1. INTRODUCTION

This memo defines the mime content-type 'message/cpim. This is a common message format for CPIM-compliant messaging protocols [14].

While being prepared for CPIM, this format is quite general and may be reused by other applications with similar requirements. Application specifications that adopt this as a base format should answer the questions rasied in section 6 of this document.

1.1 Motivation

The Common Profile for Instant Messaging (CPIM) [14] specification defines a number of operations to be supported and criteria to be satisfied for interworking diverse instant messaging protocols. The intent is to allow a variety of different protocols interworking through gateways to support cross-protocol messaging that meets the requirements of RFC 2779 [15].

To adequately meet the security requirements of RFC 2779, a common message format is needed so that end-to-end signatures and encryption may be applied. This document describes a common canonical message format that must be used by any CPIM-compliant message transfer protocol, and over which signatures are calculated for end-to-end security.

1.2 Background

RFC 2779 requires that an instant message can carry a MIME payload [3,4]; thus some level of support for MIME will be a common element of any CPIM compliant protocol. Therefore it seems reasonable that a common message format should use a MIME/RFC822 syntax, as protocol implementations must already contain code to parse this.

Unfortunately, using pure RFC822/MIME [2] can be problematic:

- Irregular lexical structure -- RFC822 allows a number of optional encodings and multiple ways to encode a particular value. For example RFC822 comments may be encoded in multiple ways. For security purposes, a single encoding method must be defined as a basis for computing message digest values. Protocols that transmit data in a different format would otherwise lose information needed to verify a signature.

- Weak internationalization -- RFC822 requires header values to use 7-bit ASCII, which is problematic for encoding international character sets. Mechanisms for language tagging in RFC822 headers [16] are awkward to use and have limited applicability.

- Mutability -- addition, modification or removal of header information. Because it is not explicitly forbidden, many applications that process MIME content (e.g. MIME gateways) rebuild or restructure messages in transit. This obliterates most attempt at achieving security (e.g. signatures), leaving receiving applications unable to verify the received data.

- Message and payload separation -- there is not a clear syntactic distinction between message metadata and message content.

- Limited extensibility (X-headers are problematic).

- No support for structured information (text string values only).

- Some processors impose line length limitations The message format defined by this memo overcomes some of these difficulties by having a syntax that is generally compatible with the format accepted by MIME/RFC822 parsers, but simplified, and having a stricter syntax. It also defines mechanisms to support some desired features not covered by the RFC822/MIME format specifications.

1.3 Goals

This specification aims to satisfy the following goals:

- a securable end-to-end format for a message (a canonical message format for signature calculation)

- independent of any specific application

- capable of conveying a range of different address types

- assumes an 8-bit clean message-transfer protocol

- evolvable: extensible by multiple parties

- to clearly separate message metadata from message content

- a simple, regular, easily parsed syntax

- a compact, low-overhead format for simple messages

1.4 Terminology and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

> NOTE: Comments like this provide additional nonessential information about the rationale behind this document. Such information is not needed for building a conformant implementation, but may help those who wish to understand the design in greater depth.

[[[Editorial comments and questions about outstanding issues are provided in triple brackets like this. These working comments should be resolved and removed prior to final publication.]]]

2. OVERALL MESSAGE STRUCTURE

The message/cpim format encapsulates an arbitrary MIME message content, together with message- and content-related metadata. This can optionally be signed or encrypted using MIME security multiparts in conjunction with an appropriate security scheme.

A message/cpim object is a multipart entity, where the first part contains the message metadata and the second part is the message content. The two parts are syntactically separated by a blank line, to keep the message header information (with its more stringent syntax rules) separate from the MIME message content headers.

