Network Working Group   G. Klyne
Request for Comments: 3862   Nine by Nine
Category: Standards Track   D. Atkins
                                                         IHTFP Consulting
                                                             August 2004

          Common Presence and Instant Messaging (CPIM): Message Format

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

   This memo defines the MIME content type 'Message/CPIM', a message
   format for protocols that conform to the Common Profile for Instant
   Messaging (CPIM) specification.

Table of Contents

1. Introduction

   This memo defines the MIME content type 'Message/CPIM', a message
   format for protocols that conform to the Common Profile for Instant
   Messaging (CPIM) specification. This is a common message format for
   CPIM-compliant messaging protocols [26].

   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
   address the questions raised in section 6 of this document.

1.1. Motivation

   The Common Profile for Instant Messaging (CPIM) [26] specification
   defines a number of operations to be supported and criteria to be
   satisfied for interworking between 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 [20].

   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, whereby signatures are calculated for end-to-end security.

   The design of this message format is intended to enable security to
   be applied, while itself remaining agnostic about the specific
   security mechanisms that may be appropriate for a given application.
   For CPIM instant messaging and presence, specific security protocols
   are specified by the CPIM instant messaging [26] and CPIM presence
   [27] specifications.

   Also note that the message format described here is not itself a MIME
   data format, although it may be contained within a MIME object, and
   may contain MIME objects. See section 2 for more details.

1.2. Background

   RFC 2779 requires that an instant message can carry a MIME payload
   [1][2]; 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 RFC2822/MIME-like syntax [9], as
   protocol implementations must already contain code to parse this.

   Unfortunately, using pure RFC2822/MIME can be problematic:

o Irregular lexical structure -- RFC2822/MIME allows a number of
   optional encodings and multiple ways to encode a particular value.
   For example, RFC2822/MIME 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.

o Weak internationalization -- RFC2822/MIME requires header values
   to use 7-bit ASCII, which is problematic for encoding
   international character sets. Mechanisms for language tagging in
   RFC2822/MIME headers [16] are awkward to use and have limited
   applicability.

o 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
   attempts at achieving security (e.g., signatures), leaving
   receiving applications unable to verify the data received.

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

o Limited extensibility. (X-headers are problematic because they
   may not be standardized; this leads to situations where a header
   starts out as experimental but then finds widespread application,
   resulting in a common usage that cannot be standardized.)

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

o Some processors impose line length limitations.

The message format defined by this memo overcomes some of these
difficulties by having a simplified syntax that is generally
compatible with the format accepted by RFC2822/MIME parsers and
having a stricter syntax. It also defines mechanisms to support some
desired features not covered by the RFC2822/MIME format
specifications.

1.3. Goals

   This specification aims to satisfy the following goals:

   o a securable end-to-end format for a message (a canonical message
      format to serve as a basis for signature calculation, rather than
      specified security mechanisms).

   o independence of any specific application

   o capability of conveying a range of different address types

   o assumption of an 8-bit clean message-transfer protocol

   o evolvable: extensible by multiple parties

   o a clear separation of message metadata from message content

   o a simple, regular, easily parsed syntax

   o 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 BCP 14, RFC 2119 [4].

   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.

2. Overall Message Structure

   The CPIM message format encapsulates 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 two-part entity, where the first part
   contains the message metadata and the second part is the message
   content. The two parts are separated from the enclosing MIME header
   fields and also from each other by blank lines. The message metadata
   header information obeys more stringent syntax rules than the MIME
   message content headers that may be carried within the message.

   A 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 MIME 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 RFC 2822 message
headers in email [9], 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:

o The entire header MUST be contained on a single line. The line
   terminator is not considered part of the header value.

o Only one header per line. Multiple headers MUST NOT be included
   on a single line.

o Processors SHOULD NOT impose any line-length limitations.

o There MUST NOT be any whitespace at the beginning or end of a
   line.

o UTF-8 character encoding [13] MUST be used throughout.

o The character sequence CR,LF (13,10) MUST be used to terminate
   each line.

o The header name contains only US-ASCII characters (see section 3.1
   and section 3.6 for the specific syntax).

o 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.

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

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

o Header names MUST match exactly (i.e., "From:" and "from:" are
   different headers).

o If a header name is not recognized or not understood, the header
   should be ignored. But see also the "Require:" header (section
   4.7).

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

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

   Examples:

   To: Pooh Bear <im:pooh@100akerwood.com>
   From: <im:piglet@100akerwood.com>
   DateTime: 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.
   (Rather than invent something completely new, the escape mechanism
   has been adopted from that 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:

   o 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).

