Network Working Group   Nathaniel Borenstein
Internet Draft   Ned Freed
<draft-ietf-822ext-mime-imb-01.txt>

Multipurpose Internet Mail Extensions (MIME) Part One:

Format of Internet Message Bodies

November 21, 1994

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months. Internet-Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet-Drafts as reference material or to cite them other than as a "working draft" or "work in progress".

To learn the current status of any Internet-Draft, please check the 1id-abstracts.txt listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

1. Abstract

STD 11, RFC 822 defines a message representation protocol specifying considerable detail about message headers, but which leaves the message content, or message body, as flat US-ASCII text. This document redefines the format of message bodies to allow multi-part textual and non-textual message bodies to be represented and exchanged without loss of information. This is based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, this document is largely orthogonal to (rather than a revision of) RFC 822.

In particular, this document is designed to provide facilities to include multiple parts in a single message, to represent body text in character sets other than US-ASCII, to represent formatted multi-font text messages, to represent non-textual material such as images and audio fragments, and generally to facilitate later extensions defining new types of Internet mail for use by cooperating mail agents.

This document does NOT extend Internet mail header fields to permit anything other than US-ASCII text data. Such extensions are the subject of [RFC-MIME-HEADERS].

This document is a revision of RFC 1521, which was a revision of RFC 1341. Significant differences from RFC 1521 are summarized in Appendix G.

2. Table of Contents

3. Introduction

Since its publication in 1982, RFC 822 [RFC-822] has defined the standard format of textual mail messages on the Internet. Its success has been such that the RFC 822 format has been adopted, wholly or partially, well beyond the confines of the Internet and the Internet SMTP transport defined by RFC 821 [RFC-821]. As the format has seen wider use, a number of limitations have proven increasingly restrictive for the user community.

RFC 822 was intended to specify a format for text messages. As such, non-text messages, such as multimedia messages that might include audio or images, are simply not mentioned. Even in the case of text, however, RFC 822 is inadequate for the needs of mail users whose languages require the use of character sets richer than US-ASCII. Since RFC 822 does not specify mechanisms for mail containing audio, video, Asian language text, or even text in most European languages, additional specifications are needed.

One of the notable limitations of RFC 821/822 based mail systems is the fact that they limit the contents of electronic mail messages to relatively short lines of 7-bit US-ASCII. This forces users to convert any non-textual data that they may wish to send into seven-bit bytes representable as printable US-ASCII characters before invoking a local mail UA (User Agent, a program with which human users send and receive mail). Examples of such encodings currently used in the Internet include pure hexadecimal, uuencode, the 3-in-4 base **64 scheme specified in RFC 1421, the Andrew Toolkit** Representation [ATK], and many others.

The limitations of RFC 822 mail become even more apparent as gateways are designed to allow for the exchange of mail messages between RFC 822 hosts and X.400 hosts. X.400 [X400] specifies mechanisms for the inclusion of non-textual body parts within electronic mail messages. The current standards for the mapping of X.400 messages to RFC 822 messages specify either that X.400 non-textual body parts must be converted to (not encoded in) IA5Text format, or that they must be discarded, notifying the RFC 822 user that discarding has occurred. This is clearly undesirable, as information that a user may wish to receive is lost. Even though a user agent may not have the capability of dealing with the non-textual

body part, the user might have some mechanism external to the UA that can extract useful information from the body part. Moreover, it does not allow for the fact that the message may eventually be gatewayed back into an X.400 message handling system (i.e., the X.400 message is "tunneled" through Internet mail), where the non-textual information would definitely become useful again.

This document describes several mechanisms that combine to solve most of these problems without introducing any serious incompatibilities with the existing world of RFC 822 mail. In particular, it describes:

(1) A MIME-Version header field, which uses a version number to declare a message to be conformant with this specification and allows mail processing agents to distinguish between such messages and those generated by older or non-conformant software, which are presumed to lack such a field.

(2) A Content-Type header field, generalized from RFC 1049 [RFC-1049], which can be used to specify the type and subtype of data in the body of a message and to fully specify the native representation (encoding) of such data.

(3) A Content-Transfer-Encoding header field, which can be used to specify an auxiliary encoding that was applied to the data in order to allow it to pass through mail transport mechanisms which may have data or character set limitations.

(4) Two additional header fields that can be used to further describe the data in a body, the Content-ID and Content-Description header fields.

All of these header fields defined in this document are subject to the general syntactic rules for header fields specified in RFC 822. In particular, all of these header fields can include RFC 822 comments, which have no semantic content and should be ignored during MIME processing.

The generalized Content-Type header field values can be used to identify both discrete and composite bodies. The following types of discrete bodies are currently defined:

(1) A "text" Content-Type value, which can be used to represent textual information in a number of character sets and formatted text description languages in a standardized manner.

(2) An "image" Content-Type value, for transmitting still image (picture) data.

(3) An "audio" Content-Type value, for transmitting audio or voice data.

(4) A "video" Content-Type value, for transmitting video or moving image data, possibly with audio as part of the composite video data format.

(5) An "application" Content-Type value, which can be used to transmit application data or binary data, and hence, among other uses, to implement an electronic mail file transfer service.

Two types of composite bodies are currently defined:

(1) A "multipart" Content-Type value, which can be used to combine several body parts, possibly of differing types of data, into a single message.

(2) A "message" Content-Type value, for encapsulating another message or part of a message.

MIME's Content-Type mechanism has been carefully designed to be extensible, and it is expected that the set of content-type/subtype pairs and their associated parameters will grow significantly with time. Several other MIME entities, most notably the list of the name of character sets registered for MIME usage, are likely to have new values defined over time. In order to ensure that the set of such values is developed in an orderly, well-specified, and public manner, MIME sets up a registration process which uses the Internet Assigned Numbers Authority (IANA) as a central registry for MIME's extension areas. The registration process is described in RFC REG [RFC-REG].

Finally, to specify and promote interoperability, Appendix A of this document provides a basic applicability statement for a subset of the above mechanisms that defines a minimal level

of "conformance" with this document.

HISTORICAL NOTE: Several of the mechanisms described in this document may seem somewhat strange or even baroque at first reading. It is important to note that compatibility with existing standards AND robustness across existing practice were two of the highest priorities of the working group that developed this document. In particular, compatibility was always favored over elegance.

MIME was first defined and published as RFC 1341 [RFC-1341] and RFC1342 [RFC-1342], then revised in RFC 1521 [RFC-1521] and RFC 1522 [RFC-1522]. This document is a relatively minor updating of RFC 1521, and is intended to supersede it. The companion document RFC MIME-HEADERS [RFC-MIME-HEADERS] in turn supersedes RFC 1522.

The differences between this document and RFC 1521 are summarized in Appendix G. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. RFC 822 and RFC 1123 [RFC-1123] also provide essential background for MIME since no conforming implementation of MIME can violate them. In addition, several other informational RFC documents will be of interest to the MIME implementor, in particular RFC 1344 [RFC-1344], RFC 1345 [RFC-1345], and RFC 1524 [RFC-1524].

4. Notations, Conventions, and Generic BNF Grammar

Although the mechanisms specified in this document are all described in prose, most are also described formally in the augmented BNF notation of RFC 822. Implementors will need to be familiar with this notation in order to understand this specification, and are referred to RFC 822 for a complete explanation of the augmented BNF notation.

Some of the augmented BNF in this document makes reference to syntactic entities that are defined in RFC 822 and not in this document. A complete formal grammar, then, is obtained by Appendix D of this document, the collected grammar, with the BNF of RFC 822 plus the modifications to RFC 822 defined in RFC 1123, which specifically changes the syntax for `return', `date' and `mailbox'.

The term CRLF, in this document, refers to the sequence of the two US-ASCII characters CR (decimal value 13) and LF (decimal value 10) which, taken together, in this order, denote a line break in RFC 822 mail.

The term "character set" is used in this document to refer to a method used with one or more tables to convert a sequence of octets into a sequence of characters. Note that unconditional conversion in the other direction is not required, in that not all characters may be available in a given character set and a character set may provide more than one sequence of octets to represent a particular character. This definition is intended to allow various kinds of character encodings, from simple single-table mappings such as US-ASCII to complex table switching methods such as those that use ISO 2022's techniques. However, the definition associated with a MIME character set name must fully specify the mapping to be performed from octets to characters. In particular, use of external profiling information to determine the exact mapping is not permitted.

The term "message", when not further qualified, means either the (complete or "top-level") message being transferred on a network, or a message encapsulated in a body part of type "message".

The term "body part", in this document, refers to either the a single part message or one of the parts in the body of a

multipart entity. A body part has a header and a body, so it makes sense to speak about the body of a body part.

The term "entity", in this document, means either a message or a body part. All kinds of entities share the property that they have a header and a body.

The term "body", when not further qualified, means the body of an entity, that is the body of either a message or of a body part.

NOTE: The previous four definitions are clearly circular. This is unavoidable, since the overall structure of a MIME message is indeed recursive.

"7bit data" refers to data that is all represented as short lines of US-ASCII. CR (decimal value 13) and LF (decimal value 10) characters only occur as part of CRLF line separation sequences and no NULs (US-ASCII value 0) are allowed.

(1) "8bit data" refers to data that is all represented as short lines, but there may be non-US-ASCII characters (octets with the high-order bit set) present. As with "7bit data" CR and LF characters only occur as part of CRLF line separation sequences and no NULs are allowed.

(2) "Binary data" refers to data where any sequence of octets whatsoever is allowed.

"Lines" are defined as sequences of octets separated by a CRLF sequences. This is consistent with both RFC 821 and RFC 822. Lines in MIME bodies must also be terminated with a CRLF, but the terminating CRLF on the last line of the body may properly be part of a subsequent boundary marker rather than being part of the body itself.

In this document, all numeric and octet values are given in decimal notation. All Content-Type values, subtypes, and parameter names as defined in this document are case-insensitive. However, parameter values are case-sensitive unless otherwise specified for the specific parameter.

FORMATTING NOTE: Notes, such at this one, provide additional nonessential information which may be skipped by the reader

without missing anything essential. The primary purpose of these non-essential notes is to convey information about the rationale of this document, or to place this document in the proper historical or evolutionary context. Such information may in particular be skipped by those who are focused entirely on building a conformant implementation, but may be of use to those who wish to understand why certain design choices were made.

5. MIME Header Fields

MIME defines a number of new RFC 822 header fields that are used to describe the content of messages. These header fields occur in two contexts:

(1) As part of a regular RFC 822 message header.

(2) In a MIME body part header within a multipart construct.

The formal definition of these header fields is as follows:

```
MIME-message-headers := fields
                        version CRLF
                        [ content CRLF ]
                        [ encoding CRLF ]
                        [ id CRLF ]
                        [ description CRLF ]
                        *( mime-extension-field CRLF )
                        ; The ordering of the header
                        ; fields implied by this BNF
                        ; definition should be ignored

MIME-part-headers := [ content CRLF ]
                     [ encoding CRLF ]
                     [ id CRLF ]
                     [ description CRLF ]
                     *( mime-extension-field CRLF )
                     ; The ordering of the header
                     ; fields implied by this BNF
                     ; definition should be ignored
```

The syntax of the various specific MIME header fields will be described in the following sections.

5.1. MIME-Version Header Field

Since RFC 822 was published in 1982, there has really been only one format standard for Internet messages, and there has been little perceived need to declare the format standard in use. This document is an independent document that complements RFC 822. Although the extensions in this document have been defined in such a way as to be compatible with RFC

822, there are still circumstances in which it might be desirable for a mail-processing agent to know whether a message was composed with the new standard in mind.

Therefore, this document defines a new header field, "MIME-Version", which is to be used to declare the version of the Internet message body format standard in use.

Messages composed in accordance with this document MUST include such a header field, with the following verbatim text:

```
MIME-Version: 1.0
```

The presence of this header field is an assertion that the message has been composed in compliance with this document.