Thus, the complete message looks something like this:

```
m: Content-type: message/cpim
s:
h: (message-metadata-headers)
s:
e: (encapsulated MIME message-body)
```

The end of the message body is defined by the framing mechanism of the protocol used. The tags 'm:', 's:', 'h:', 'e:', and 'x:' are not part of the message format and are used here to indicate the different parts of the message, thus:

```
m:  MIME headers for the overall message
s:  a blank separator line
h:  message headers
e:  encapsulated MIME object containing the message content
x:  MIME security multipart message wrapper
```

2.1 Message/cpim MIME headers

The message MIME headers identify the message as a CPIM-formatted message. The only required header is:

Content-type: message/cpim

Other MIME headers may be used as appropriate for the message transfer environment.

2.2 Message headers

Message headers carry information relevant to the end-to-end transfer of the message from sender to receiver. Message headers MUST NOT be modified, reformatted or reordered in transit, but in some circumstances they MAY be examined by a CPIM message transfer protocol.

The message headers serve a similar purpose to RFC822 message headers in email [2], and have a similar but restricted allowable syntax.

The basic header syntax is:

Key: Value

where "Key" is a header name and "Value" is the corresponding header value. The following considerations apply:

- The entire header MUST be contained on a single line. The line terminator is not considered part of the header value.
- Only one header per line. Multiple headers MUST NOT be included on a single line.
- Processors SHOULD NOT impose any line-length limitations.
- There MUST NOT be any whitespace at the beginning or end of a line.
- UTF-8 character encoding [21] MUST be used throughout.
- The character sequence CR,LF (13,10) MUST be used to terminate each line.
- The header name contains only US-ASCII characters (see later for the specific syntax)

- The header MUST NOT contain any control characters (0-31). If a header value needs to represent control characters then the escape mechanism described below MUST be used.

- There MUST be a single space character (32) following the header name and colon.

- Multiple headers using the same key (header name) are allowed. (Specific header semantics may dictate only one occurrence of any particular header.)

- Headers names MUST match exactly (i.e. "From:" and "from:" are different headers).

- If a header name is not recognized or not understood, the header should be ignored. But see also the "Requires:" header.

- Interpretation (e.g. equivalence) of header values is dependent on the particular header definition. Message processors MUST preserve exactly all octets of all headers (both name and value).

- Message processors MUST NOT change the order of message headers.

Examples:

```
To: Pooh Bear <im:pooh@100akerwood.com>
From: <im:piglet@100akerwood.com>
Date: 2001-02-02T10:48:54-05:00
```

2.3 Character escape mechanism

This mechanism MUST be used to code control characters in a header, having Unicode code points in the range U+0000 to U+001f or U+007f. (The escape mechanism is as used by the Java programming language.) Note that the escape mechanism is applied to a UCS-2 character, NOT to the octets of its UTF-8 coding. Mapping from/to UTF-8 coding is performed without regard for escape sequences or character coding. (The header syntax is defined so that octets corresponding to control characters other than CR and LF do not appear in the output.)

An arbitrary UCS-2 character is escaped using the form:

\uxxxx

where:

\ is U+005c (backslash)
u is U+0075 (lower case letter U)
xxxx is a sequence of exactly four hexadecimal digits
(0-9, a-f or A-F) or
(U+0030-U+0039, U+0041-U+0046, or U+0061-0066)

The hexadecimal number 'xxxx' is the UCS code-point value of the escaped character.

Further, the following special sequences introduced by "\" are used:

\\ for \ (backslash, U+005c)
\" for " (double quote, U+0022)
\' for ' (single quote, U+0027)
\b for backspace (U+0008)
\t for tab (U+0009)
\n for linefeed (U+000a)
\r for carriage return (U+000d)

2.3.1 Escape mechanism usage

When generating messages conformant with this specification:

- The special sequences listed above MUST be used to encode any occurrence of the following characters that appear anywhere in a header: backslash (U+005c), backspace (U+0008), tab (U+0009), linefeed (U+000a) or carriage return (U+000d).

- The special sequence \' MUST be used for any occurrence of a single quote (U+0027) that appears within a string delimited by single quotes.

- The special sequence \" MUST be used for any occurrence of a double quote (U+0022) that appears within a string delimited by double quotes.

