```
Internet Draft                                            Paul Hoffman
draft-hoffman-imaa-03.txt                                   IMC & VPNC
October 8, 2003                                      Adam M. Costello
Expires April 8, 2004                                      UC Berkeley
```

Internationalizing Mail Addresses in Applications (IMAA)

Status of this Memo

Abstract

The Internationalizing Domain Names in Applications (IDNA) specification describes how to process domain names that contain characters outside the ASCII repertoire. A user who has a non-ASCII domain name may want to use it in an Internet mail address that contains non-ASCII characters not only in the domain part but also in the local part (the part to the left of the "@"). This document describes how to use non-ASCII characters in local parts. It defines internationalized local parts (ILPs), internationalized mail addresses (IMAs), and a mechanism called IMAA for handling them in a standard fashion.

1. Introduction

A mail address consists of local part, an at-sign (@), and a domain name. The IDNA specification [IDNA] describes how to handle domain names that have non-ASCII characters. This document describes how to handle non-ASCII characters in the rest of the mail address.

This document explicitly does not discuss internationalization of display names and comments in mail addresses that appear in message headers [MSGFMT]. MIME part three [MIME3] describes how use an extended set of characters in message headers, and this document does not alter that specification.

This document is being discussed on the ietf-imaa mailing list. See <http://www.imc.org/ietf-imaa/> for information about subscribing and the list's archive.

1.1 Relationship to IDNA

This document relies heavily on IDNA for both its concepts and its justification. This document omits a great deal of the justification and design information that might otherwise be found here because it is identical to that in IDNA. Anyone reading this document needs to have first read [IDNA], [PUNYCODE], [NAMEPREP], and [STRINGPREP].

There are a few key differences between the way IMAA treats local parts of mail addresses and the way IDNA treats domain names.

- The ACE infix for internationalized local parts is different from the ACE prefix for internationalized domain labels.

- Domain names have an intrinsic segmentation into labels, and are already segmented before transformations are performed. Local parts, on the other hand, have no intrinsic segmentation. The transformations on local parts perform a segmentation internally, but it has no external significance.

- There is no UseSTD3ASCIIRules flag for local parts.

One apparent difference that is not really a difference is the handling of quoting mechanisms. IDNA did not discuss quoting because the phrase "domain label" is presumed to refer to a simple literal string. [DNS] defines domain labels in terms of their literal form (which is used in DNS protocol messages), and later introduces a quoting syntax for representing domain labels in master files, but there is never any doubt that the domain label itself is a simple unstructured sequence. It goes without saying that domain labels obtained from contexts that use quoting (like master files) need to be reduced to their literal form before any processing is done on them.

Local parts, on the other hand, are defined in [MSGFMT] and [SMTP] in terms of their quoted form, as they appear in message headers and SMTP commands. Later it is stated that the quotation characters are not really part of the local part. To avoid any ambiguity, IMAA explicitly discusses the process of dequoting and requoting local parts.

2. Terminology

The key words "MUST", "SHALL", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

Code point, Unicode, and ASCII are defined in [IDNA].

The "protected code points" are 0..40, 5B..60, 7B..7F (in other words, those corresponding to ASCII characters other than letters and digits).

A "mail address" consists of a local part, an at-sign, and a domain name, in that order. The exact details of the syntax depend on the context; for example, a "mailbox" in [SMTP] and an "addr-spec" in [MSGFMT] are both mail addresses, but they define slightly different syntaxes for local parts and domain names.

A "dequoted local part" is the simple literal text string that is the intended "meaning" of a local part after it has undergone lexical interpretation. A dequoted local part excludes optional white space, comments, and lexical metacharacters (like backslashes and quotation marks used to quote other characters). Dequoted local parts are generally not allowed in protocols (like SMTP commands and message headers), but they are needed by IMAA as an intermediate form. The dequoted form of X is sometimes written dequote(X).

An "internationalized local part" (ILP) is anything that satisfies both of the following conditions: (1) It conforms to the same syntax as a non-internationalized local part except that non-ASCII Unicode characters are allowed wherever ASCII letters are allowed. (2) After it has been dequoted, the ToASCII operation can be applied to it without failing (see section 4). The term "internationalized local part" is a generalization, embracing both old ASCII local parts and new non-ASCII local parts. Although most Unicode characters can appear in internationalized local parts, ToASCII will fail for some inputs. Anything that fails to satisfy condition 2 is not a valid internationalized local part.