Since it is possible that a future document might extend the message format standard again, a formal BNF is given for the content of the MIME-Version field:

```
version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

Thus, future format specifiers, which might replace or extend "1.0", are constrained to be two integer fields, separated by a period. If a message is received with a MIME-version value other than "1.0", it cannot be assumed to conform with this specification.

Note that the MIME-Version header field is required at the top level of a message. It is not required for each body part of a multipart entity. It is required for the embedded headers of a body of type "message" if and only if the embedded message is itself claimed to be MIME-conformant.

It is not possible to fully specify how a mail reader that conforms with MIME as defined in this document should treat a message that might arrive in the future with some value of MIME-Version other than "1.0".

It is also worth noting that version control for specific content-types is not accomplished using the MIME-Version mechanism. In particular, some formats (such as application/postscript) have version numbering conventions that are internal to the document format. Where such conventions exist, MIME does nothing to supersede them. Where

no such conventions exist, a MIME type might use a "version" parameter in the content-type field if necessary.

NOTE TO IMPLEMENTORS: When checking MIME-Version values any RFC 822 comment strings that are present must be ignored. In particular, the following four MIME-Version fields are equivalent:

```
MIME-Version: 1.0

MIME-Version: 1.0 (produced by MetaSend Vx.x)

MIME-Version: (produced by MetaSend Vx.x) 1.0

MIME-Version: 1.(produced by MetaSend Vx.x)0
```

5.2. Content-Type Header Field

The purpose of the Content-Type field is to describe the data contained in the body fully enough that the receiving user agent can pick an appropriate agent or mechanism to present the data to the user, or otherwise deal with the data in an appropriate manner.

HISTORICAL NOTE: The Content-Type header field was first defined in RFC 1049. RFC 1049 Content-types used a simpler and less powerful syntax, but one that is largely compatible with the mechanism given here.

The Content-Type header field is used to specify the nature of the data in the body of an entity, by giving type and subtype identifiers, and by providing auxiliary information that may be required for certain types. After the type and subtype names, the remainder of the header field is simply a set of parameters, specified in an attribute/value notation. The ordering of parameters is not significant.

In general, the top-level Content-Type is used to declare the general type of data, while the subtype specifies a specific format for that type of data. Thus, a Content-Type of "image/xyz" is enough to tell a user agent that the data is an image, even if the user agent has no knowledge of the specific image format "xyz". Such information can be used, for example, to decide whether or not to show a user the raw data

from an unrecognized subtype -- such an action might be reasonable for unrecognized subtypes of text, but not for unrecognized subtypes of image or audio. For this reason, registered subtypes of text, image, audio, and video should not contain embedded information that is really of a different type. Such compound formats should be represented using the "multipart" or "application" types.

Parameters are modifiers of the content-subtype, and as such do not fundamentally affect the nature of the content. The set of meaningful parameters depends on the content-type and subtype. Most parameters are associated with a single specific subtype. However, a given top-level content-type may define parameters which are applicable to any subtype of that type. For example, the "charset" parameter is applicable to any subtype of "text", while the "boundary" parameter is required for any subtype of the "multipart" content-type.

There are NO globally-meaningful parameters that apply to all content-types. Truly global mechanisms are best addressed, in the MIME model, by the definition of additional Content-* header fields.

An initial set of seven top-level Content-Types is defined by this document. Five of these are discrete types whose content is essentially opaque as far as MIME processing is concerned. The remaining two are composite types whose contents require additional handling by MIME processors.

This set of top-level Content-Types is intended to be substantially complete. It is expected that additions to the larger set of supported types can generally be accomplished by the creation of new subtypes of these initial types. In the future, more top-level types may be defined only by a standards-track extension to this standard. If another top-level type is to be used for any reason, it must be given a name starting with "X-" to indicate its non-standard status and to avoid a potential conflict with a future official name.

5.2.1. Syntax of the Content-Type Header Field

In the Augmented BNF notation of RFC 822, a Content-Type header field value is defined as follows:

```
  content :=  "Content-Type" ":" type "/" subtype
              *(";" parameter)
              ; Matching of type and subtype is
              ; ALWAYS case-insensitive

  type := discrete-type / composite-type

  discrete-type := "text" / "image" / "audio" / "video" /
                   "application" / extension-token

  composite-type := "message" / "multipart" / extension-token

  extension-token := iana-token / ietf-token / x-token

  iana-token := <a publicly-defined extension token,
                registered with IANA, as specified in
                RFC REG [RFC-REG]>

  ietf-token := <a publicly-defined extension token,
                initially registered with IANA and
                subsequently standardized by the IETF>

  x-token := <The two characters "X-" or "x-" followed, with
              no intervening white space, by any token>

  subtype := extension-token

  parameter := attribute "=" value

  attribute := token

  value := token / quoted-string

  token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
              or tspecials>

  tspecials :=  "(" / ")" / "<" / ">" / "@" /
                "," / ";" / ":" / "\" / <">
                "/" / "[" / "]" / "?" / "="
                ; Must be in quoted-string,
                ; to use within parameter values
```

Note that the definition of "tspecials" is the same as the RFC **822 definition of "specials" with the addition of the three** characters "/", "?", and "=", and the removal of ".".

Note also that a subtype specification is MANDATORY -- it may not be omitted from a Content-Type header field. As such, there are no default subtypes.

The type, subtype, and parameter names are not case sensitive. For example, TEXT, Text, and TeXt are all equivalent top-level Content Types. Parameter values are normally case sensitive, but sometimes are interpreted in a case-insensitive fashion, depending on the intended use. (For example, multipart boundaries are case-sensitive, but the "access-type" parameter for message/External-body is not case-sensitive.)

Note that the value of a quoted string parameter does not include the quotes. That is, the quotation marks in a quoted-string are not a part of the value of the parameter, but are merely used to delimit that parameter value. In addition, comments are allowed in accordance with RFC 822 rules for structured header fields. Thus the following two forms

```
Content-type: text/plain; charset=us-ascii (Plain text)

Content-type: text/plain; charset="us-ascii"
```

are completely equivalent.

Beyond this syntax, the only syntactic constraint on the definition of subtype names is the desire that their uses must not conflict. That is, it would be undesirable to have two different communities using "Content-Type: application/foobar" to mean two different things. The process of defining new content-subtypes, then, is not intended to be a mechanism for imposing restrictions, but simply a mechanism for publicizing the usages. There are, therefore, two acceptable mechanisms for defining new Content-Type subtypes:

(1)   Private values (starting with "X-") may be defined bilaterally between two cooperating agents without outside registration or standardization.

(2)   New standard values MUST be documented, registered with, and approved by IANA, as described in RFC REG.

5.2.2. Definition of a Top-Level Content-Type

The definition of a top-level content-type consists of:

(1) a name and a description of the type, including criteria for whether a particular type would qualify under that type,

(2) the names and definitions of parameters, if any, which are defined for all subtypes of that type (including whether such parameters are required or optional),

(3) how a user agent and/or gateway should handle unknown subtypes of this type,

(4) general considerations on gatewaying objects of this top-level type, if any, and

(5) any restrictions on content-transfer-encodings for objects of this top-level type.

5.2.3. Initial Set of Top-Level Content-Types

The initial seven standard top-level Content-Types are detailed in the bulk of this document. The five discrete top-level Content-Types are:

(1) text -- textual information. The subtype "plain" in particular indicates plain (unformatted) text. No special software is required to get the full meaning of the text, aside from support for the indicated character set. Other subtypes are to be used for enriched text in forms where application software may enhance the appearance of the text, but such software must not be required in order to get the general idea of the content. Possible subtypes thus include any word processor format that can be read without resorting to software that understands the format. In particular, formats that employ embeddded binary formatting information are not considered directly readable. A very simple and portable subtype, richtext, was defined in RFC 1341 [RFC-1341], with a further revision in RFC 1563 [RFC-1563] under the name "enriched".

(2) image -- image data. Image requires a display device (such as a graphical display, a graphics printer, or a FAX machine) to view the information. Initial subtypes are defined for two widely-used image formats, jpeg and gif.

(3) audio -- audio data. Audio requires an audio output device (such as a speaker or a telephone) to "display" the contents. An initial subtype "basic" is defined in this document.

(4) video -- video data. Video requires the capability to display moving images, typically including specialized hardware and software. An initial subtype "mpeg" is defined in this document.

(5) application -- some other kind of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The subtype "octet-stream" is to be used in the case of uninterpreted binary data, in which case the simplest recommended action is to offer to write the information into a file for the user. The "PostScript" subtype is also defined for the transport of PostScript material. Other expected uses for "application" include spreadsheets, data for mail-based scheduling systems, and languages for "active" (computational) email, and word processing formats that are not directly readable. Note that security considerations may exist for some types of application data, most notably application/PostScript and any form of active mail. These issues are discussed later in this document.

The two composite top-level Content-Types are:

(1) multipart -- data consisting of multiple parts of independent data types. Four subtypes are initially defined, including the basic "mixed" subtype specifying a generic mixed set of parts, "alternative" for representing the same data in multiple formats, "parallel" for parts intended to be viewed simultaneously, and "digest" for multipart entities in which each part is of type "message".

(2) message -- an encapsulated message. A body of Content-Type "message" is itself all or part of some kind of message object. Such objects may in turn contain other messages and body parts of their own. The "rfc822" subtype is used when the encpsulated content is itself an RFC 822 message. The "partial" subtype is defined for partial RFC 822 messages, to permit the fragmented transmission of bodies that are thought to be too large to be passed through mail transport facilities in one piece. Another subtype, "external-body", is defined for specifying large bodies by reference to an external data source.

Default RFC 822 messages without a MIME Content-Type header are taken by this protocol to be plain text in the US-ASCII character set, which can be explicitly specified as:

```
Content-type: text/plain; charset=us-ascii
```

This default is assumed if no Content-Type is specified. In the presence of a MIME-Version header field, a receiving User Agent can also assume that plain US-ASCII text was the sender's intent. Plain US-ASCII text must still be assumed in the absence of a MIME-Version specification, but the sender's intent might have been otherwise.

RATIONALE: In the absence of any Content-Type header field or MIME-Version header field, it is impossible to be certain that a message is actually text in the US-ASCII character set, since it might well be a message that, using some set of nonstandard conventions that predate this document, includes text in another character set or non-textual data in a manner that cannot be automatically recognized (e.g., a uuencoded compressed UNIX tar file). Although there is no fully acceptable alternative to treating such untyped messages as "text/plain; charset=us-ascii", implementors should remain aware that if a message lacks both the MIME-Version and the Content-Type header fields, it may in practice contain almost anything.

It should be noted that the list of Content-Type values given here may be augmented in time, via the mechanisms described above, and that the set of subtypes is expected to grow substantially.

When a mail reader encounters mail with an unknown Content-type value, it should generally treat it as equivalent to "application/octet-stream", as described later in this document.

5.3. Content-Transfer-Encoding Header Field

Many Content-Types which could be usefully transported via email are represented, in their "natural" format, as 8-bit character or binary data. Such data cannot be transmitted over some transport protocols. For example, RFC 821 (SMTP) restricts mail messages to 7-bit US-ASCII data with lines no longer than 1000 characters.

It is necessary, therefore, to define a standard mechanism for encoding such data into a 7-bit short-line format. Proper labelling of unencoded material in less restrictive formats for direct use over less restrictive transports is also desireable. This document specifies that such encodings will be indicated by a new "Content-Transfer-Encoding" header field. This field has not been defined by any previous standard.

5.3.1. Content-Transfer-Encoding Syntax

The Content-Transfer-Encoding field's value is a single token specifying the type of encoding, as enumerated below. Formally:

```
  encoding := "Content-Transfer-Encoding" ":" mechanism

  mechanism := "7bit" / "8bit" / "binary" /
               "quoted-printable" / "base64" /
               ietf-token / x-token
```

These values are not case sensitive -- Base64 and BASE64 and bAsE64 are all equivalent. An encoding type of 7BIT requires that the body is already in a 7-bit mail-ready representation. This is the default value -- that is, "Content-Transfer-Encoding: 7BIT" is assumed if the Content-Transfer-Encoding header field is not present.

5.3.2. Content-Transfer-Encoding Semantics

This single token actually provides two pieces of information. It specifies what sort of encoding transformation the body was subjected to, and it specifies what the domain of the result is.

Three transformations are currently defined: identity, the "quoted-printable" encoding, and the "base64" encoding. The domains are "binary", "8bit" and "7bit".

The values "7bit", "8bit", and "binary" all mean that the identity (i.e. NO) encoding transformation has been performed. As such, they serve simply as indicators of the domain of the body part data, and provide useful information about the sort of encoding that might be needed for transmission in a given transport system. The terms "7bit data", "8bit data", and "binary data" are all defined in Section 4.

The quoted-printable and base64 encodings transform their input from an arbitrary domain into material in the "7bit" domain, thus making it safe to carry over restricted transports. The specific definition of the transformations are given below.

The proper Content-Transfer-Encoding label must always be used. Labelling unencoded data containing 8-bit characters as "7bit" is not allowed, nor is labelling unencoded non-line-oriented data as anything other than "binary" allowed.

Unlike Content-Type subtypes, a proliferation of Content-Transfer-Encoding values is both undesirable and unnecessary. However, establishing only a single transformation into the "7bit" domain does not seem possible. There is a tradeoff between the desire for a compact and efficient encoding of largely-binary data and the desire for a readable encoding of data that is mostly, but not entirely, 7-bit. For this reason, at least two encoding mechanisms are necessary: a "readable" encoding (quoted-printable) and a "dense" encoding (base64).

Mail transport for unencoded 8-bit data is defined in RFC 1652 [RFC-1652]. As of the publication of this document, there are no standardized Internet mail transports for which it is legitimate to include unencoded binary data in mail bodies.

Thus there are no circumstances in which the "binary" Content-Transfer-Encoding is actually valid on the Internet. However, in the event that binary mail transport becomes a reality in Internet mail, or when this document is used in conjunction with any other binary-capable transport mechanism, binary bodies should be labelled as such using this mechanism.

NOTE: The five values defined for the Content-Transfer-Encoding field imply nothing about the Content-Type other than the algorithm by which it was encoded or the transport system requirements if unencoded.

Implementors may, if necessary, define new Content-Transfer-Encoding values, but must use an x-token, which is a name prefixed by "X-", to indicate its non-standard status, e.g., "Content-Transfer-Encoding: x-my-new-encoding". However, unlike Content-Types and subtypes, the creation of new Content-Transfer-Encoding values is STRONGLY discouraged, as it seems likely to hinder interoperability with little potential benefit. Such use is therefore allowed only as the result of an agreement between cooperating user agents.

If a Content-Transfer-Encoding header field appears as part of a message header, it applies to the entire body of that message. If a Content-Transfer-Encoding header field appears as part of a body part's headers, it applies only to the body of that body part. If an entity is of type "multipart" the Content-Transfer-Encoding is not permitted to have any value other than "7bit", "8bit" or "binary". Even more severe restrictions apply to some subtypes of the "message" type.

It should be noted that email is character-oriented, so that the mechanisms described here are mechanisms for encoding arbitrary octet streams, not bit streams. If a bit stream is to be encoded via one of these mechanisms, it must first be converted to an 8-bit byte stream using the network standard bit order ("big-endian"), in which the earlier bits in a stream become the higher-order bits in a 8-bit byte. A bit stream not ending at an 8-bit boundary must be padded with zeroes. This document provides a mechanism for noting the addition of such padding in the case of the application/octet-stream Content-Type, which has a "padding" parameter.

The encoding mechanisms defined here explicitly encode all data in US-ASCII. Thus, for example, suppose an entity has header fields such as:

```
Content-Type: text/plain; charset=ISO-8859-1
Content-transfer-encoding: base64
```

This must be interpreted to mean that the body is a base64 US-ASCII encoding of data that was originally in ISO-8859-1, and will be in that character set again after decoding.

The following sections will define the two standard encoding mechanisms. The definition of new content-transfer-encodings is explicitly discouraged and should only occur when absolutely necessary. All content-transfer-encoding namespace except that beginning with "X-" is explicitly reserved to the IANA for future use. Private agreements about content-transfer-encodings are also explicitly discouraged.

Certain Content-Transfer-Encoding values may only be used on certain Content-Types. In particular, it is EXPRESSLY FORBIDDEN to use any encodings other than "7bit", "8bit", or "binary" with any composite Content-Type, i.e. one that recursively includes other Content-Type fields. Currently the only composite Content-Types are "multipart" and "message". All encodings that are desired for bodies of type multipart or message must be done at the innermost level, by encoding the actual body that needs to be encoded.

It should also be noted that, by definition, if a composite entity has a transfer-encoding value such as "7bit", but one of the enclosed parts has a less restrictive value such as "8bit", then either the outer "7bit" labelling is in error, because 8-bit data are included, or the inner "8bit" labelling placed an unnecessarily high demand on the transport system because the actual included data were actually 7-bit-safe.

NOTE ON ENCODING RESTRICTIONS: Though the prohibition against using content-transfer-encodings on composite body data may seem overly restrictive, it is necessary to prevent nested encodings, in which data are passed through an encoding algorithm multiple times, and must be decoded multiple times in order to be properly viewed. Nested encodings add considerable complexity to user agents: Aside from the obvious efficiency problems with such multiple encodings, they

can obscure the basic structure of a message. In particular, they can imply that several decoding operations are necessary simply to find out what types of bodies a message contains. Banning nested encodings may complicate the job of certain mail gateways, but this seems less of a problem than the effect of nested encodings on user agents.

NOTE ON THE RELATIONSHIP BETWEEN CONTENT-TYPE AND CONTENT-TRANSFER-ENCODING: It may seem that the Content-Transfer-Encoding could be inferred from the characteristics of the Content-Type that is to be encoded, or, at the very least, that certain Content-Transfer-Encodings could be mandated for use with specific Content-Types. There are several reasons why this is not the case. First, given the varying types of transports used for mail, some encodings may be appropriate for some Content-Type/transport combinations and not for others. (For example, in an 8-bit transport, no encoding would be required for text in certain character sets, while such encodings are clearly required for 7-bit SMTP.)

Second, certain Content-Types may require different types of transfer encoding under different circumstances. For example, many PostScript bodies might consist entirely of short lines of 7-bit data and hence require no encoding at all. Other PostScript bodies (especially those using Level 2 PostScript's binary encoding mechanism) may only be reasonably represented using a binary transport encoding. Finally, since Content-Type is intended to be an open-ended specification mechanism, strict specification of an association between Content-Types and encodings effectively couples the specification of an application protocol with a specific lower-level transport. This is not desirable since the developers of a Content-Type should not have to be aware of all the transports in use and what their limitations are.

NOTE ON TRANSLATING ENCODINGS: The quoted-printable and base64 encodings are designed so that conversion between them is possible. The only issue that arises in such a conversion is the handling of line breaks. When converting from quoted-printable to base64 a line break must be converted into a CRLF sequence. Similarly, a CRLF sequence in base64 data must be converted to a quoted-printable line break, but ONLY when converting text data.

NOTE ON CANONICAL ENCODING MODEL: There was some confusion, in earlier drafts of this document, regarding the model for when email data was to be converted to canonical form and encoded, and in particular how this process would affect the treatment of CRLFs, given that the representation of newlines varies greatly from system to system, and the relationship between content-transfer-encodings and character sets. A canonical model for encoding is presented as Appendix F for this reason.

5.3.3. Quoted-Printable Content-Transfer-Encoding

The Quoted-Printable encoding is intended to represent data that largely consists of octets that correspond to printable characters in the US-ASCII character set. It encodes the data in such a way that the resulting octets are unlikely to be modified by mail transport. If the data being encoded are mostly US-ASCII text, the encoded form of the data remains largely recognizable by humans. A body which is entirely US-ASCII may also be encoded in Quoted-Printable to ensure the integrity of the data should the message pass through a character-translating, and/or line-wrapping gateway.

In this encoding, octets are to be represented as determined by the following rules:

(1) (General 8-bit representation) Any octet, except those indicating a line break according to the newline convention of the canonical (standard) form of the data being encoded, may be represented by an "=" followed by a two digit hexadecimal representation of the octet's value. The digits of the hexadecimal alphabet, for this purpose, are "0123456789ABCDEF". Uppercase letters must be used when sending hexadecimal data, though a robust implementation may choose to recognize lowercase letters on receipt. Thus, for example, the decimal value 12 (US-ASCII form feed) can be represented by "=0C", and the decimal value 61 (US-ASCII EQUAL SIGN) can be represented by "=3D". This rule must be followed except when the following rules allow an alternative encoding.

(2) (Literal representation) Octets with decimal values of 33 through 60 inclusive, and 62 through 126, inclusive,

MAY be represented as the US-ASCII characters which correspond to those octets (EXCLAMATION POINT through LESS THAN, and GREATER THAN through TILDE, respectively).

(3) (White Space) Octets with values of 9 and 32 MAY be represented as US-ASCII TAB (HT) and SPACE characters, respectively, but MUST NOT be so represented at the end of an encoded line. Any TAB (HT) or SPACE characters on an encoded line MUST thus be followed on that line by a printable character. In particular, an "=" at the end of an encoded line, indicating a soft line break (see rule #5) may follow one or more TAB (HT) or SPACE characters. It follows that an octet with decimal value 9 or 32 appearing at the end of an encoded line must be represented according to Rule #1. This rule is necessary because some MTAs (Message Transport Agents, programs which transport messages from one user to another, or perform a part of such transfers) are known to pad lines of text with SPACEs, and others are known to remove "white space" characters from the end of a line. Therefore, when decoding a Quoted-Printable body, any trailing white space on a line must be deleted, as it will necessarily have been added by intermediate transport agents.

(4) (Line Breaks) A line break in a text body, represented as a CRLF sequence in the text canonical form, must be represented by a (RFC 822) line break, which is also a CRLF sequence, in the Quoted-Printable encoding. Since the canonical representation of types other than text do not generally include the representation of line breaks as CRLF sequences, no hard line breaks (i.e. line breaks that are intended to be meaningful and to be displayed to the user) should occur in the quoted-printable encoding of such types. Sequences like "=0D", "=0A", "=0A=0D" and "=0D=0A" will routinely appear in non-text data represented in quoted-printable, of course.

Note that many implementations may elect to encode the local representation of various content types directly, as described in Appendix F. In particular, this may apply to plain text material on systems that use newline conventions other than CRLF delimiters. Such

an implementation is permissible, but the generation of line breaks must be generalized to account for the case where alternate representations of newline sequences are used.

(5) (Soft Line Breaks) The Quoted-Printable encoding REQUIRES that encoded lines be no more than 76 characters long. If longer lines are to be encoded with the Quoted-Printable encoding, "soft" line breaks must be used. An equal sign as the last character on a encoded line indicates such a non-significant ("soft") line break in the encoded text.

Thus if the "raw" form of the line is a single unencoded line that says:

```
  Now's the time for all folk to come to the aid of their country.
```

This can be represented, in the Quoted-Printable encoding, as:

```
  Now's the time =
  for all folk to come=
   to the aid of their country.
```

This provides a mechanism with which long lines are encoded in such a way as to be restored by the user agent. The 76 character limit does not count the trailing CRLF, but counts all other characters, including any equal signs.

Since the hyphen character ("-") is represented as itself in the Quoted-Printable encoding, care must be taken, when encapsulating a quoted-printable encoded body in a multipart entity, to ensure that the encapsulation boundary does not appear anywhere in the encoded body. (A good strategy is to choose a boundary that includes a character sequence such as "=_" which can never appear in a quoted-printable body. See the definition of multipart messages later in this document.)

NOTE: The quoted-printable encoding represents something of a compromise between readability and reliability in transport. Bodies encoded with the quoted-printable encoding will work reliably over most mail gateways, but may not work perfectly over a few gateways, notably those involving translation into EBCDIC. A higher level of confidence is offered by the base64 Content-Transfer-Encoding. A way to get reasonably reliable

transport through EBCDIC gateways is to also quote the US-ASCII characters

```
  !"#$@[\]^`{|}~
```

according to rule #1. See Appendix B for more information.

Because quoted-printable data is generally assumed to be line-oriented, it is to be expected that the representation of the breaks between the lines of quoted printable data may be altered in transport, in the same manner that plain text mail has always been altered in Internet mail when passing between systems with differing newline conventions. If such alterations are likely to constitute a corruption of the data, it is probably more sensible to use the base64 encoding rather than the quoted-printable encoding.

WARNING TO IMPLEMENTORS: If binary data are encoded in quoted-printable, care must be taken to encode CR and LF characters as "=0D" and "=0A", respectively. In particular, a CRLF sequence in binary data should be encoded as "=0D=0A". Otherwise, if CRLF were represented as a hard line break, it might be incorrectly decoded on platforms with different line break conventions.

For formalists, the syntax of quoted-printable data is described by the following grammar:

```
  quoted-printable := ([*(ptext / SPACE / TAB) ptext]
                       ["="] CRLF)
                      ; Maximum line length of 76 characters
                      ; excluding CRLF

  ptext := octet / safe-char

  safe-char := <any US-ASCII character except "=",
                SPACE, or TAB>
               ; Characters not listed as "mail-safe" in
               ; Appendix B are also not recommended.

  octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
           ; Octet must be used for characters > 127, =,
           ; SPACE, or TAB, and is recommended for any
           ; characters not listed in Appendix B as
           ; "mail-safe".
```

IMPORTANT NOTE: The addition of LWSP between the elements shown in this BNF is NOT allowed since this BNF does not specify a structured header field.

5.3.4. Base64 Content-Transfer-Encoding

The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the unencoded data. This encoding is virtually identical to the one used in Privacy Enhanced Mail (PEM) applications, as defined in RFC 1421 [RFC-1421].

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

NOTE: This subset has the important property that it is represented identically in all versions of ISO 646, including US-ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. Other popular encodings, such as the encoding used by the uuencode utility and the base85 encoding specified as part of Level 2 PostScript, do not share these properties, and thus do not fulfill the portability requirements a binary transport encoding for mail must meet.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8-bit byte, and the eighth bit will be the low-order bit in the first 8-bit byte, and so on.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified

in Table 1, below, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., ".", CR, LF) and to the encapsulation boundaries defined in this document (e.g., "-").

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The encoded output stream must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Table 1 must be ignored by decoding software. In base64 data, characters other than those in Table 1, line breaks, and other white space probably indicate a transmission error, about which a warning message or even a message rejection might be appropriate under some circumstances.

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character. Since all base64 input is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output

will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Because it is used only for padding at the end of the data, the occurrence of any "=" characters may be taken as evidence that the end of the data has been reached (without truncation in transit). No such assurance is possible, however, when the number of octets transmitted was a multiple of three.

Any characters outside of the base64 alphabet are to be ignored in base64-encoded data. The same applies to any invalid sequence of characters in the base64 encoding, such as "====="

Care must be taken to use the proper octets for line breaks if base64 encoding is applied directly to text material that has not been converted to canonical form. In particular, text line breaks must be converted into CRLF sequences prior to base64 encoding. The important thing to note is that this may be done directly by the encoder rather than in a prior canonicalization step in some implementations.

NOTE: There is no need to worry about quoting apparent encapsulation boundaries within base64-encoded parts of multipart entities because no hyphen characters are used in the base64 encoding.

5.4. Content-ID Header Field

In constructing a high-level user agent, it may be desirable to allow one body to make reference to another. Accordingly, bodies may be labelled using the "Content-ID" header field, which is syntactically identical to the "Message-ID" header field:

```
id := "Content-ID" ":" msg-id
```

Like the Message-ID values, Content-ID values must be generated to be world-unique.

The Content-ID value may be used for uniquely identifying MIME entities in several contexts, particularly for caching data referenced by the message/external-body mechanism. Although the Content-ID header is generally optional, its use is MANDATORY in implementations which generate data of the optional MIME Content-type "message/external-body". That is, each message/external-body entity must have a Content-ID field to permit caching of such data.

It is also worth noting that the Content-ID value has special semantics in the case of the multipart/alternative content-type. This is explained in the section of this document dealing with multipart/alternative.

5.5. Content-Description Header Field

The ability to associate some descriptive information with a given body is often desirable. For example, it may be useful to mark an "image" body as "a picture of the Space Shuttle Endeavor." Such text may be placed in the Content-Description header field. This header field is always optional.

```
  description := "Content-Description" ":" *text
```

The description is presumed to be given in the US-ASCII character set, although the mechanism specified in RFC MIME-HEADERS [RFC-MIME-HEADERS] may be used for non-US-ASCII Content-Description values.

5.6. Additional MIME Header Fields

Future documents may elect to define additional MIME header fields for various purposes. Any new header field that further describes the content of a message should begin with the string "Content-" to allow such fields which appear in a message header to be distinguished from ordinary RFC 822 message header fields.

```
  MIME-extension-field := <Any RFC 822 header field which
                           begins with the string
                           "Content-">
```

6. Predefined Content-Type Values

This document defines seven initial Content-Type values and an extension mechanism for private or experimental types. Further standard types must be defined by new published specifications. It is expected that most innovation in new types of mail will take place as subtypes of the seven types defined here. The most essential characteristics of the seven content-types are summarized in Appendix E.

6.1. Discrete Content-Type Values

Five of the seven initial Content-Type values refer to discrete bodies. The content of such entities is handled by non-MIME mechanisms; they are opaque to MIME processors.

6.1.1. Text Content-Type

The text Content-Type is intended for sending material which is principally textual in form. A "charset" parameter may be used to indicate the character set of the body text for some text subtypes, notably including the subtype "text/plain", which indicates plain (unformatted) text. The default Content-Type for Internet mail if none is specified is "text/plain; charset=us-ascii".

Beyond plain text, there are many formats for representing what might be known as "extended text" -- text with embedded formatting and presentation information. An interesting characteristic of many such representations is that they are to some extent readable even without the software that interprets them. It is useful, then, to distinguish them, at the highest level, from such unreadable data as images, audio, or text represented in an unreadable form. In the absence of appropriate interpretation software, it is reasonable to show subtypes of text to the user, while it is not reasonable to do so with most nontextual data.

Such formatted textual data should be represented using subtypes of text. Plausible subtypes of text are typically given by the common name of the representation format, e.g., "text/enriched" [RFC-1563].

6.1.1.1. Representation of Line Breaks

The canonical form of any MIME text type MUST represent a line break as a CRLF sequence. Similarly, any occurrence of CRLF in text MUST represent a line break. Use of CR and LF outside of line break sequences is also forbidden.

This rule applies regardless of format or character set or sets involved.

6.1.1.2. Charset Parameter

A critical parameter that may be specified in the Content-Type field for text/plain data is the character set. This is specified with a "charset" parameter, as in:

```
  Content-type: text/plain; charset=iso-8859-1
```

Unlike some other parameter values, the values of the charset parameter are NOT case sensitive. The default character set, which must be assumed in the absence of a charset parameter, is US-ASCII.

The specification for any future subtypes of "text" must specify whether or not they will also utilize a "charset" parameter, and may possibly restrict its values as well. When used with a particular body, the semantics of the "charset" parameter should be identical to those specified here for "text/plain", i.e., the body consists entirely of characters in the given charset. In particular, definers of future text subtypes should pay close attention the the implications of multioctet character sets for their subtype definitions.

This RFC specifies the definition of the charset parameter for the purposes of MIME to be the name of a character set, as "character set" as defined in Section 4 of this document. The rules regarding line breaks detailed in the previous section must also be observed -- a character set whose definition does not conform to these rules cannot be used in a MIME text type.

An initial list of predefined character set names can be found at the end of this section. Additional character sets may be registered with IANA as described in RFC REG.

Note that if the specified character set includes 8-bit data, a Content-Transfer-Encoding header field and a corresponding encoding on the data are required in order to transmit the body via some mail transfer protocols, such as SMTP.

The default character set, US-ASCII, has been the subject of some confusion and ambiguity in the past. Not only were there some ambiguities in the definition, there have been wide variations in practice. In order to eliminate such ambiguity and variations in the future, it is strongly recommended that new user agents explicitly specify a character set via the Content-Type header field. "US-ASCII" does not indicate an arbitrary 7-bit character code, but specifies that the body uses character coding that uses the exact correspondence of octets to characters specified in US-ASCII. National use variations of ISO 646 [ISO-646] are NOT US-ASCII and their use in Internet mail is explicitly discouraged. The omission of the ISO 646 character set is deliberate in this regard. The character set name of "US-ASCII" explicitly refers to ANSI X3.4-1986 [US-ASCII] only. The character set name "ASCII" is reserved and must not be used for any purpose.

NOTE: RFC 821 explicitly specifies "ASCII", and references an earlier version of the American Standard. Insofar as one of the purposes of specifying a Content-Type and character set is to permit the receiver to unambiguously determine how the sender intended the coded message to be interpreted, assuming anything other than "strict ASCII" as the default would risk unintentional and incompatible changes to the semantics of messages now being transmitted. This also implies that messages containing characters coded according to national variations on ISO 646, or using code-switching procedures (e.g., those of ISO 2022), as well as 8-bit or multiple octet character encodings MUST use an appropriate character set specification to be consistent with this specification.

The complete US-ASCII character set is listed in ANSI X3.4-**1986. Note that the control characters including DEL (0-31,** 127) have no defined meaning apart from the combination CRLF (US-ASCII values 13 and 10) indicating a new line. Two of the characters have de facto meanings in wide use: FF (12) often means "start subsequent text on the beginning of a new page"; and TAB or HT (9) often (though not always) means "move the cursor to the next available column after the current position where the column number is a multiple of 8 (counting the first

column as column 0)." Apart from this, any use of the control characters or DEL in a body must be part of a private agreement between the sender and recipient. Such private agreements are discouraged and should be replaced by the other capabilities of this document.

NOTE: Beyond US-ASCII, an enormous proliferation of character sets is possible. It is the opinion of the IETF working group that a large number of character sets is NOT a good thing. We would prefer to specify a SINGLE character set that can be used universally for representing all of the world's languages in electronic mail. Unfortunately, existing practice in several communities seems to point to the continued use of multiple character sets in the near future. For this reason, we define names for a small number of character sets for which a strong constituent base exists.

The defined charset values are:

(1) US-ASCII -- as defined in ANSI X3.4-1986 [US-ASCII].

(2) ISO-8859-X -- where "X" is to be replaced, as necessary, for the parts of ISO-8859 [ISO-8859]. Note that the ISO 646 character sets have deliberately been omitted in favor of their 8859 replacements, which are the designated character sets for Internet mail. As of the publication of this document, the legitimate values for "X" are the digits 1 through 9.

All of these character sets are used as pure 7- or 8-bit sets without any shift or escape functions. The meaning of shift and escape sequences in these character sets is not defined.

The character sets specified above are the ones that were relatively uncontroversial during the drafting of MIME. This document does not endorse the use of any particular character set other than US-ASCII, and recognizes that the future evolution of world character sets remains unclear. It is expected that in the future, additional character sets will be registered for use in MIME.

Note that the character set used, if anything other than US-ASCII, must always be explicitly specified in the Content-Type field.

No other character set name may be used in Internet mail without the publication of a formal specification and its registration with IANA, or by private agreement, in which case the character set name must begin with "X-".

Implementors are discouraged from defining new character sets for mail use unless absolutely necessary.

The "charset" parameter has been defined primarily for the purpose of textual data, and is described in this section for that reason. However, it is conceivable that non-textual data might also wish to specify a charset value for some purpose, in which case the same syntax and values should be used.

In general, mail-sending software should always use the "lowest common denominator" character set possible. For example, if a body contains only US-ASCII characters, it should be marked as being in the US-ASCII character set, not ISO-8859-1, which, like all the ISO-8859 family of character sets, is a superset of US-ASCII. More generally, if a widely-used character set is a subset of another character set, and a body contains only characters in the widely-used subset, it should be labelled as being in that subset. This will increase the chances that the recipient will be able to view the mail correctly.

6.1.1.3. Plain Subtype

The simplest and most important subtype of text is "plain". This indicates plain (unformatted) text. The default Content-Type for Internet mail, "text/plain; charset=us-ascii", describes existing Internet practice. That is, it is the type of body defined by RFC 822.

No other text subtype is defined by this document.

6.1.1.4. Unrecognized Subtypes

Unrecognized subtypes of text should be treated as subtype "plain" as long as the MIME implementation knows how to handle the charset. Unrecognized subtypes which also specify an unrecognized charset should be treated as "application/octet-stream".

6.1.2. Image Content-Type

A Content-Type of "image" indicates that the body contains an image. The subtype names the specific image format. These names are not case sensitive. Two initial subtypes are "jpeg" for the JPEG format, JFIF encoding, and "gif" for GIF format [GIF].

The list of image subtypes given here is neither exclusive nor exhaustive, and is expected to grow as more types are registered with IANA, as described in RFC REG.

Unrecognized subtypes of image should at a miniumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of image that they do not specifically recognize to a robust general-purpose image viewing application, if such an application is available.

6.1.3. Audio Content-Type

A Content-Type of "audio" indicates that the body contains audio data. Although there is not yet a consensus on an "ideal" audio format for use with computers, there is a pressing need for a format capable of providing interoperable behavior.

The initial subtype of "basic" is specified to meet this requirement by providing an absolutely minimal lowest common denominator audio format. It is expected that richer formats for higher quality and/or lower bandwidth audio will be defined by a later document.

The content of the "audio/basic" subtype is single channel audio encoded using 8-bit ISDN mu-law [PCM] at a sample rate of 8000 Hz.

Unrecognized subtypes of audio should at a miniumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of audio that they do not specifically recognize to a robust general-purpose audio playing application, if such an application is available.

6.1.4. Video Content-Type

A Content-Type of "video" indicates that the body contains a time-varying-picture image, possibly with color and coordinated sound. The term "video" is used extremely generically, rather than with reference to any particular technology or format, and is not meant to preclude subtypes such as animated drawings encoded compactly. The subtype "mpeg" refers to video coded according to the MPEG standard [MPEG].

Note that although in general this document strongly discourages the mixing of multiple media in a single body, it is recognized that many so-called "video" formats include a representation for synchronized audio, and this is explicitly permitted for subtypes of "video".

Unrecognized subtypes of video should at a minumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of video that they do not specifically recognize to a robust general-purpose video display application, if such an application is available.

6.1.5. Application Content-Type

The "application" Content-Type is to be used for discrete data which do not fit in any of the other categories, and particularly for data to be processed by mail-based uses of application programs. This is information which must be processed by an application before it is viewable or usable to a user. Expected uses for Content-Type application include mail-based file transfer, spreadsheets, data for mail-based scheduling systems, and languages for "active" (computational) email. (The latter, in particular, can pose security problems which must be understood by implementors, and are considered in detail in the discussion of the application/PostScript content-type.)

For example, a meeting scheduler might define a standard representation for information about proposed meeting dates. An intelligent user agent would use this information to conduct a dialog with the user, and might then send further mail based on that dialog. More generally, there have been

several "active" messaging languages developed in which programs in a suitably specialized language are sent through the mail and automatically run in the recipient's environment.

Such applications may be defined as subtypes of the "application" Content-Type. This document defines two subtypes: octet-stream, and PostScript.

The subtype of application will often be the name of the application for which the data are intended. This does not mean, however, that any application program name may be used freely as a subtype of application. Usage of any subtype (other than subtypes beginning with "x-") must be registered with IANA, as described in RFC REG.

6.1.5.1. Octet-Stream Subtype

The "octet-stream" subtype is used to indicate that a body contains arbitrary binary data. The set of currently defined parameters is:

(1) TYPE -- the general type or category of binary data. This is intended as information for the human recipient rather than for any automatic processing.

(2) PADDING -- the number of bits of padding that were appended to the bit-stream comprising the actual contents to produce the enclosed 8-bit byte-oriented data. This is useful for enclosing a bit-stream in a body when the total number of bits is not a multiple of 8.

Both of these parameters are optional.

An additional parameter, "CONVERSIONS", was defined in RFC 1341 **but has since been removed.** RFC 1341 also defined the use of a "NAME" parameter which gave a suggested file name to be used if the data were to be written to a file. This has been deprecated in anticipation of a separate Content-Disposition header field, to be defined in a subsequent RFC.

The recommended action for an implementation that receives application/octet-stream mail is to simply offer to put the data in a file, with any Content-Transfer-Encoding undone, or

perhaps to use it as input to a user-specified process.

To reduce the danger of transmitting rogue programs through the mail, it is strongly recommended that implementations NOT implement a path-search mechanism whereby an arbitrary program named in the Content-Type parameter (e.g., an "interpreter=" parameter) is found and executed using the mail body as input.

6.1.5.2. PostScript Subtype

A Content-Type of "application/postscript" indicates a PostScript program. Currently two variants of the PostScript language are allowed; the original level 1 variant is described in [POSTSCRIPT] and the more recent level 2 variant is described in [POSTSCRIPT2].

PostScript is a registered trademark of Adobe Systems, Inc. Use of the MIME content-type "application/postscript" implies recognition of that trademark and all the rights it entails.

The PostScript language definition provides facilities for internal labelling of the specific language features a given program uses. This labelling, called the PostScript document structuring conventions, or DSC, is very general and provides substantially more information than just the language level. The use of document structuring conventions, while not required, is strongly recommended as an aid to interoperability. Documents which lack proper structuring conventions cannot be tested to see whether or not they will work in a given environment. As such, some systems may assume the worst and refuse to process unstructured documents.

The execution of general-purpose PostScript interpreters entails serious security risks, and implementors are discouraged from simply sending PostScript email bodies to "off-the-shelf" interpreters. While it is usually safe to send PostScript to a printer, where the potential for harm is greatly constrained by typical printer environments, implementors should consider all of the following before they add interactive display of PostScript bodies to their mail readers.

The remainder of this section outlines some, though probably not all, of the possible problems with sending PostScript

through the mail.

(1) Dangerous operations in the PostScript language include, but may not be limited to, the PostScript operators "deletefile", "renamefile", "filenameforall", and "file". "File" is only dangerous when applied to something other than standard input or output. Implementations may also define additional nonstandard file operators; these may also pose a threat to security. "Filenameforall", the wildcard file search operator, may appear at first glance to be harmless. Note, however, that this operator has the potential to reveal information about what files the recipient has access to, and this information may itself be sensitive. Message senders should avoid the use of potentially dangerous file operators, since these operators are quite likely to be unavailable in secure PostScript implementations. Message receiving and displaying software should either completely disable all potentially dangerous file operators or take special care not to delegate any special authority to their operation. These operators should be viewed as being done by an outside agency when interpreting PostScript documents. Such disabling and/or checking should be done completely outside of the reach of the PostScript language itself; care should be taken to insure that no method exists for re-enabling full-function versions of these operators.

(2) The PostScript language provides facilities for exiting the normal interpreter, or server, loop. Changes made in this "outer" environment are customarily retained across documents, and may in some cases be retained semipermanently in nonvolatile memory. The operators associated with exiting the interpreter loop have the potential to interfere with subsequent document processing. As such, their unrestrained use constitutes a threat of service denial. PostScript operators that exit the interpreter loop include, but may not be limited to, the exitserver and startjob operators. Message sending software should not generate PostScript that depends on exiting the interpreter loop to operate, since the ability to exit will probably be unavailable in secure PostScript implementations. Message receiving and displaying software should

completely disable the ability to make retained changes to the PostScript environment by eliminating or disabling the "startjob" and "exitserver" operations. If these operations cannot be eliminated or completely disabled the password associated with them should at least be set to a hard-to-guess value.

(3) PostScript provides operators for setting system-wide and device-specific parameters. These parameter settings may be retained across jobs and may potentially pose a threat to the correct operation of the interpreter. The PostScript operators that set system and device parameters include, but may not be limited to, the "setsystemparams" and "setdevparams" operators. Message sending software should not generate PostScript that depends on the setting of system or device parameters to operate correctly. The ability to set these parameters will probably be unavailable in secure PostScript implementations. Message receiving and displaying software should disable the ability to change system and device parameters. If these operators cannot be completely disabled the password associated with them should at least be set to a hard-to-guess value.

(4) Some PostScript implementations provide nonstandard facilities for the direct loading and execution of machine code. Such facilities are quite obviously open to substantial abuse. Message sending software should not make use of such features. Besides being totally hardware-specific, they are also likely to be unavailable in secure implementations of PostScript. Message receiving and displaying software should not allow such operators to be used if they exist.

(5) PostScript is an extensible language, and many, if not most, implementations of it provide a number of their own extensions. This document does not deal with such extensions explicitly since they constitute an unknown factor. Message sending software should not make use of nonstandard extensions; they are likely to be missing from some implementations. Message receiving and displaying software should make sure that any nonstandard PostScript operators are secure and don't present any kind of threat.

(6) It is possible to write PostScript that consumes huge amounts of various system resources. It is also possible to write PostScript programs that loop indefinitely. Both types of programs have the potential to cause damage if sent to unsuspecting recipients. Message-sending software should avoid the construction and dissemination of such programs, which is antisocial. Message receiving and displaying software should provide appropriate mechanisms to abort processing of a document after a reasonable amount of time has elapsed. In addition, PostScript interpreters should be limited to the consumption of only a reasonable amount of any given system resource.

(7) It is possible to include raw binary information inside PostScript in various forms. This is not recommended for use in email, both because it is not supported by all PostScript interpreters and because it significantly complicates the use of a MIME Content-Transfer-Encoding. (Without such binary, PostScript may typically be viewed as line-oriented data. The treatment of CRLF sequences becomes extremely problematic if binary and line-oriented data are mixed in a single Postscript data stream.)

(8) Finally, bugs may exist in some PostScript interpreters which could possibly be exploited to gain unauthorized access to a recipient's system. Apart from noting this possibility, there is no specific action to take to prevent this, apart from the timely correction of such bugs if any are found.

6.1.5.3. Other Application Subtypes

It is expected that many other subtypes of application will be defined in the future. MIME implementations must at a minimum treat any unrecognized subtypes as being equivalent to "application/octet-stream".

6.2. Composite Content-Type Values

The remaining two of the seven initial Content-Type values refer to composite entities. Composite entities are handled using MIME mechanisms -- a MIME processor typically handles the body directly.

6.2.1. Multipart Content-Type

In the case of multiple part entities, in which one or more different sets of data are combined in a single body, a "multipart" Content-Type field must appear in the entity's header. The body must then contain one or more "body parts," each preceded by an encapsulation boundary, and the last one followed by a closing boundary. Each part starts with an encapsulation boundary, and then contains a body part consisting of a header area, a blank line, and a body area. Thus a body part is similar to an RFC 822 message in syntax, but different in meaning.

A body part is NOT to be interpreted as actually being an RFC 822 **message.** To begin with, NO header fields are actually required in body parts. A body part that starts with a blank line, therefore, is allowed and is a body part for which all default values are to be assumed. In such a case, the absence of a Content-Type header indicates that the corresponding body has a content-type of "text/plain; charset=US-ASCII"".

The only header fields that have defined meaning for body parts are those the names of which begin with "Content-". All other header fields are generally to be ignored in body parts. Although they should generally be retained in mail processing, they may be discarded by gateways if necessary. Such other fields are permitted to appear in body parts but must not be depended on. "X-" fields may be created for experimental or private purposes, with the recognition that the information they contain may be lost at some gateways.

NOTE: The distinction between an RFC 822 message and a body part is subtle, but important. A gateway between Internet and X.400 **mail, for example, must be able to tell the difference** between a body part that contains an image and a body part that contains an encapsulated message, the body of which is a

GIF image. In order to represent the latter, the body part must have "Content-Type: message/rfc822", and its body (after the blank line) must be the encapsulated message, with its own "Content-Type: image/gif" header field. The use of similar syntax facilitates the conversion of messages to body parts, and vice versa, but the distinction between the two must be understood by implementors. (For the special case in which all parts actually are messages, a "digest" subtype is also defined.)

As stated previously, each body part is preceded by an encapsulation boundary. The encapsulation boundary MUST NOT appear inside any of the encapsulated parts. Thus, it is crucial that the composing agent be able to choose and specify a unique boundary that will separate the parts.

All present and future subtypes of the "multipart" type must use an identical syntax. Subtypes may differ in their semantics, and may impose additional restrictions on syntax, but must conform to the required syntax for the multipart type. This requirement ensures that all conformant user agents will at least be able to recognize and separate the parts of any multipart entity, even of an unrecognized subtype.

As stated in the definition of the Content-Transfer-Encoding field, no encoding other than "7bit", "8bit", or "binary" is permitted for entities of type "multipart". The multipart delimiters and header fields are always represented as 7-bit US-ASCII in any case (though the header fields may encode non-US-ASCII header text as per RFC MIME-HEADERS, and data within the body parts can be encoded on a part-by-part basis, with Content-Transfer-Encoding fields for each appropriate body part.

Message transport agents, relays, and gateways are commonly known to alter the top-level header of an RFC 822 message. In particular, they frequently add, remove, or reorder header fields. Such alterations are explicitly forbidden for the headers of any body part which occurs within an enclosing multipart body part.

6.2.1.1. Common Syntax

This section defines a common syntax for subtypes of multipart. All subtypes of multipart must use this syntax. A simple example of a multipart message also appears in this section. An example of a more complex multipart message is given in Appendix C.

The Content-Type field for multipart entities requires one parameter, "boundary", which is used to specify the encapsulation boundary. The encapsulation boundary is defined as a line consisting entirely of two hyphen characters ("-", decimal value 45) followed by the boundary parameter value from the Content-Type header field.

NOTE: The hyphens are for rough compatibility with the earlier RFC 934 method of message encapsulation, and for ease of searching for the boundaries in some implementations. However, it should be noted that multipart messages are NOT completely compatible with RFC 934 encapsulations; in particular, they do not obey RFC 934 quoting conventions for embedded lines that begin with hyphens. This mechanism was chosen over the RFC 934 mechanism because the latter causes lines to grow with each level of quoting. The combination of this growth with the fact that SMTP implementations sometimes wrap long lines made the RFC 934 mechanism unsuitable for use in the event that deeply-nested multipart structuring is ever desired.

WARNING TO IMPLEMENTORS: The grammar for parameters on the Content-type field is such that it is often necessary to enclose the boundaries in quotes on the Content-type line. This is not always necessary, but never hurts. Implementors should be sure to study the grammar carefully in order to avoid producing invalid Content-type fields. Thus, a typical multipart Content-Type header field might look like this:

```
  Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
```

But the following is not valid:

```
  Content-Type: multipart/mixed; boundary=gc0pJq0M:08jU534c0p
```

(because of the colon) and must instead be represented as

```
  Content-Type: multipart/mixed; boundary="gc0pJq0M:08jU534c0p"
```

This Content-Type value indicates that the content consists of one or more parts, each with a structure that is syntactically identical to an RFC 822 message, except that the header area is allowed to be completely empty, and that the parts are each preceded by the line