- Quote characters that delimit a string value MUST NOT be escaped.

- The general escape sequence \uxxxx MUST be used for any other control character (U+0000 to U+0007, U+000b to U+000c, U+000e to U+001f or u+007f) that appears anywhere in a header.

- All other characters MUST NOT be represented using an escape sequence.

When processing a message based on this specification, the escape sequence usage described above MUST be recognized.

Further, any other occurrence of any escape sequence described above SHOULD be recognized and treated as an occurrence of the corresponding Unicode character.

Any backslash ('\') character SHOULD be interpreted as introducing an escape sequence. Any unrecognized escape sequence SHOULD be treated as an instance of the character following the backslash character. An isolated backslash that is the last character of a header SHOULD be ignored.

2.4 Message content

The final section of a message/cpim is the MIME-encapsulated message content, which follows standard MIME formatting rules [3,4].

The MIME content headers MUST include at least a Content-Type header. The content may be any MIME type.

Example:

```
e: Content-Type: text/plain; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: This is my encapsulated text message content
```

3. MESSAGE HEADER SYNTAX

A header is made of two parts, a name and a value, separated by a colon character (':') followed by a single space (32), and terminated by a sequence of CR,LF (13,10).

Headers use UTF-8 character encoding thoughout, per RFC 2279 [21].

3.1 Header names

The header name is a sequence of US-ASCII characters, excluding control characters, SPACE or separator characters. Use of the character "." in a header name is reserved for a namespace prefix separator.

Separator characters are:

```
SEPARATORS   = "(" / ")" / "<" / ">" / "@"
             / "," / ";" / ":" / "
             / "/" / "[" / "]" / "?" / "="
             / "{" / "}" / SP
```

NOTE: the range of allowed characters was determined by examination of HTTP and RFC822 header name formats and choosing the more resticted. The intent is to allow CPIM headers to follow a syntax that is compatible with the allowed syntax for both RFC 822 [2] and HTTP [18] (including HTTP-derived protocols such as SIP).

3.2 Header Value

A header value has a structure defined by the corresponding header specification. Implementations that use a particular header must adhere to the format and usage rules thus defined when creating or processing a message containing that header.

The other general constraints on header formats MUST also be followed (one line, UTF-8 character encoding, no control characters, etc.)

3.3 Language Tagging

Full internationalization of a protocol requires that a language can be indicated for any human-readable text [6,19].

A message header may indicate a language for its value by including ';lang=tag' after the header name and colon, where 'tag' is a language identifying token per RFC 3066 [7].

Example:

```
Subject:;lang=fr Objet de message
```

If the language parameter is not applied a header, any human-readable text is assumed to use the language identified as 'i-default' [19].

3.4 Namespaces for header name extensibility

NOTE: this section defines a framework for header extensibility whose use is optional. If no header extensions are allowed by an application then these structures may never be used.

An application that uses this message format is expected to define the set of headers that are required and allowed for that application. This section defines a header extensibility framework that can be used with any application.

The extensibility framework is based on that provided for XML [11] by XML namespaces [12]. All headers are associated with a "namespace", which is in turn associated with a globally unique URI.

Within a particular message instance, header names are associated with a particular namespace through the presence or absence of a namespace prefix, which is a leading part of the header name followed by a period ("."); e.g.

```
prefix.header-name: header-value
```

Here, 'prefix' is the header name prefix, 'header-name' is the header name within the namespace associated with 'prefix', and 'header-value' is the value for this header.

```
header-name: header-value
```

In this case, the header name prefix is absent, and the given 'header-name' is associated with a default namespace.

An application that uses this format designates a default namespace for any headers that are not more explicitly associated with any namespace. In many cases, the default namespace may be all that is needed.