A "traditional local part" is a local part that contains only ASCII characters and whose dequoted form would be left unchanged by the ToUnicode operation (see section 4).

An "internationalized mail address" (IMA) consists of an internationalized local part, an at-sign, and an internationalized domain name [IDNA], in that order.

Equivalence of local parts is defined in terms of the dequoted form (see above) and the ToASCII operation, which constructs an ASCII form for a given dequoted local part (whether or not the local part was already an ASCII local part). Two traditional local parts X and Y are equivalent if and only if dequote(X) and dequote(Y) are exactly identical. (That is not a new rule, it is inferred from [SMTP] and [MSGFMT].) For internationalized local parts X and Y that are not both traditional, they are defined to be equivalent if and only if ToASCII(dequote(X)) matches ToASCII(dequote(Y)) using a case-insensitive ASCII comparison. Unlike traditional local parts, non-traditional internationalized local parts are always case-insensitive.

Two internationalized mail addresses are equivalent if and only if their local parts are equivalent (according to the previous definition) and their domain parts are equivalent (according to IDNA).

To allow internationalized labels to be handled by existing applications, IDNA uses an "ACE local part" (ACE stands for ASCII Compatible Encoding). An ACE local part is an internationalized local part that can be rendered in ASCII and is equivalent to an internationalized local part that cannot be rendered in ASCII. Given any internationalized local part (in dequoted form) that cannot be rendered in ASCII, the ToASCII operation will convert it to an equivalent ACE local part (whereas an ASCII local part will be left unaltered by ToASCII). ACE local parts are unsuitable for display to users. The ToUnicode operation will convert any local part (in dequoted form) to an equivalent non-ACE local part. In fact, an ACE local part is formally defined to be any local part that the ToUnicode operation would alter (whereas non-ACE local part are left unaltered by ToUnicode). The ToASCII and ToUnicode operations are specified in section 4.

The "ACE infix" is defined in this document to be a string of ASCII characters that occurs within every encoded segment in a dequoted ACE local part. It is specified in section 5.

A "mail address slot" is defined in this document to be a protocol element or a function argument or a return value (and so on) explicitly designated for carrying a mail address (or part of a mail address). Mail address slots exist, for example, in the MAIL and RCPT commands of the SMTP protocol, in the To: and Received: fields of message headers, and in a mailto: URI in the href attribute of an HTML <A> tag. General text that just happens to contain an mail address is not a mail address slot; for example, a mail address appearing in the plain text body of a message is not occupying a mail address slot.

An "IMA-aware mail address slot" is defined in this document to be a mail address slot explicitly designated for carrying an internationalized mail address as defined in this document. The designation may be static (for example, in the specification of the protocol or interface) or dynamic (for example, as a result of negotiation in an interactive session).

An "IMA-unaware mail address slot" is defined in this document to be any mail address slot that is not an IMA-aware mail address slot. Obviously, this includes any mail address slot whose specification predates this document.

3. Requirements and applicability

3.1 Requirements

IMAA conformance means adherence to the following four requirements:

1) In an internationalized mail address, the following characters MUST be recognized as at-signs for separating the local part from the domain name: U+0040 (commercial at), U+FF20 (fullwidth commercial at).

2) Whenever a mail address (or part of a mail address) is put into an IMA-unaware mail address slot (see section 2), it MUST contain only ASCII characters. Given an internationalized mail address, an equivalent mail address satisfying this requirement can be obtained by applying ToASCII to the local part as specified in section 4, changing the at-sign to U+0040, and processing the domain name as specified in [IDNA].

3) ACE local parts obtained from mail address slots SHOULD be hidden from users when it is known that the environment can handle the non-ACE form, except when the ACE form is explicitly requested. When it is not known whether or not the environment can handle the non-ACE form, the application MAY use the non-ACE form (which might fail, such as by not being displayed properly), or it MAY use the ACE form (which will look unintelligible to the user). Given an internationalized local part, an equivalent non-ACE local part can be obtained by applying the ToUnicode operation as specified in section 4. When requirements 2 and 3 both apply, requirement 2 takes precedence.