```
  --gc0pJq0M:08jU534c0p
```

The encapsulation boundary MUST occur at the beginning of a line, i.e., following a CRLF, and the initial CRLF is considered to be attached to the encapsulation boundary rather than part of the preceding part. The boundary must be followed immediately either by another CRLF and the header fields for the next part, or by two CRLFs, in which case there are no header fields for the next part (and it is therefore assumed to be of Content-Type text/plain).

NOTE: The CRLF preceding the encapsulation line is conceptually attached to the boundary so that it is possible to have a part that does not end with a CRLF (line break). Body parts that must be considered to end with line breaks, therefore, must have two CRLFs preceding the encapsulation line, the first of which is part of the preceding body part, and the second of which is part of the encapsulation boundary.

Encapsulation boundaries must not appear within the encapsulations, and must be no longer than 70 characters, not counting the two leading hyphens.

The encapsulation boundary following the last body part is a distinguished delimiter that indicates that no further body parts will follow. Such a delimiter is identical to the previous delimiters, with the addition of two more hyphens at the end of the line:

```
  --gc0pJq0M:08jU534c0p--
```

There appears to be room for additional information prior to the first encapsulation boundary and following the final boundary. These areas should generally be left blank, and implementations must ignore anything that appears before the first boundary or after the last one.

NOTE: These "preamble" and "epilogue" areas are generally not used because of the lack of proper typing of these parts and the lack of clear semantics for handling these areas at gateways, particularly X.400 gateways. However, rather than leaving the preamble area blank, many MIME implementations have found this to be a convenient place to insert an explanatory note for recipients who read the message with pre-MIME software, since such notes will be ignored by MIME-compliant software.

NOTE: Because encapsulation boundaries must not appear in the body parts being encapsulated, a user agent must exercise care to choose a unique boundary. The boundary in the example above could have been the result of an algorithm designed to produce boundaries with a very low probability of already existing in the data to be encapsulated without having to prescan the data. Alternate algorithms might result in more "readable" boundaries for a recipient with an old user agent, but would require more attention to the possibility that the boundary might appear in the encapsulated part. The simplest boundary possible is something like "---", with a closing boundary of "-----".

As a very simple example, the following multipart message has two parts, both of them plain text, one of them explicitly typed and one of them implicitly typed:

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Date: Sun, 21 Mar 1993 23:56:48 -0800 (PST)
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"

This is the preamble.  It is to be ignored, though it
is a handy place for mail composers to include an
explanatory note to non-MIME conformant readers.

--simple boundary

This is implicitly typed plain US-ASCII text.
It does NOT end with a linebreak.
--simple boundary
Content-type: text/plain; charset=us-ascii
```

This is explicitly typed plain US-ASCII text.
It DOES end with a linebreak.

--simple boundary--

This is the epilogue. It is also to be ignored.

The use of a Content-Type of multipart in a body part within another multipart entity is explicitly allowed. In such cases, for obvious reasons, care must be taken to ensure that each nested multipart entity uses a different boundary delimiter. See Appendix C for an example of nested multipart entities.

The use of the multipart Content-Type with only a single body part may be useful in certain contexts, and is explicitly permitted.

The only mandatory global parameter for the multipart Content-Type is the boundary parameter, which consists of 1 to **70 characters from a set of characters known to be very robust** through email gateways, and NOT ending with white space. (If a boundary appears to end with white space, the white space must be presumed to have been added by a gateway, and must be deleted.) It is formally specified by the following BNF:

```
boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

bcharsnospace := DIGIT / ALPHA / "'" / "(" / ")" /
                 "+" / "_" / "," / "-" / "." /
                 "/" / ":" / "=" / "?"
```

Overall, the body of a multipart entity may be specified as follows:

```
dash-boundary := "--" boundary
                 ; boundary taken from Content-Type
                 ; field.
```

```
multipart-body := preamble dash-boundary
                  [*LWSP-char] CRLF
                  body-part *encapsulation
                  close-delimiter [*LWSP-char]
                  CRLF epilogue

encapsulation := delimiter [*LWSP-char]
                 CRLF body-part

delimiter := CRLF dash-boundary

close-delimiter := CRLF dash-boundary "--"

preamble := discard-text

epilogue := discard-text

discard-text := *text *(*text CRLF)
                ; To be ignored upon receipt.

body-part := <"message" as defined in RFC 822, with all
              header fields optional, not starting with the
              specified dash-boundary, and with the
              delimiter not occurring anywhere in the
              message body.  Note that the semantics of a
              part differ from the semantics of a message,
              as described in the text.>
```

IMPORTANT NOTE: The addition of LWSP between the elements shown in this BNF is NOT allowed since this BNF does not specify a structured header field.

NOTE: In certain transport enclaves, RFC 822 restrictions such as the one that limits bodies to printable US-ASCII characters may not be in force. (That is, the transport domains may resemble standard Internet mail transport as specified in RFC 821 and assumed by RFC 822, but without certain restrictions.) The relaxation of these restrictions should be construed as locally extending the definition of bodies, for example to include octets outside of the US-ASCII range, as long as these extensions are supported by the transport and adequately documented in the Content-Transfer-Encoding header field. However, in no event are headers (either message headers or body-part headers) allowed to contain anything other than US-ASCII characters.

NOTE: Conspicuously missing from the multipart type is a notion of structured, related body parts. In general, it seems premature to try to standardize interpart structure yet. It is recommended that those wishing to provide a more structured or integrated multipart messaging facility should define a subtype of multipart that is syntactically identical, but that always expects the inclusion of a distinguished part that can be used to specify the structure and integration of the other parts, probably referring to them by their Content-ID field. If this approach is used, other implementations will not recognize the new subtype, but will treat it as the primary subtype (multipart/mixed) and will thus be able to show the user the parts that are recognized.

6.2.1.2. Handling Nested Messages and Multiparts

The "message/rfc822" subtype defined in a subsequent section of this document has no terminating condition other than running out of data. Similarly, an improperly truncated multipart object may not have any terminating boundary marker, and does arise in practice due to mail system malfunctions.

It is essential that such objects be handled correctly when they are themselves imbedded inside of another multipart structure. MIME implementations are therefore required to recognize outer level boundary markers at ANY level of inner nesting. It is not sufficient to only check for the next expected marker or other terminating condition.

6.2.1.3. Mixed Subtype

The "mixed" subtype of multipart is intended for use when the body parts are independent and need to be bundled in a particular order. Any multipart subtypes that an implementation does not recognize must be treated as being of subtype "mixed".

6.2.1.4. Alternative Subtype

The multipart/alternative type is syntactically identical to multipart/mixed, but the semantics are different. In particular, each of the parts is an "alternative" version of

the same information.

Systems should recognize that the content of the various parts are interchangeable. Systems should choose the "best" type based on the local environment and preferences, in some cases even through user interaction. As with multipart/mixed, the order of body parts is significant. In this case, the alternatives appear in an order of increasing faithfulness to the original content. In general, the best choice is the LAST part of a type supported by the recipient system's local environment.

Multipart/alternative may be used, for example, to send mail in a fancy text format in such a way that it can easily be displayed anywhere:

```
  From: Nathaniel Borenstein <nsb@bellcore.com>
  To: Ned Freed <ned@innosoft.com>
  Date: Mon, 22 Mar 1993 09:41:09 -0800 (PST)
  Subject: Formatted text mail
  MIME-Version: 1.0
  Content-Type: multipart/alternative; boundary=boundary42

  --boundary42
  Content-Type: text/plain; charset=us-ascii

    ... plain text version of message goes here ...

  --boundary42
  Content-Type: text/enriched

    ... RFC 1563 text/enriched version of same message
        goes here ...

  --boundary42
  Content-Type: application/x-whatever

    ... fanciest version of same message goes here ...

  --boundary42--
```

In this example, users whose mail system understood the "application/x-whatever" format would see only the fancy version, while other users would see only the enriched or plain text version, depending on the capabilities of their

system.

In general, user agents that compose multipart/alternative entities must place the body parts in increasing order of preference, that is, with the preferred format last. For fancy text, the sending user agent should put the plainest format first and the richest format last. Receiving user agents should pick and display the last format they are capable of displaying. In the case where one of the alternatives is itself of type "multipart" and contains unrecognized sub-parts, the user agent may choose either to show that alternative, an earlier alternative, or both.

NOTE: From an implementor's perspective, it might seem more sensible to reverse this ordering, and have the plainest alternative last. However, placing the plainest alternative first is the friendliest possible option when multipart/alternative entities are viewed using a non-MIME-conformant mail reader. While this approach does impose some burden on conformant mail readers, interoperability with older mail readers was deemed to be more important in this case.

It may be the case that some user agents, if they can recognize more than one of the formats, will prefer to offer the user the choice of which format to view. This makes sense, for example, if mail includes both a nicely-formatted image version and an easily-edited text version. What is most critical, however, is that the user not automatically be shown multiple versions of the same data. Either the user should be shown the last recognized version or should be given the choice.

NOTE ON THE SEMANTICS OF CONTENT-ID IN MULTIPART/ALTERNATIVE: Each part of a multipart/alternative entity represents the same data, but the mappings between the two are not necessarily without information loss. For example, information is lost when translating ODA to PostScript or plain text. It is recommended that each part should have a different Content-ID value in the case where the information content of the two parts is not identical. And when the information content is identical -- for example, where several parts of type "message/external-body" specify alternate ways to access the identical data -- the same Content-ID field value should be used, to optimize any caching mechanisms that might be present on the recipient's end. However, the

Content-ID values used by the parts should NOT be the same Content-ID value that describes the multipart/alternative as a whole, if there is any such Content-ID field. That is, one Content-ID value will refer to the multipart/alternative entity, while one or more other Content-ID values will refer to the parts inside it.

6.2.1.5. Digest Subtype

This document defines a "digest" subtype of the multipart Content-Type. This type is syntactically identical to multipart/mixed, but the semantics are different. In particular, in a digest, the default Content-Type value for a body part is changed from "text/plain" to "message/rfc822". This is done to allow a more readable digest format that is largely compatible (except for the quoting convention) with RFC 934.

A digest in this format might, then, look something like this:

```
  From: Moderator-Address
  To: Recipient-List
  Date: Mon, 22 Mar 1994 13:34:51 +0000
  Subject: Internet Digest, volume 42
  MIME-Version: 1.0
  Content-Type: multipart/digest;
                boundary="---- next message ----"

  ------ next message ----

  From: someone-else
  Date: Fri, 26 Mar 1993 11:13:32 +0200
  Subject: my opinion

    ...body goes here ...

  ------ next message ----

  From: someone-else-again
  Date: Fri, 26 Mar 1993 10:07:13 -0500
  Subject: my different opinion

    ... another body goes here ...
```

------ next message ------

6.2.1.6. Parallel Subtype

This document defines a "parallel" subtype of the multipart Content-Type. This type is syntactically identical to multipart/mixed, but the semantics are different. In particular, in a parallel entity, the order of body parts is not significant.

A common presentation of this type is to display all of the parts simultaneously on hardware and software that are capable of doing so. However, composing agents should be aware that many mail readers will lack this capability and will show the parts serially in any event.

6.2.1.7. Other Multipart Subtypes

Other multipart subtypes are expected in the future. MIME implementations must in general treat unrecognized subtypes of multipart as being equivalent to "multipart/mixed".

6.2.2. Message Content-Type

It is frequently desirable, in sending mail, to encapsulate another mail message. A special Content-Type, "message", is defined to facilitate this. In particular, the "rfc822" subtype of "message" is used to encapsulate RFC 822 messages.

NOTE: It has been suggested that subtypes of message might be defined for forwarded or rejected messages. However, forwarded and rejected messages can be handled as multipart messages in which the first part contains any control or descriptive information, and a second part, of type message/rfc822, is the forwarded or rejected message. Composing rejection and forwarding messages in this manner will preserve the type information on the original message and allow it to be correctly presented to the recipient, and hence is strongly encouraged.

Subtypes of message often impose restrictions on what encodings are allowed. These restrictions are described in conjunction with each specific subtype.

Mail gateways, relays, and other mail handling agents are commonly known to alter the top-level header of an RFC 822 message. In particular, they frequently add, remove, or reorder header fields. Such alterations are explicitly forbidden for the encapsulated headers embedded in the bodies of messages of type "message."

6.2.2.1. RFC822 Subtype

A Content-Type of "message/rfc822" indicates that the body contains an encapsulated message, with the syntax of an RFC **822 message.** However, unlike top-level RFC 822 messages, the restriction that each message/rfc822 body must include a "From", "Date", and at least one destination header is removed and replaced with the requirement that at least one of "From", "Subject", or "Date" must be present.

No encoding other than "7bit", "8bit", or "binary" is permitted for parts of type "message/rfc822". The message header fields are always US-ASCII in any case, and data within the body can still be encoded, in which case the Content-Transfer-Encoding header field in the encapsulated message will reflect this. Non-US-ASCII text in the headers of an encapsulated message can be specified using the mechanisms described in RFC MIME-HEADERS.