A namespace is identified by a URI. In this usage, the URI is used simply as a globally unique identifier, and there is no requirement that it can be used for any other purpose. Any legal globally unique URI MAY be used to identify a namespace. (By "globally unique", we mean constructed according to some set of rules so that it is reasonable to expect that nobody else will use the same URI for a different purpose.) A URI used as an identifier MUST be a full absolute-URI, per RFC 2396 [10]. (Relative URIs and URI- references containing fragment identifiers MUST NOT be used for this purpose.)

Within a specific message, a 'NS' header is used to declare a namespace prefix and associate it with a URI that identifies a namespace. Following that declaration, within the scope of that message, the combination of namespace prefix and header name indicates a globally unique identifier for the header (consisting of the namespace URI and header name). For example:

```
NS: MyFeatures <mid:MessageFeatures@id.foo.com>
MyFeatures.WackyMessageOption: Use-silly-font
```

This defines a namespace prefix 'MyFeatures' associated with the namespace identifier 'mid:MessageFeatures@id.foo.com'. Subsequently the prefix indicates that the WackyMessageOption header name referenced is associated with the identified namespace.

A namespace prefix declaration MUST precede any use of that prefix.

With the exception of any application-specific predefined namespace prefixes (see section 6), a namespace prefix is strictly local to the message in which it occurs. The actual prefix used has no global significance. This means that the headers:

```
xxx.name: value
yyy.name: value
```

in two different messages may have exactly the same effect if namespace prefixes 'xxx' and 'yyy' are associated with the same namespace URI. Thus the following have exactly the same meaning:

```
NS: acme <http://id.acme.widgets/wily-headers/>
acme.runner-trap: set
```

and

```
NS: widget <http://id.acme.widgets/wily-headers/>
widget.runner-trap: set
```

A 'NS' header without a header prefix name specifies a default namespace for subsequent headers; that is a namespace that is associated with header names not having a prefix. For example:

```
NS: <http://id.acme.widgets/wily-headers/>
runner-trap: set
```

has the same meaning as the previous examples.

This framework allows different implementers to create extension headers without the worry of header name duplication; each defines headers within their own namespace.

3.5 Mandatory-to-recognize features

Sometimes it is necessary for the sender of a message to insist that some functionality is understood by the recipient. By using the mandatory-to-recognize indicator, a sender is notifying the recipient that it MUST understand the named header or feature in order to properly understand the message.

A header or feature is indicated as being mandatory-to-recognize by a 'Require:' header. For example:

```
Require: MyFeatures.VitalMessageOption
MyFeatures.VitalMessageOption: Confirmation-requested
```

Multiple required header names may be listed in a single 'Require' header, separated by commas.

> NOTE: indiscriminate use of 'Require:' headers could harm interoperability. It is suggested that any implementer who defines required headers also publish the header specifications so other implementations can succesfully interoperate.

The 'Require:' header MAY also be used to indicate that some non-header semantics must be implemented by the recipient, even when it does not appear as a header. For example:

```
Require: Locale.MustRenderKanji
```

might be used to indicate that message content includes characters from the Kanji repertoire, which must be rendered for proper understanding of the message. In this case, the header name is just a token (using header name syntax and namespace association) that indicates some desired behaviour.