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

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

   o Single- or double-quote characters that delimit a string value
      MUST NOT be escaped.

   o 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.

   o 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 an 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 [1][2].

   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 contains two parts, a name and a value, separated by a colon
   character (':') and single space (32). It is terminated by the
   sequence CR,LF (13,10).

   Headers use UTF-8 character encoding throughout, per RFC 3629 [13].

   NOTE: in the descriptions that follow, header field names and other
   specified text values MUST be used exactly as given, using exactly
   the indicated upper- and lower- case letters. In this respect, the
   ABNF usage differs from RFC 2234 [6].

3.1. Header Names

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

   Separator characters are:

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

   NOTE: The range of allowed characters was determined by examination
   of HTTP and RFC 2822 header name formats and choosing the more
   restricted. The intent is to allow CPIM headers to follow a syntax
   that is compatible with the allowed syntax for both RFC 2822 [9] and
   HTTP [18] (including HTTP-derived protocols such as SIP [21]).

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 [15][7].

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 [10].

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' [7].

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 [22] by XML namespaces [23]. 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.

The Message/CPIM media type registration designates a default namespace for any headers that are not more explicitly associated with any namespace. In most cases, this default namespace is 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 [8]. (Relative URIs and URI-references
containing fragment identifiers MUST NOT be used for this purpose.)

Within a specific message, an '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 successfully 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 [6]. 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.

   NOTE: Specified text values MUST be used as given, using exactly the
   indicated upper- and lower-case letters. In this respect, the ABNF
   usage here differs from RFC 2234 [6].

   Collected syntax:

   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      = %x21 / %x23-27 / %x2a-2b / %x2d
                 / %x5e-60 / %x7c / %x7e
                 / ALPHA / DIGIT

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

   SEPARATORS    = "(" / ")" / "<" / ">" / "@"     ; 28/29/3c/3e/40
                 / "," / ";" / ":" / "\" / DQUOTE ; 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      = %x80-7fffffff
```

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
```

NOTE: the above syntax comes from an older version of UTF-8, and is
included for compatibility with UTF-8 software based on the earlier
specifications. Applications generating this message format SHOULD
generate UTF-8 that matches the more restricted specification in RFC
3629 [13].

4. Header Definitions

   This specification defines a core set of headers that are available
   for use by applications: an 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) those that have been based on similar headers in RFC 2822 [9],
      specified here with corresponding semantics.

   Header names and syntax are described without a namespace
   qualification, and the associated namespace URI is listed as part of
   the header specification. 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:ietf:params:cpim-headers:>, which is defined according to [12].

   NOTE: Header names and other text MUST be used as given, using
   exactly the indicated upper- and lower-case letters. In this
   respect, the ABNF usage here differs from RFC 2234 [6].

4.1. The 'From' Header

   Indicates the sender of a message.

   Header name: From

   Namespace URI:
      <urn:ietf:params:cpim-headers:>

   Syntax:
      (see also section 3.6)

      From-header = "From" ": " [Formal-name] "<" URI ">"
                         ; "From" is case-sensitive

   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:ietf:params:cpim-headers:>

    Syntax:
        (see also section 3.6)

        To-header = "To" ": " [Formal-name] "<" URI ">"
                          ; "To" is case-sensitive

    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:ietf:params:cpim-headers:>

   Syntax:
      (see also section 3.6)

      Cc-header   = "cc" ": " [Formal-name] "<" URI ">"
                        ; "cc" is case-sensitive

   Description:
      Indicates a courtesy copy 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 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: DateTime

   Namespace URI:
      <urn:ietf:params:cpim-headers:>

   Syntax:
      (see also section 3.6)

      DateTime-header = "DateTime" ": " date-time
                        ; "DateTime" is case-sensitive

    (where the syntax of 'date-time' is a profile of ISO8601 [24]
    defined in "Date and Time on the Internet" [11])

  Description:
    The 'DateTime' header supplies the 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 senders's current time when the message was transmitted,
    using ISO 8601 [24] date and time format as profiled in "Date and
    Time on the Internet: Timestamps" [11].

  Example:

    DateTime: 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:ietf:params:cpim-headers:>

  Syntax:
    (see also section 3.6)

    Subject-header = "Subject" ":" [";" Lang-param] SP *HEADERCHAR
                     ; "Subject" is case-sensitive

  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
    indicate the message subject.

  Example:

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

4.6. The 'NS' Header

   Declare a local namespace prefix.