It should be noted that, despite the use of the numbers "822", a message/rfc822 entity can include enhanced information as defined in this document. In other words, a message/rfc822 message may be a MIME message.

6.2.2.2. Partial Subtype

The "partial" subtype is defined to allow large entities to be delivered as several separate pieces of mail and automatically reassembled by the receiving user agent. (The concept is similar to IP fragmentation and reassembly in the basic Internet Protocols.) This mechanism can be used when intermediate transport agents limit the size of individual

messages that can be sent. Content-Type "message/partial" thus indicates that the body contains a fragment of a larger message.

Three parameters must be specified in the Content-Type field of type message/partial: The first, "id", is a unique identifier, as close to a world-unique identifier as possible, to be used to match the parts together. (In general, the identifier is essentially a message-id; if placed in double quotes, it can be ANY message-id, in accordance with the BNF for "parameter" given earlier in this specification.) The second, "number", an integer, is the part number, which indicates where this part fits into the sequence of fragments. The third, "total", another integer, is the total number of parts. This third subfield is required on the final part, and is optional (though encouraged) on the earlier parts. Note also that these parameters may be given in any order.

Thus, part 2 of a 3-part message may have either of the following header fields:

```
  Content-Type: Message/Partial; number=2; total=3;
                id="oc=jpbe0M2Yt4s@thumper.bellcore.com"

  Content-Type: Message/Partial;
                id="oc=jpbe0M2Yt4s@thumper.bellcore.com";
                number=2
```

But part 3 MUST specify the total number of parts:

```
  Content-Type: Message/Partial; number=3; total=3;
                id="oc=jpbe0M2Yt4s@thumper.bellcore.com"
```

Note that part numbering begins with 1, not 0.

When the parts of a message broken up in this manner are put together, the result is a complete MIME entity, which may have its own Content-Type header field, and thus may contain any other data type.

6.2.2.2.1. Message Fragmentation and Reassembly

The semantics of a reassembled partial message must be those of the "inner" message, rather than of a message containing

the inner message. This makes it possible, for example, to send a large audio message as several partial messages, and still have it appear to the recipient as a simple audio message rather than as an encapsulated message containing an audio message. That is, the encapsulation of the message is considered to be "transparent".

When generating and reassembling the parts of a message/partial message, the headers of the encapsulated message must be merged with the headers of the enclosing entities. In this process the following rules must be observed:

(1) All of the header fields from the initial enclosing entity (part one), except those that start with "Content-" and the specific header fields "Subject", "Message-ID", "Encrypted", and "MIME-Version", must be copied, in order, to the new message.

(2) Only those header fields in the enclosed message which start with "Content-" and "Subject", "Message-ID", "Encrypted", and "MIME-Version" must be appended, in order, to the header fields of the new message. Any header fields in the enclosed message which do not start with "Content-" (except for "Message-ID", "Encrypted", and "MIME-Version") will be ignored.

(3) All of the header fields from the second and any subsequent messages will be ignored.

6.2.2.2.2. Fragmentation and Reassembly Example

If an audio message is broken into two parts, the first part might look something like this:

```
X-Weird-Header-1: Foo
From: Bill@host.com
To: joe@otherhost.com
Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
Subject: Audio mail (part 1 of 2)
Message-ID: <id1@host.com>
MIME-Version: 1.0
Content-type: message/partial; id="ABC@host.com";
              number=1; total=2
```

```
  X-Weird-Header-1: Bar
  X-Weird-Header-2: Hello
  Message-ID: <anotherid@foo.com>
  Subject: Audio mail
  MIME-Version: 1.0
  Content-type: audio/basic
  Content-transfer-encoding: base64

    ... first half of encoded audio data goes here ...
```

and the second half might look something like this:

```
  From: Bill@host.com
  To: joe@otherhost.com
  Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
  Subject: Audio mail (part 2 of 2)
  MIME-Version: 1.0
  Message-ID: <id2@host.com>
  Content-type: message/partial;
                id="ABC@host.com"; number=2; total=2

    ... second half of encoded audio data goes here ...
```

Then, when the fragmented message is reassembled, the resulting message to be displayed to the user should look something like this:

```
  X-Weird-Header-1: Foo
  From: Bill@host.com
  To: joe@otherhost.com
  Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
  Subject: Audio mail
  Message-ID: <anotherid@foo.com>
  MIME-Version: 1.0
  Content-type: audio/basic
  Content-transfer-encoding: base64

    ... first half of encoded audio data goes here ...
    ... second half of encoded audio data goes here ...
```

Because data of type "message" may never be encoded in base64 or quoted-printable, a problem might arise if message/partial entities are constructed in an environment that supports binary or 8-bit transport. The problem is that the binary data would be split into multiple message/partial messages,

each of them requiring binary transport. If such messages were encountered at a gateway into a 7-bit transport environment, there would be no way to properly encode them for the 7-bit world, aside from waiting for all of the fragments, reassembling the inner message, and then encoding the reassembled data in base64 or quoted-printable. Since it is possible that different fragments might go through different gateways, even this is not an acceptable solution. For this reason, it is specified that MIME entities of type message/partial must always have a content-transfer-encoding of 7-bit (the default). In particular, even in environments that support binary or 8-bit transport, the use of a content-transfer-encoding of "8bit" or "binary" is explicitly prohibited for entities of type message/partial.

Because some message transfer agents may choose to automatically fragment large messages, and because such agents may use very different fragmentation thresholds, it is possible that the pieces of a partial message, upon reassembly, may prove themselves to comprise a partial message. This is explicitly permitted.

The inclusion of a "References" field in the headers of the second and subsequent pieces of a fragmented message that references the Message-Id on the previous piece may be of benefit to mail readers that understand and track references. However, the generation of such "References" fields is entirely optional.

Finally, it should be noted that the "Encrypted" header field has been made obsolete by Privacy Enhanced Messaging (PEM) [RFC1421, RFC1422, RFC1423, and RFC1424], but the rules above are nevertheless believed to describe the correct way to treat it if it is encountered in the context of conversion to and from message/partial fragments.

6.2.2.3. External-Body Subtype

The external-body subtype indicates that the actual body data are not included, but merely referenced. In this case, the parameters describe a mechanism for accessing the external data.

When an entity is of type "message/external-body", it consists of a header, two consecutive CRLFs, and the message header for the encapsulated message. If another pair of consecutive CRLFs appears, this of course ends the message header for the encapsulated message. However, since the encapsulated message's body is itself external, it does NOT appear in the area that follows. For example, consider the following message:

```
  Content-type: message/external-body;
                access-type=local-file;
                name="/u/nsb/Me.gif"
  Content-type: image/gif
  Content-ID: <id42@guppylake.bellcore.com>
  Content-Transfer-Encoding: binary

  THIS IS NOT REALLY THE BODY!
```

The area at the end, which might be called the "phantom body", is ignored for most external-body messages. However, it may be used to contain auxiliary information for some such messages, as indeed it is when the access-type is "mail-server". The only access-type defined in this document that uses the phantom body is "mail-server", but other access-types may be defined in the future in other documents that use this area.

The encapsulated headers in ALL message/external-body entities MUST include a Content-ID header field to give a unique identifier by which to reference the data. This identifier may be used for caching mechanisms, and for recognizing the receipt of the data when the access-type is "mail-server".

Note that, as specified here, the tokens that describe external-body data, such as file names and mail server commands, are required to be in the US-ASCII character set. If this proves problematic in practice, a new mechanism may be required as a future extension to MIME, either as newly defined access-types for message/external-body or by some other mechanism.

As with message/partial, MIME entities of type message/external-body MUST have a content-transfer-encoding of 7-bit (the default). In particular, even in environments that support binary or 8-bit transport, the use of a content-

transfer-encoding of "8bit" or "binary" is explicitly prohibited for entities of type message/external-body.

6.2.2.3.1. General External-Body Parameters

The parameters that may be used with any message/external-body are:

(1) ACCESS-TYPE -- A word indicating the supported access mechanism by which the file or data may be obtained. This word is not case sensitive. Values include, but are not limited to, "FTP", "ANON-FTP", "TFTP", "LOCAL-FILE", and "MAIL-SERVER". Future values, except for experimental values beginning with "X-", must be registered with IANA, as described in RFC REG. This parameter is unconditionally mandatory and MUST be present on EVERY message/external-body.

(2) EXPIRATION -- The date (in the RFC 822 "date-time" syntax, as extended by RFC 1123 to permit 4 digits in the year field) after which the existence of the external data is not guaranteed. This parameter may be used with ANY access-type and is ALWAYS optional.

(3) SIZE -- The size (in octets) of the data. The intent of this parameter is to help the recipient decide whether or not to expend the necessary resources to retrieve the external data. Note that this describes the size of the data in its canonical form, that is, before any Content-Transfer-Encoding has been applied or after the data have been decoded. This parameter may be used with ANY access-type and is ALWAYS optional.

(4) PERMISSION -- A case-insensitive field that indicates whether or not it is expected that clients might also attempt to overwrite the data. By default, or if permission is "read", the assumption is that they are not, and that if the data is retrieved once, it is never needed again. If PERMISSION is "read-write", this assumption is invalid, and any local copy must be considered no more than a cache. "Read" and "Read-write" are the only defined values of permission. This parameter may be used with ANY access-type and is

ALWAYS optional.

The precise semantics of the access-types defined here are described in the sections that follow.

6.2.2.3.2. The 'ftp' and 'tftp' Access-Types

An access-type of FTP or TFTP indicates that the message body is accessible as a file using the FTP [RFC-959] or TFTP [RFC-783] protocols, respectively. For these access-types, the following additional parameters are mandatory:

(1) NAME -- The name of the file that contains the actual body data.

(2) SITE -- A machine from which the file may be obtained, using the given protocol. This must be a fully qualified domain name, not a nickname.

(3) Before any data are retrieved, using FTP, the user will generally need to be asked to provide a login id and a password for the machine named by the site parameter. For security reasons, such an id and password are not specified as content-type parameters, but must be obtained from the user.

In addition, the following parameters are optional:

(1) DIRECTORY -- A directory from which the data named by NAME should be retrieved.

(2) MODE -- A case-insensitive string indicating the mode to be used when retrieving the information. The valid values for access-type "TFTP" are "NETASCII", "OCTET", and "MAIL", as specified by the TFTP protocol [RFC-783]. The valid values for access-type "FTP" are "ASCII", "EBCDIC", "IMAGE", and "LOCALn" where "n" is a decimal integer, typically 8. These correspond to the representation types "A" "E" "I" and "L n" as specified by the FTP protocol [RFC-959]. Note that "BINARY" and "TENEX" are not valid values for MODE and that "OCTET" or "IMAGE" or "LOCAL8" should be used instead. IF MODE is not specified, the default value is "NETASCII" for TFTP and "ASCII" otherwise.

6.2.2.3.3. The 'anon-ftp' Access-Type

The "anon-ftp" access-type is identical to the "ftp" access type, except that the user need not be asked to provide a name and password for the specified site. Instead, the ftp protocol will be used with login "anonymous" and a password that corresponds to the user's email address.

6.2.2.3.4. The 'local-file' Access-Type

An access-type of "local-file" indicates that the actual body is accessible as a file on the local machine. Two additional parameters are defined for this access type:

(1) NAME -- The name of the file that contains the actual body data. This parameter is mandatory for the "local-file" access-type.

(2) SITE -- A domain specifier for a machine or set of machines that are known to have access to the data file. This optional parameter is used to describe the locality of reference for the data, that is, the site or sites at which the file is expected to be visible. Asterisks may be used for wildcard matching to a part of a domain name, such as "*.bellcore.com", to indicate a set of machines on which the data should be directly visible, while a single asterisk may be used to indicate a file that is expected to be universally available, e.g., via a global file system.

6.2.2.3.5. The 'mail-server' Access-Type

The "mail-server" access-type indicates that the actual body is available from a mail server. Two additional parameters are defined for this access-type:

(1) SERVER -- The email address of the mail server from which the actual body data can be obtained. This parameter is mandatory for the "mail-server" access-type.

(2) SUBJECT -- The subject that is to be used in the mail that is sent to obtain the data. Note that keying mail

servers on Subject lines is NOT recommended, but such mail servers are known to exist. This is an optional parameter.

Because mail servers accept a variety of syntaxes, some of which is multiline, the full command to be sent to a mail server is not included as a parameter on the content-type line. Instead, it is provided as the "phantom body" when the content-type is message/external-body and the access-type is mail-server.

Note that MIME does not define a mail server syntax. Rather, it allows the inclusion of arbitrary mail server commands in the phantom body. Implementations must include the phantom body in the body of the message it sends to the mail server address to retrieve the relevant data.