3.6 Collected message header syntax

The following description of message header syntax uses ABNF, per RFC 2234 [17]. Most of this syntax can be interpreted as defining UCS character sequences or UTF-8 octet sequences. Alternate productions at the end allow for either interpretation.

```
Header        = Header-name ":" *( ";" Parameter ) SP
                Header-value
                CRLF

Header-name   = [ Name-prefix "." ] Name
Name-prefix   = Name

Parameter     = Lang-param / Ext-param
Lang-param    = "lang=" Language-tag
Ext-param     = Param-name "=" Param-value
Param-name    = Name
Param-value   = Token / Number / String

Header-value  = *HEADERCHAR

Name          = 1*NAMECHAR
Token         = 1*TOKENCHAR
Number        = 1*DIGIT
String        = DQUOTE *( Str-char / Escape ) DQUOTE
Str-char      = %x20-21 / %x23-5B / %x5D-7E / UCS-high
Escape        = "\" ( "u" 4(HEXDIG)     ; UCS codepoint
                    / "b"               ; Backspace
                    / "t"               ; Tab
                    / "n"               ; Linefeed
                    / "r"               ; Return
                    / DQUOTE            ; Double quote
                    / "'"               ; Single quote
                    / "\" )             ; Backslash

Formal-name   = 1*( Token SP ) / String
URI           = <defined as absolute-URI by RFC 2396>
Language-tag  = <defined by RFC 3066>

              ; Any UCS character except CTLs, or escape
HEADERCHAR    = UCS-no-CTL / Escape

              ; Any US-ASCII char except ".", CTLs or SEPARATORS:
NAMECHAR      = %21 / %23-26 / %2a-2b / %2d / %5e-60 / %7c / %7e
              / ALPHA / DIGIT

              ; Any UCS char except CTLs or SEPARATORS:
TOKENCHAR     = NAMECHAR / "." / UCS-high
```

```
SEPARATORS   = "(" / ")" / "<" / ">" / "@"     ; 28/29/3c/3e/40
             / "," / ";" / ":" / "\" / <">     ; 2c/3b/3a/5c/22
             / "/" / "[" / "]" / "?" / "="     ; 2f/5b/5d/3f/3d
             / "{" / "}" / SP                  ; 7b/7d/20
CTL          = <Defined by RFC 2234 -- %x0-%x1f, %x7f>
CRLF         = <Defined by RFC 2234 -- CR, LF>
SP           = <defined by RFC 2234 -- %x20>
DIGIT        = <defined by RFC 2234 -- '0'-'9'>
HEXDIG       = <defined by RFC 2234 -- '0'-'9', 'A'-'F', 'a'-'f'>
ALPHA        = <defined by RFC 2234 -- 'A'-'Z', 'a'-'z'>
DQUOTE       = <defined by RFC 2234 -- %x22>
```

To interpret the syntax in a general UCS character environment, use the following productions:

```
UCS-no-CTL   = %x20-7e / UCS-high
UCS-high     = %x80-ffffffff
```

To interpret the syntax as defining UTF-8 coded octet sequences, use the following productions:

```
UCS-no-CTL   = UTF8-no-CTL
UCS-high     = UTF8-multi
UTF8-no-CTL  = %x20-7e / UTF8-multi
UTF8-multi   = %xC0-DF %x80-BF
             / %xE0-EF %x80-BF %x80-BF
             / %xF0-F7 %x80-BF %x80-BF %x80-BF
             / %xF8-FB %x80-BF %x80-BF %x80-BF %x80-BF
             / %xFC-FD %x80-BF %x80-BF %x80-BF %x80-BF %x80-BF
```

4. HEADER DEFINITIONS

This specification defines a core set of headers that are defined and available for use by applications: the application specification must indicate the headers that may be used, those that must be recognized and those that must appear in any message (see section 6).

The header definitions that follow fall into two categories:

(a) those that are part of the CPIM format extensibility framework, and

(b) some that have been based on similar headers in RFC 822, specified here with corresponding semantics.

Header names and syntax are given without a namespace qualification, and the associated namespace URI is listed as part of the header

description. Any of the namespace associations already mentioned (implied default namespace, explicit default namespace or implied namespace prefix or explicit namespace prefix declaration) may be used to identify the namespace.

All headers defined here are associated with the namespace URI <[[[urn:iana:cpim-headers]]]>, which is defined according to [22].

4.1 The 'From' header

Indicates the sender of a message.

Header name: From

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
From-header = "From" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates the sender or originator of a message.

If present, the 'Formal-name' identifies the person or "real world" name for the originator.

The URI indicates an address for the originator.

Examples:

```
From: Winnie the Pooh <im:pooh@100akerwood.com>

From: <im:tigger@100akerwood.com>
```

4.2 The 'To' header

Specifies an intended recipient of a message.

Header name: To

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
To-header = "To" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates the recipient of a message.