   Header name: NS

   Namespace URI:
      <urn:ietf:params:cpim-headers:>

   Syntax:
      (see also section 3.6)

      NS-header = "NS" ": " [Name-prefix] "<" URI ">"
                        ; "NS" is case-sensitive

   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: Require

   Namespace URI:
      <urn:ietf:params:cpim-headers:>

   Syntax:
      (see also section 3.6)

      Require-header = "Require" ": " Header-name *("," Header-name)
                        ; "Require" is case-sensitive

   Description:
      Indicates a header or feature that must be implemented or
      understood by the receiver for correct message processing. See
      section 3.5 for more details.

Note that the required header or feature does not have to be used
in the message, but for brevity it is recommended that an
implementation does not issue the 'Required' header for unused
features.

Example:

Require: MyAlias.VitalHeader

5. Examples

The examples in the following sections use the per-line tags below 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 <urn:ietf:params:cpim-headers:> is
the implied default namespace for the application.

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: DateTime: 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 Esing MIME multipart/signed

   In order to secure a Message/CPIM, an application or implementation
   may use RFC 1847 [14], and some appropriate security protocols (e.g.,
   S/MIME [19] or openPGP [17]), and cryptographic scheme.

   Using S/MIME [19] and pkcs7, the above message would look like this:

```
   x: Content-Type: multipart/signed; boundary=next;
                    micalg=sha1;
                    protocol=application/pkcs7-signature
   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: DateTime: 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-signature
   x:
   x: (signature stuff)
       :
   x: --next--
```

6. Application Design Considerations

   As defined, the 'Message/CPIM' content type uses a default namespace
   URI 'urn:ietf:params-cpim-headers:', and does not define any other
   implicit namespace prefixes. Applications that have different
   requirements should define and register a different MIME media type,
   specify the required default namespace URI and define any implied
   namespace prefixes as part of the media type specification.

Applications using this specification must also specify:

o all headers that must be recognized by implementations of the
   application

o any headers that must be present in all messages created by that
   application.

o 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).

o Security mechanisms and crytography schemes to be used with the
   application, including any mandatory-to-implement security
   provisions.

The goal of providing a definitive message format to which security
mechanisms can be applied places some constraints on the design of
applications that use this message format:

o 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.

o 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 with 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 has changed and who changed it by following
   the trail of amendments. Another metaphor is including the old
   message in a new envelope.)

In chosing security mechanisms for an applications, the following IAB
survey documents may be helpful:

o Security Mechanisms for the Internet [28]

o A Survey of Authentication Mechanisms [29].

7. IANA Considerations

   This memo calls for two new IANA registrations:

   o A new MIME content-type value, Message/CPIM, per RFC 2048 [3].
      The registration template can be found in section 7.1 below.

   o A new IANA URN sub-namespace, urn:ietf:params:cpim-headers:, per
      RFC 3553 [12]. The registration template can be found in section
      7.2 below.

7.1. Registration for Message/CPIM Content Type

      To: ietf-types@iana.org

      Subject: Registration of MIME media type Message/CPIM

      MIME media type name: message

      MIME subtype name: CPIM

      Required parameters: (None)

      Optional parameters: (None)

   Encoding considerations:
      Intended to be used in 8-bit clean environments, with non-
      transformative encoding (8-bit or binary, according to the content
      contained within the message; the CPIM message headers can be
      handled in an 8-bit text environment).

      This content type could be used with a 7-bit transfer environment
      if appropriate transfer encoding is used. NOTE that for this
      purpose, enclosed MIME content MUST BE treated as opaque data and
      encoded accordingly. Any encoding must be reversed before any
      enclosed MIME content can be accessed.

   Security considerations:
      The content may contain signed data, so any transfer encoding MUST
      BE exactly reversed before the content is processed.

      See also the security considerations for email messages (RFC 2822
      [9]).

   Interoperability considerations:
      This content format is intended to be used to exchange possibly-
      secured messages between different instant messaging protocols.
      Very strict adherence to the message format (including whitespace
      usage) may be needed to achieve interoperability.

   Published specification: RFC 3862

   Applications which use this media type: Instant messaging

   Additional information:
      The default namespace URI associated with this content-type is
      'urn:ietf:params:cpim-headers:'. (See RFC 3862 for further
      details.)

      See also the Common Profile for Instant Messaging (CPIM) [26].

