://www.pcre.org/>.

[POS-ERE] IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008 and IEEE Std 1003.1-2008/Cor 1-2013), ""Standard for Information Technology - Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7" (incorporating Technical Corrigendum 1), Section 9.4, "Extended Regular Expressions"", April 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html>.

[RFC0821] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, DOI 10.17487/RFC0821, August 1982, <http://www.rfc-editor.org/info/rfc821>.

[RFC0822] Crocker, D., "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES", STD 11, RFC 822, DOI 10.17487/RFC0822, August 1982, <http://www.rfc-editor.org/info/rfc822>.

[RFC0973] Mockapetris, P., "Domain system changes and observations", RFC 973, DOI 10.17487/RFC0973, January 1986, <http://www.rfc-editor.org/info/rfc973>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://www.rfc-editor.org/info/rfc1123>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2821] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, DOI 10.17487/RFC2821, April 2001, <http://www.rfc-editor.org/info/rfc2821>.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <http://www.rfc-editor.org/info/rfc2822>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <http://www.rfc-editor.org/info/rfc5321>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <http://www.rfc-editor.org/info/rfc5891>.

[RFC5892] Faltstrom, P., Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, DOI 10.17487/RFC5892, August 2010, <http://www.rfc-editor.org/info/rfc5892>.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for Internationalized Email", RFC 6530, DOI 10.17487/RFC6530, February 2012, <http://www.rfc-editor.org/info/rfc6530>.

[RFC6531] Yao, J. and W. Mao, "SMTP Extension for Internationalized Email", RFC 6531, DOI 10.17487/RFC6531, February 2012, <http://www.rfc-editor.org/info/rfc6531>.

[RFC6532] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 9.0.0", August 2016.

6.2. Informative References

[MTECHDEV] Partridge, J., "The Technical Development of Internet Email", IEEE Annals of the History of Computing, Vol. 30, No. 2, DOI 10.1109/MAHC.2008.32 , June 2008.

[RFC0524] White, J., "Proposed Mail Protocol", RFC 524, DOI 10.17487/RFC0524, June 1973, <http://www.rfc-editor.org/info/rfc524>.

[RFC0561] Bhushan, A., Pogran, K., Tomlinson, R., and J. White, "Standardizing Network Mail Headers", RFC 561, DOI 10.17487/RFC0561, September 1973, <http://www.rfc-editor.org/info/rfc561>.

[RFC0680] Myer, T. and D. Henderson, "Message Transmission Protocol", RFC 680, DOI 10.17487/RFC0680, April 1975, <http://www.rfc-editor.org/info/rfc680>.

[RFC0724] Crocker, D., Pogran, K., Vittal, J., and D. Henderson, "Proposed official standard for the format of ARPA Network messages", RFC 724, DOI 10.17487/RFC0724, May 1977, <http://www.rfc-editor.org/info/rfc724>.

[RFC0733] Crocker, D., Vittal, J., Pogran, K., and D. Henderson, "Standard for the format of ARPA network text messages", RFC 733, DOI 10.17487/RFC0733, November 1977, <http://www.rfc-editor.org/info/rfc733>.

[RFC0772] Sluizer, S. and J. Postel, "Mail Transfer Protocol", RFC 772, DOI 10.17487/RFC0772, September 1980, <http://www.rfc-editor.org/info/rfc772>.

[RFC0882] Mockapetris, P., "Domain names: Concepts and facilities", RFC 882, DOI 10.17487/RFC0882, November 1983, <http://www.rfc-editor.org/info/rfc882>.

[RFC0883] Mockapetris, P., "Domain names: Implementation specification", RFC 883, DOI 10.17487/RFC0883, November 1983, <http://www.rfc-editor.org/info/rfc883>.

[RFC1818] Postel, J., Li, T., and Y. Rekhter, "Best Current Practices", RFC 1818, DOI 10.17487/RFC1818, August 1995, <http://www.rfc-editor.org/info/rfc1818>.

[RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", RFC 1912, DOI 10.17487/RFC1912, February 1996, <http://www.rfc-editor.org/info/rfc1912>.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <http://www.rfc-editor.org/info/rfc5598>.

Appendix A. Test Vectors

[[NB: This appendix will include a large set of test vectors to test matching and validation patterns.]]

Appendix B. Change Log

Draft-03 just updates the date to keep the document active.

The document status is now marked as Informational instead of Best Current Practice (although it seems that it could go either way).

The authors decided to focus on "modern" ASCII-only email identifiers first and to get those right before tackling Unicode email identifiers and "obsolete" ABNF productions. This draft-01 preserves the main text of draft-00 rather than remove potentially useful text. Deliverable and modern email addresses, and modern Message-IDs, have been addressed. The Unicode work remains unfinished for now; "obsolete" ABNF productions (which are still useful for archival applications) will also be addressed in future drafts.

The authors decided to write one set of regular expressions in one dialect (namely, PCRE/PCRE2) before tackling others (e.g., JavaScript). Different dialects will be addressed in future drafts.

Authors' Addresses

Sean Leonard
Penango, Inc.
5900 Wilshire Blvd
Ste 2600
Los Angeles, CA 90036
USA

Email: dev+ietf@seantek.com

Joe Hildebrand
Cisco Systems

Email: jhildebr@cisco.com

Tony Hansen
AT&T Laboratories
200 Laurel Ave South
Middletown, NJ 07748
USA

Email: tony@att.com