If present, the 'Formal-name' identifies the person or "real world" name for the recipient.

The URI indicates an address for the recipient.

Multiple recipients may be indicated by including multiple 'To' headers.

Examples:

To: Winnie the Pooh <im:pooh@100akerwood.com>

To: <im:tigger@100akerwood.com>

4.3 The 'cc' header

Specifies a non-primary recipient ("courtesy copy") for a message.

Header name: cc

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Cc-header   = "cc" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates a courtesy copy recipient of a message.

If present, the 'Formal-name', if present, identifies the person or "real world" name for the recipient.

The URI indicates an address for the recipient.

Multiple courtesy copy recipients may be indicated by including multiple 'cc' headers.

Examples:

cc: Winnie the Pooh <im:pooh@100akerwood.com>

cc: <im:tigger@100akerwood.com>

4.4 The 'DateTime' header

Specifies the date and time a message was sent.

Header name: Date

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax:

DateTime-header = "DateTime" ": " date-time

(where the syntax of 'date-time' is a profile of ISO8601, defined in "Date and Time on the Internet" [23])

Description:

The 'Date' header supplies the current date and time at which the sender sent the message.

One purpose of the this header is to provide for protection against a replay attack, by allowing the recipient to know when the message was intended to be sent. The value of the date header is the current time at the sender when the message was transmitted, using ISO 8601 date and time format as profiles in "Date and Time on the Internet: Timestamps" [23].

Example:

Date: 2001-02-01T12:16:49-05:00

4.5 The 'Subject' header

Contains a description of the topic of the message.

Header name: Subject

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Subject-header = "Subject" ":" [ lang-param ] SP *HEADERCHAR
```

Description:

The 'Subject' header supplies the sender's description of the topic or content of the message.

The sending agent should specify the language parameter if it has any reasonable knowledge of the language used by the sender to describe the message.

Example:

```
Subject:;lang=en Eeyore's feeling very depressed today
```

4.6 The 'NS' header

The "NS" header is used to declare a local namespace prefix.

Header name: NS

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
NS-header = "NS" ": " [ Name-prefix ] "<" URI ">"
```

Description:

Declares a namespace prefix that may be used in subsequent header names. See section 3.4 for more details.

Example:

```
NS: MyAlias <mid:MessageFeatures@id.foo.com>
MyAlias.MyHeader: private-extension-data
```

4.7 The 'Require' header

Specify a header or feature that must be implemented by the receiver for correct message processing.

Header name: NS

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Require-header = "Require" ": " Header-name *( "," Header-name )
```

Description:

Declares a namespace prefix that may be used in subsequent header names. See section 3.5 for more details.

Note that there is no requirement that the required header actually be used, but for brevity it is recommended that an implemention not use issue require header for unused headers.

Example:

```
Require: MyAlias.VitalHeader
```

5. EXAMPLES

The examples in the following sections use the following per-line tags to indicate different parts of the overall message format:

- m: MIME headers for the overall message
- s: a blank separator line
- h: message headers
- e: encapsulated MIME object containing the message content
- x: MIME security multipart message wrapper

The following examples also assume that <[[[urn:iana:cpim-headers]]]> is the implied default namespace for the application concerned.

5.1 An example message/cpim message

The following example shows a message/cpim message:

```
m: Content-type: message/cpim
s:
h: From: MR SANDERS <im:piglet@100akerwood.com>
h: To: Depressed Donkey <im:eeyore@100akerwood.com>
h: Date: 2000-12-13T13:40:00-08:00
h: Subject: the weather will be fine today
h: Subject:;lang=fr beau temps prevu pour aujourd'hui
h: NS: MyFeatures <mid:MessageFeatures@id.foo.com>
h: Require: MyFeatures.VitalMessageOption
h: MyFeatures.VitalMessageOption: Confirmation-requested
h: MyFeatures.WackyMessageOption: Use-silly-font
s:
e: Content-type: text/xml; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: <body>
e: Here is the text of my message.
e: </body>
```

5.2 An example using MIME multipart/signed

In order to secure a message/cpim, an application or implementation should use RFC 1847 and some appropriate cryptographic scheme.

Using S/MIME and pkcs7, the above message would look like this:

```
x: Content-Type: multipart/signed; boundary=next;
                 MDALG=SHA-1; type=application/pkcs
x:
x: --next
m: Content-Type: message/cpim
s:
h: From: MR SANDERS <im:piglet@100akerwood.com>
h: To: Dopey Donkey <im:eeyore@100akerwood.com>
h: Date: 2000-12-13T13:40:00-08:00
h: Subject: the weather will be fine today
h: Subject:;lang=fr beau temps prevu pour aujourd'hui
h: NS: MyFeatures <mid:MessageFeatures@id.foo.com>
h: Require: MyFeatures.VitalMessageOption
h: MyFeatures.VitalMessageOption: Confirmation-requested
h: MyFeatures.WackyMessageOption: Use-silly-font
s:
```

```
e: Content-type: text/xml; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: <body>
e: Here is the text of my message.
e: </body>
x: --next
x: Content-Type: application/pkcs7
x:
x: (signature stuff)
    :
x: --next--
```

6. APPLICATION DESIGN CONSIDERATIONS

Applications using this specification must specify:

- a default namespace URI for messages created and processed by that application
- any namespace prefixes that are implicitly defined for messages created and processed by that application
- all headers that must be recognized by implementations of the application
- any headers that must be present in messages created by that application.
- any headers that may appear more than once in a message, and how they are to be interpreted (e.g. how to interpret multiple 'subject:' headers with different language parameter values).

Within a network of message transfer agents, an intermediate gateway MUST NOT change the message/cpim content in any way. This implies that headers cannot be changed or reordered, transfer encoding cannot be changed, languages cannot be changed, etc.

Because message/cpim messages are immutable, any transfer agent that wants to modify the message should create a new message/cpim message with the modified header and containing the original message as its content. (This approach is similar to real-world bill-of-lading handling, where each person in the chain attaches a new sheet to the message. Then anyone can validate the original message and see what was changed and who changed it by following the trail of amendments. Another metaphor is including the old message in a new envelope.)

7. IANA CONSIDERATIONS

[[[Registration template for message/cpim content type]]]

[[[Registration of namespace URN for CPIM headers]]]

8. INTERNATIONALIZATION CONSIDERATIONS

Message headers use UTF-8 character encoding throughout, so can convey the full UCS-4 (Unicode, ISO/IEC 10646) character repertoire.

Language tagging is provided for message headers using the "Language" parameter.

Message content is any MIME-encapsulated content, and normal MIME content internationalization considerations apply.

9. SECURITY CONSIDERATIONS

The message/cpim format is designed with security in mind. In particular it is designed to be used with MIME security multiparts for signatures and encryption. To this end, message/cpim messages must be considered immutable once created.

Because message/cpim messages are binary messages (due to UTF-8 encoding), if they are transmitted across non-8-bit-clean transports then the transfer agent must tunnel the entire message. Changing the message data encoding is not an allowable option. This implies that the message/cpim must be encapsulated by the message tranfer system and unencapsulated at the receiving end of the tunnel.

The resulting message must have no data loss due to the encoding and unencoding of the message. For example, an application may choose to apply the MIME base64 content-transfer-encoding to the message/cpim object to meet this requirement.

10. ACKNOWLEDGEMENTS

The authors thank the following for their helpful comments: Harald Alvestrand, Walter Houser, Leslie Daigle, [[[....]]]

11. REFERENCES

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

[2] Crocker, D., "Standard for the format of ARPA Internet text messages", RFC 822, STD 11, August 1982.

[3] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[4] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046 November 1996.

[5] Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC 2048, BCP 13, November 1996.