4) If two mail addresses are equivalent and either one refers to a mailbox, then both MUST refer to the same mailbox, regardless of whether they use the same form of at-sign.

   Discussion: This implies that non-ASCII local parts cannot be deployed in domains whose mail exchangers are case-sensitive. IMAA is designed to work without upgrading mail exchangers, but it works only for mail exchangers that treat ASCII local parts as case-insensitive (which is the common and preferred behavior). All local parts received by an IMA-unaware mail exchanger are ASCII, either traditional or ACE, and a case-insensitive exchanger will automatically obey requirement 4 without being aware of it. Case-sensitive exchangers will not correctly handle ACE local parts, but administrators can simply refrain from creating ACE local parts in those domains. This is necessary because a round-trip through ToUnicode and ToASCII is not case-preserving, and therefore the result might refer to a different mailbox (in violation of requirement 4) if interpreted by a case-sensitive mail exchanger.

3.2 Applicability

IMAA is applicable to all mail addresses in all mail address slots except where it is explicitly excluded.

This implies that IMAA is applicable to protocols that predate IMAA. Note that mail addresses occupying mail address slots in those protocols MUST be in ASCII form (see section 3.1, requirement 2).

3.2.1. Case-sensitive local parts

IMAA does not apply to local parts that are interpreted case-sensitively (see section 3.1 requirement 4).

3.2.2. Local parts versus domain names

The IMAA ToASCII and ToUnicode operations apply to local parts, not to domain labels. The IDNA ToASCII and ToUnicode operations apply to domain labels, not to local parts. There exist conventions for transplanting local parts into domain labels (in DNS SOA records, for example), and there may exist conventions for transplanting domain names into local parts. Such conventions that predate IMAA are IMA-unaware, and therefore the domain labels receiving the transplanted local parts and the local parts receiving the transplanted domain names are IMA-unaware slots. Therefore the strings MUST be in ASCII form before they are transplanted. If they were transplanted in non-ASCII form they would risk being passed through the wrong ToASCII operation.

4. Conversion operations

An application converts a local part put into an IMA-unaware mail address slot or displayed to a user. This section specifies the steps to perform in the conversion, and the ToASCII and ToUnicode operations.

The input to ToASCII or ToUnicode is a dequoted local part that is a sequence of Unicode code points (remember that all ASCII code points are also Unicode code points). If a local part is represented using a character set other than Unicode or US-ASCII, it will first need to be transcoded to Unicode.

Starting from a local part, the steps that an application takes to do the conversions are:

1) Decide whether the local part is a "stored string" or a "query string" as described in [STRINGPREP] (see section 6 below for a discussion). If this conversion follows the "queries" rule from [STRINGPREP], set the flag called "AllowUnassigned".

2) Save a copy of the local part.

3) Dequote the local part; that is, perform lexical interpretation and remove all nonliteral characters. For example, for a local part that uses the lexical syntax of [SMTP] or [MSGFMT], unfold it, remove comments and unquoted white space, and remove backslashes and quotation marks used to quote other characters. The result is a simple literal text string.

4) Process the string with either the ToASCII or the ToUnicode operation as appropriate. Typically, you use the ToASCII operation if you are about to put the local part into an IMA-unaware slot, and you use the ToUnicode operation if you are displaying the local part to a user.

5) If step 4 had no effect on the string, and if the saved local part from step 2 is a valid representation of the string in the destination context, then the saved local part SHOULD be used, otherwise proceed to step 6.

6) Apply whatever quoting is needed in the destination context (if any). For "mailbox" slots [SMTP] and "addr-spec" slots [MSGFMT] the following action suffices: If the string contains any control characters, spaces, or specials [MSGFMT], or if it begins or ends with a dot, or contains two consecutive dots, then convert it to a quoted-string: insert a backslash before every quotation mark and backslash, then enclose the string with quotation marks.

The destination context might also impose a length restriction. Depending on whether the restriction applies to the quoted form or the dequoted form, the application might want to check the length at the very end or just after step 4.