   Person & email address to contact for further information:
      G. Klyne, <GK-IETF@ninebynine.org>

   Intended usage: LIMITED USE

   Author/Change controller: IETF

7.2. Registration for urn:ietf:params:cpim-headers

   Registry name: cpim-headers

   Specification:
      RFC 3862. Additional values may be defined by standards track
      RFCs that update or obsolete RFC 3862.

   Repository:
      http://www.iana.org/assignments/cpim-headers

   Index value:
      The index value is a CPIM message header name, which may consist
      of a sequence from a restricted set of US-ASCII characters, as
      defined above.

   URN Formation:
      The URI for a header is formed from its name by:

      a) replacing any non-URN characters (as defined by RFC 2141 [5])
         with the corresponding '%hh' escape sequence (per RFC 2396
         [8]); and

      b) prepending the resulting string with 'urn:ietf:params:cpim-
         headers:'.

      Thus, the URI corresponding to the CPIM message header 'From:'
      would be 'urn:ietf:params:cpim-headers:From'. The URI
      corresponding to the (putative) CPIM message header 'Top&Tail'
      would be 'urn:ietf:params:cpim-headers:Top%26Tail'.

8. Internationalization Considerations

   Message headers use UTF-8 character encoding throughout; hence, they
   can convey the full UCS-4 (Unicode [30], ISO/IEC 10646 [25])
   character repertoire.

   Language tagging is provided for message headers using the "Lang"
   parameter (section 3.3).

   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 option. This implies that the
   Message/CPIM must be encapsulated by the message transfer system and
   unencapsulated at the receiving end of the tunnel.

   The resulting message must not have 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, Mark Day, Brian Raymor.

11. References

11.1. Normative References

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

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

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

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

   [5] Moats, R., "URN Syntax", RFC 2141, May 1997.

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

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

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

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

   [10] Alvestrand, H., "Tags for the Identification of Languages", BCP
        47, RFC 3066, January 2001.

   [11] Klyne, G. and C. Newman, "Date and Time on the Internet:
        Timestamps", RFC 3339, July 2002.

   [12] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF
        URN Sub-namespace for Registered Protocol Parameters", BCP 73,
        RFC 3553, June 2003.

   [13] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD
        63, RFC 3629, November 2003.

11.2. Informative References

   [14] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security
        Multiparts for MIME: Multipart/Signed and Multipart/Encrypted",
        RFC 1847, October 1995.

   [15] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson,
        R., Crispin, M., and P. Svanberg, "The Report of the IAB
        Character Set Workshop held 29 February - 1 March, 1996", RFC
        2130, April 1997.

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

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

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

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

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

   [21] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,
        Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP:
        Session Initiation Protocol", RFC 3261, June 2002.

   [22] Bray, T., Paoli, J., Sperberg-McQueen, C., and E. Maler,
        "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C
        Recommendation xml, October 2000,
        <http://www.w3.org/TR/2000/REC-xml-20001006>.

   [23] Bray, T., Hollander, D., and A. Layman, "Namespaces in XML", W3C
        Recommendation xml-names, January 1999,
        <http://www.w3.org/TR/REC-xml-names>.

   [24] International Organization for Standardization, "Data elements
        and interchange formats - Information interchange -
        Representation of dates and times", ISO Standard 8601, June
        1988.

   [25] International Organization for Standardization, "Information
        Technology - Universal Multiple-octet coded Character Set (UCS)
        - Part 1: Architecture and Basic Multilingual Plane", ISO
        Standard 10646-1, May 1993.

   [26] Peterson, J., "Common Profile for Instant Messaging (CPIM)", RFC
        3860, August 2004.

   [27] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859,
        August 2004.

   [28] Bellovin, S., Kaufman, C., and J. Schiller, "Security Mechanisms
        for the Internet", RFC 3631, December 2003.

   [29] Rescorla, E., "A Survey of Authentication Mechanisms", Work in
        Progress, March 2004.

   [30] The Unicode Consortium, "The Unicode Standard, Version 4.0",
        Addison-Wesley, Boston, MA. ISBN 0-321-18578-1, April 2003,
        <http://www.unicode.org/unicode/standard/versions/
        enumeratedversions.html#Unicode_4_0_0>.

12. Authors' Addresses

   Graham Klyne
   Nine by Nine

   EMail: GK-IETF@ninebynine.org
   URI: http://www.ninebynine.net/

   Derek Atkins
   IHTFP Consulting
   6 Farragut Ave
   Somerville, MA 02144
   USA

   Phone: +1 617 623 3745
   EMail: derek@ihtfp.com, warlord@alum.mit.edu

13. Full Copyright Statement

Acknowledgement