[6] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson, R., Crispin, M., Svanberg, P., "Report from the IAB Character Set Workshop", RFC 2130, April 1997.

[7] Alvestrand, H., "Tags for the Identification of Languages", RFC 3066, January 2001. (Defines Content-language header.)

[8] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[9] Callas, J., Donnerhacke, L., Finney, H. and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[10] Berners-Lee, T., Fielding, R.T. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[11] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", W3C recommendation: <http://www.w3.org/TR/REC-xml>, 10 February 1998.

[12] Tim Bray, Dave Hollander, and Andrew Layman "Namespaces in XML", W3C recommendation: <http://www.w3.org/TR/REC-xml-names>, 14 January 1999.

[13] "Data elements and interchange formats - Information interchange - Representation of dates and times" ISO 8601:1988(E) International Organization for Standardization June 1988.

[14] Crocker, D.H., Diacakis, A., Mazzoldi, F., Huitema, C., Klyne, G., Rose, M.T., Rosenberg, J., Sparks, R. and H. Sugano, "A Common Profile for Instant Messaging (CPIM)", draft-thenine-im-common-00 (work in progress), August 2000.

[15] Day, M., Aggarwal, S., Mohr, G., and J. Vincent "Instant Messaging / Presence Protocol Requirements" RFC 2779 February 2000.

[16] N. Freed, K. Moore "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations" RFC 2231 November 1997.

[17] D. Crocker, P. Overell "Augmented BNF for Syntax Specifications: ABNF" RFC 2234 November 1997.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee "Hypertext Transfer Protocol -- HTTP/1.1" RFC 2616 June 1999.

[19] Alvestrand, H, "IETF Policy on Character Sets and Languages", RFC 2277, BCP 18, January 1998.

[20] Freed, N., and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2278, January 1998.

[21] F. Yergeau "UTF-8, a transformation format of ISO 10646" RFC 2279 January 1998.

[22] M. Mealling "A URN Namespace for IANA Registered Protocol Elements" draft-mealling-iana-urn-00.txt (work in progress) November 2000

[23] C. Newman, G. Klyne "Date and Time on the Internet: Timestamps" draft-ietf-impp-datetime-03.txt (work in progress) May 2001.

12. AUTHORS' ADDRESSES

Derek Atkins
Telcordia Technologies
6 Farragut Ave
Somerville, MA 02144
USA.
Telephone: +1 617 623 3745
E-mail: warlord@research.telcordia.com
E-mail: warlord@alum.mit.edu

Graham Klyne
Baltimore Technologies - Content Security Group,
1310 Waterside,
Arlington Business Park
Theale
Reading, RG7 4SA
United Kingdom.
Telephone: +44 118 903 8000
Facsimile: +44 118 903 9000
E-mail: GK@ACM.ORG

Appendix A: Amendment history

00a 01-Feb-2001 Memo initially created.

00b 06-Feb-2001 Editorial review. Reworked namespace framework description. Deferred specification of mandatory headers to the application specification, allowing this document to be less application-dependent. Expanded references. Replaced some text with ABNF syntax descriptions. Reordered some major sections.

00c 07-Feb-2001 Folded in some review comments. Fix up some syntax problems. Other small editorial changes. Add some references.

01a 29-Mar-2001 Incorporate review comments. State (simply) that this is a canonical end-to-end format for the purpose of signature calculation. Defined escape mechanism for control characters. Header name parameters placed after the ":". Changed name of Date: header to DateTime:. Revised syntax to separate character-level syntax from UTF-8 octet- level syntax.

01b 30-Mar-2001 State explicitly that unrecognized header names should be ignored. Remove text about (non)significance of header order: simply say that order must be preserved.

02a 30-May-2001 Updated reference to date/time draft. Editorial changes.

03a 13-Jun-2001 Tighten up application of escape sequences.

TODO:

- confirm urn namespace for headers (currently depends on a work-in-progress).
- Complete IANA considerations

REVIEW CHECKLIST:

(Points to be checked or considered more widely on or before final review.)

- The desirability of a completely rigid syntax.
- Escape mechanism details.

Full copyright statement