This process is designed to handle quoting and dequoting when necessary; however, local parts that need quoting can be difficult for humans to use. This is already true for ASCII local parts, and is even more true for internationalized local parts. It is inadvisable to create such local parts if they are to be used by humans.

The following two subsections define the ToASCII and ToUnicode operations that are used in step 4.

In ToASCII and ToUnicode, the operation of Nameprep is split into two halves that are applied at different times. One half consists of Nameprep steps 1 (map) and 2 (normalize); the other half consists of Nameprep steps 3 (prohibit) and 4 (check bidi). The split is easy to remember because steps 1 and 2 are string transformations that can never fail, while steps 3 and 4 are checks that do nothing but succeed or fail.

This description of the protocol uses specific procedure names, names of flags, and so on, in order to facilitate the specification of the protocol. These names, as well as the actual steps of the procedures, are not required of an implementation. In fact, any implementation which has the same external behavior as specified in this document conforms to this specification.

4.1 ToASCII

The ToASCII operation takes a sequence of Unicode code points that make up a dequoted local part and transforms it into a sequence of code points in the ASCII range (0..7F). If ToASCII succeeds, the original sequence and the resulting sequence are equivalent dequoted local parts.

It is important to note that the ToASCII operation can fail. ToASCII fails if any step of it fails. If any step of the ToASCII operation fails, that string MUST NOT be used as an internationalized local part. The method for dealing with this failure is application-specific.

The inputs to ToASCII are a sequence of code points, and the AllowUnassigned flag. The output of ToASCII is either a sequence of ASCII code points or a failure condition.

ToASCII never alters a sequence of code points that are all in the ASCII range to begin with. Applying the ToASCII operation multiple times has exactly the same effect as applying it just once.

ToASCII consists of the following steps:

1. If the sequence contains any code points outside the ASCII range (0..7F) then proceed to step 2, otherwise stop, leaving the sequence unchanged.

2. Perform [NAMEPREP] steps 1 (map) and 2 (normalize).

3. If the sequence is empty then stop, leaving an empty result.

4. Divide the sequence into segments. Segment boundaries occur wherever a protected code point is adjacent to a non-protected code point, and nowhere else. (Therefore segments are never empty, and they alternate between segments containing only protected code points and segments containing only non-protected code points.)

5. For each segment perform the following substeps:

   (a) If the segment contains any code points outside the ASCII range (0..7F) then proceed to substep b, otherwise leave the segment unchanged.

   (b) Perform [NAMEPREP] steps 3 (prohibit) and 4 (check bidi), and fail if there is an error. The AllowUnassigned flag is used in [NAMEPREP] step 3.

   (c) Encode the sequence using the encoding algorithm in [PUNYCODE] and fail if there is an error.

   (d) Verify that the result contains no more than 59 code points.

   (e) The sequence will contain at most one instance of U+002D (hyphen-minus). If it is absent then prepend the ACE infix; otherwise verify that the ACE infix does not already occur before the hyphen-minus, and substitute the ACE infix in place of it.

6. Rejoin the segments into a single sequence.

4.2 ToUnicode

The ToUnicode operation takes a sequence of Unicode code points that make up a dequoted local part and returns a sequence of Unicode code points. If the input sequence is a dequoted local part in ACE form, then the result is an equivalent dequoted internationalized local part that is not in ACE form, otherwise the original sequence is returned unaltered.

ToUnicode never fails. If any step fails, then the original input sequence is returned immediately in that step.

The Punycode decoder can never output more code points than it inputs, but Nameprep can, and therefore ToUnicode can. Note that the number of octets needed to represent a sequence of code points depends on the particular character encoding used.

The inputs to ToUnicode are a sequence of code points, and the AllowUnassigned flag. The output of ToUnicode is a sequence of code points.

ToUnicode consists of the following steps:

1. If the sequence contains any code points outside the ASCII range (0..7F) then proceed to step 2, otherwise skip to step 3.

2. Perform [NAMEPREP] steps 1 (map) and 2 (normalize).

3. Verify that the sequence is nonempty.

4. Divide the sequence into segments (same as step 4 of ToASCII).

5. For each segment perform the following substeps:

   (a) If the ACE infix does not occur anywhere within the segment then leave the segment unchanged, otherwise save a copy of the segment and proceed to substep b.

   (b) If the ACE infix occurs at the very beginning of the segment then remove it, otherwise substitute U+002D (hyphen-minus) in place of the first occurrence of the ACE infix.

   (c) Decode the segment using the decoding algorithm in [PUNYCODE] and catch any error. If there was an error then restore the saved copy from substep a.

6. Verify that at least one segment was altered in step 5.

7. Rejoin the segments into a single sequence, and save a copy of the result.

8. Apply ToASCII to the current sequence and to a copy of the original input.

9. Verify that the two results of step 8 match using a case-insensitive ASCII comparison.

10. Return the saved copy from step 7.

5. ACE infix

[[Note to the IESG and Internet Draft readers: The two uses of the string "0iesg1" below are to be changed at time of publication to an infix that fulfills the requirements in the first paragraph. IANA will assign this value.]]

The ACE infix, used in the conversion operations (section 4), is two ASCII letters surrounded by two distinct ASCII digits. The ToASCII and ToUnicode operations MUST recognize the ACE infix in a case-insensitive manner.

The ACE infix for IMAA is "0iesg1" or any capitalization thereof.

This means that an ACE local part might be "foobar!de0iesg1jg4avhby1noc0d!0iesg1d9juau41awczczp", where "de-jg4avhby1noc0d" and "d9juau41awczczp" are the results of the encoding steps in [PUNYCODE].

While every encoded segment (segment that would be altered by

ToUnicode) within an ACE local part contains the ACE infix, not every segment containing the ACE infix is an encoded segment. Segments that contain the ACE infix but are not encoded segments will confuse users, and local parts containing such segments SHOULD NOT be used as mailbox names.

6. Stored strings and query strings

[STRINGPREP] prohibits unassigned code points in "stored strings" and allows them in "query strings", but concedes that "different Internet protocols use strings very differently, so these terms cannot be used exactly in every protocol that needs to use stringprep". In the context of IMAA, the following clarifications apply.

A string that assigns/creates the name of an object is a "stored string". A string that merely refers to an object using a name that is presumed to have been assigned/created elsewhere is a "query string".

Examples of stored strings:

* In a mail server configuration file/database, the strings that create the mail addresses associated with the local mailboxes. (These mail addresses might be defined in pieces: the domain parts might be defined by a set of local domains, and the local parts might be defined by a separate set of user names and aliases, but the net effect is that these strings create a set of mail addresses, and are therefore stored strings.)

* The msg-id in the Message-ID: field of a message header.

Examples of query strings:

* A mail address in the From: or To: or Reply-To: field of a message header.

* A mail address in the MAIL or RCPT command of SMTP.

* A mail address in a personal address book.

* A msg-id in the In-Reply-To: or References: field of a message header.

7. References

7.1 Normative references

[IDNA]   Faltstrom, P., Hoffman, P. and A. Costello,
             "Internationalizing Domain Names in Applications

(IDNA)", RFC 3490, March 2003.

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

[MSGFMT] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[NAMEPREP] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[PUNYCODE] Costello, A., "Punycode: A Bootstring encoding of Unicode for use with Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[SMTP] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[STRINGPREP] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

7.2 Informative references

[DNS] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034 and "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[MIME3] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

8. Security considerations

Because this document normatively refers to [IDNA], [NAMEPREP], [PUNYCODE], and [STRINGPREP], it includes the security considerations from those documents as well.

Internationalized local parts will cause mail addresses to become longer, and possibly make it harder to keep lines in a header under 78 characters. Lines that are longer than 78 characters (which is a SHOULD specification, not a MUST specification, in RFC 2822) could possibly cause mail user agents to fail in ways that affect security.

9. IANA considerations

IANA will assign the ACE infix in consultation with the IESG.

10. Authors' addresses

Paul Hoffman
Internet Mail Consortium and VPN Consortium
127 Segre Place
Santa Cruz, CA 95060 USA
phoffman@imc.org

Adam M. Costello
University of California, Berkeley
http://www.nicemice.net/amc/