Unlike other access-types, mail-server access is asynchronous and will happen at an unpredictable time in the future. For this reason, it is important that there be a mechanism by which the returned data can be matched up with the original message/external-body entity. MIME mailservers must use the same Content-ID field on the returned message that was used in the original message/external-body entity, to facilitate such matching.

6.2.2.3.6. Examples and Further Explanations

When the external-body mechanism is used in conjunction with the multipart/alternative Content-Type it extends the functionality of multipart/alternative to include the case where the same object is provided in the same format but via different accces mechanisms. When this is done the originator of the message must order the part first in terms of preferred formats and then by preferred access mechanisms. The recipient's viewer should then evaluate the list both in terms of format and access mechanisms.

With the emerging possibility of very wide-area file systems, it becomes very hard to know in advance the set of machines where a file will and will not be accessible directly from the file system. Therefore it may make sense to provide both a file name, to be tried directly, and the name of one or more sites from which the file is known to be accessible. An

implementation can try to retrieve remote files using FTP or any other protocol, using anonymous file retrieval or prompting the user for the necessary name and password. If an external body is accessible via multiple mechanisms, the sender may include multiple parts of type message/external-body within an entity of type multipart/alternative.

However, the external-body mechanism is not intended to be limited to file retrieval, as shown by the mail-server access-type. Beyond this, one can imagine, for example, using a video server for external references to video clips.

The embedded message header fields which appear in the body of the message/external-body data must be used to declare the Content-type of the external body if it is anything other than plain US-ASCII text, since the external body does not have a header section to declare its type. Similarly, any Content-transfer-encoding other than "7bit" must also be declared here. Thus a complete message/external-body message, referring to a document in PostScript format, might look like this:

```
From: Whomever
To: Someone
Date: Whenever
Subject: whatever
MIME-Version: 1.0
Message-ID: <id1@host.com>
Content-Type: multipart/alternative; boundary=42
Content-ID: <id001@guppylake.bellcore.com>

--42
Content-Type: message/external-body; name="BodyFormats.ps";
              site="thumper.bellcore.com"; mode="image";
              access-type=ANON-FTP; directory="pub";
              expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"

Content-type: application/postscript
Content-ID: <id42@guppylake.bellcore.com>

--42
Content-Type: message/external-body; access-type=local-file;
              name="/u/nsb/writing/rfcs/RFC-MIME.ps";
              site="thumper.bellcore.com";
              expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"
```

```
  Content-type: application/postscript
  Content-ID: <id42@guppylake.bellcore.com>

  --42
  Content-Type: message/external-body;
                access-type=mail-server
                server="listserv@bogus.bitnet";
                expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"

  Content-type: application/postscript
  Content-ID: <id42@guppylake.bellcore.com>

  get RFC-MIME.DOC

  --42--
```

Note that in the above examples, the default Content-transfer-encoding of "7bit" is assumed for the external postscript data.

Like the message/partial type, the message/external-body type is intended to be transparent, that is, to convey the data type in the external body rather than to convey a message with a body of that type. Thus the headers on the outer and inner parts must be merged using the same rules as for message/partial. In particular, this means that the Content-type header is overridden, but the From and Subject headers are preserved.

Note that since the external bodies are not transported as mail, they need not conform to the 7-bit and line length requirements, but might in fact be binary files. Thus a Content-Transfer-Encoding is not generally necessary, though it is permitted.

Note that the body of a message of type "message/external-body" is governed by the basic syntax for an RFC 822 message. In particular, anything before the first consecutive pair of CRLFs is header information, while anything after it is body information, which is ignored for most access-types.

6.2.2.4. Other Message Subtypes

MIME implementations must in general treat unrecognized subtypes of message as being equivalent to "application/octet-stream".

7. Experimental Content-Type Values

A Content-Type value beginning with the characters "X-" is a private value, to be used by consenting mail systems by mutual agreement. Any format without a rigorous and public definition must be named with an "X-" prefix, and publicly specified values shall never begin with "X-". (Older versions of the widely used Andrew system use the "X-BE2" name, so new systems should probably choose a different name.)

In general, the use of "X-" top-level types is strongly discouraged. Implementors should invent subtypes of the existing types whenever possible. The invention of new types is intended to be restricted primarily to the development of new media types for email, such as digital odors or holography, and not for new data formats in general. In many cases, a subtype of application will be more appropriate than a new top-level type.

8. Summary

Using the MIME-Version, Content-Type, and Content-Transfer-Encoding header fields, it is possible to include, in a standardized way, arbitrary types of data objects with RFC 822 conformant mail messages. No restrictions imposed by either RFC 821 or RFC 822 are violated, and care has been taken to avoid problems caused by additional restrictions imposed by the characteristics of some Internet mail transport mechanisms (see Appendix B). The "multipart" and "message" Content-Types allow mixing and hierarchical structuring of objects of different types in a single message. Further Content-Types provide a standardized mechanism for tagging messages or body parts as audio, image, or several other kinds of data. A distinguished parameter syntax allows further specification of data format details, particularly the specification of alternate character sets. Additional optional header fields provide mechanisms for certain extensions deemed desirable by many implementors. Finally, a number of useful Content-Types are defined for general use by consenting user agents, notably message/partial, and message/external-body.

9. Security Considerations

Security issues are discussed in the context of the application/postscript type and in Appendix E. Implementors should pay special attention to the security implications of any mail content-types that can cause the remote execution of any actions in the recipient's environment. In such cases, the discussion of the application/postscript type may serve as a model for considering other content-types with remote execution capabilities.

10. Authors' Addresses

For more information, the authors of this document may be contacted via Internet mail:

Nathaniel S. Borenstein
First Virtual Holdings
25 Washington Avenue
Morristown, NJ 07960
USA

Email: nsb@nsb.fv.com
Phone: +1 201 540 8967
Fax: +1 201 993 3032

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Email: ned@innosoft.com
Phone: +1 818 919 3600
Fax: +1 818919 3614

MIME is a result of the work of the Internet Engineering Task Force Working Group on Email Extensions. The chairman of that group, Greg Vaudreuil, may be reached at:

Gregory M. Vaudreuil
Tigon Corporation
17060 Dallas Parkway
Dallas Texas, 75248

Email: greg.vaudreuil@ons.octel.com
Phone: +1 214 733 2722

11. Acknowledgements

This document is the result of the collective effort of a large number of people, at several IETF meetings, on the IETF-SMTP and IETF-822 mailing lists, and elsewhere. Although any enumeration seems doomed to suffer from egregious omissions, the following are among the many contributors to this effort:

Harald Tveit Alvestrand	Marc Andreessen
Randall Atkinson	Bob Braden
Philippe Brandon	Brian Capouch
Kevin Carosso	Uhhyung Choi
Peter Clitherow	Dave Collier-Brown
Cristian Constantinof	John Coonrod
Mark Crispin	Dave Crocker
Stephen Crocker	Terry Crowley
Walt Daniels	Jim Davis
Frank Dawson	Axel Deininger
Hitoshi Doi	Kevin Donnelly
Steve Dorner	Keith Edwards
Chris Eich	Dana S. Emery
Johnny Eriksson	Craig Everhart
Patrik Faltstrom	Erik E. Fair
Roger Fajman	Alain Fontaine
Martin Forssen	James M. Galvin
Stephen Gildea	Philip Gladstone
Thomas Gordon	Keld Simonsen
Terry Gray	Phill Gross
James Hamilton	David Herron
Mark Horton	Bruce Howard
Bill Janssen	Olle Jarnefors
Risto Kankkunen	Phil Karn
Alan Katz	Tim Kehres
Neil Katin	Steve Kille
Kyuho Kim	Anders Klemets
John Klensin	Valdis Kletniek
Jim Knowles	Stev Knowles
Bob Kummerfeld	Pekka Kytolaakso
Stellan Lagerstrom	Vincent Lau
Timo Lehtinen	Donald Lindsay
Warner Losh	Carlyn Lowery
Laurence Lundblade	Charles Lynn
John R. MacMillan	Larry Masinter
Rick McGowan	Michael J. McInerny

Leo Mclaughlin	Goli Montaser-Kohsari
Keith Moore	Tom Moore
Erik Naggum	Mark Needleman
John Noerenberg	Mats Ohrman
Julian Onions	Michael Patton
David J. Pepper	Erik van der Poel
Jon Postel	Blake C. Ramsdell
Christer Romson	Luc Rooijakkers
Marshall T. Rose	Jonathan Rosenberg
Guido van Rossum	Jan Rynning
Harri Salminen	Michael Sanderson
Yutaka Sato	Markku Savela
Richard Alan Schafer	Masahiro Sekiguchi
Mark Sherman	Bob Smart
Peter Speck	Henry Spencer
Einar Stefferud	Michael Stein
Klaus Steinberger	Peter Svanberg
James Thompson	Steve Uhler
Stuart Vance	Peter Vanderbilt
Greg Vaudreuil	Ed Vielmetti
Larry W. Virden	Ryan Waldron
Rhys Weatherly	Jay Weber
Dave Wecker	Wally Wedel
Sven-Ove Westberg	Brian Wideen
John Wobus	Glenn Wright
Rayan Zachariassen	David Zimmerman

The authors apologize for any omissions from this list, which are certainly unintentional.

Appendix A -- MIME Conformance

The mechanisms described in this document are open-ended. It is definitely not expected that all implementations will support all of the Content-Types described, nor that they will all share the same extensions. In order to promote interoperability, however, it is useful to define the concept of "MIME-conformance" to define a certain level of implementation that allows the useful interworking of messages with content that differs from US-ASCII text. In this section, we specify the requirements for such conformance.

A mail user agent that is MIME-conformant MUST:

(1) Always generate a "MIME-Version: 1.0" header field.

(2) Recognize the Content-Transfer-Encoding header field and decode all received data encoded with either the quoted-printable or base64 implementations. Any non-7-bit data that is sent without encoding must be properly labelled with a content-transfer-encoding of 8bit or binary, as appropriate. If the underlying transport does not support 8bit or binary (as SMTP [RFC821] does not), the snder is required to both encode and label data using an appropriate Content-Transfer-Encoding such as quoted-printable or base64.

(3) Recognize and interpret the Content-Type header field, and avoid showing users raw data with a Content-Type field other than text. Be able to send at least text/plain messages, with the character set specified as a parameter if it is not US-ASCII.

(4) Explicitly handle the following Content-Type values, to at least the following extents:

Text:

-- Recognize and display "text" mail with the character set "US-ASCII."

-- Recognize other character sets at least to the extent of being able to inform the user about what character set the message uses.

-- Recognize the "ISO-8859-*" character sets to the extent of being able to display those characters that are common to ISO-8859-* and US-ASCII, namely all characters represented by octet values 0-127.

-- For unrecognized subtypes in a known character set, show or offer to show the user the "raw" version of the data after conversion of the content from canonical form to local form.

-- Treat material in an unknown character set as if it were "application/octet-stream".

Image, audio, and video:

-- At a minumum provide facilities to Treat any unrecognized subtypes as if they were "application/octet-stream".

Application:

-- Offer the ability to remove either of the quoted-printable or base64 encodings defined in this document if they were used and put the resulting information in a user file.

Multipart:

-- Recognize the mixed subtype. Display all relevant information on the message level and the body part header level and then display or offer to display each of the body parts individually.

-- Recognize the "alternative" subtype, and avoid showing the user redundant parts of multipart/alternative mail.

-- Recognize the "multipart/digest" subtype, specifically using "message/rfc822" rather than "text/plain" as the default content-type for encapsulations inside "multipart/digest" entities.

-- Treat any unrecognized subtypes as if they were "mixed".

Message:

-- Recognize and display at least the primary (RFC822) encapsulation in such a way as to preserve any recursive structure, that is, displaying or offering to display the encapsulated data in accordance with its Content-type.

-- Treat any unrecognized subtypes as if they were "application/octet-stream".

(5) Upon encountering any unrecognized Content-Type, an implementation must treat it as if it had a Content-Type of "application/octet-stream" with no parameter sub-arguments. How such data are handled is up to an implementation, but likely options for handling such unrecognized data include offering the user to write it into a file (decoded from its mail transport format) or offering the user to name a program to which the decoded data should be passed as input.

A user agent that meets the above conditions is said to be MIME-conformant. The meaning of this phrase is that it is assumed to be "safe" to send virtually any kind of properly-marked data to users of such mail systems, because such systems will at least be able to treat the data as undifferentiated binary, and will not simply splash it onto the screen of unsuspecting users.

There is another sense in which it is always "safe" to send data in a format that is MIME-conformant, which is that such data will not break or be broken by any known systems that are conformant with RFC 821 and RFC 822. User agents that are MIME-conformant have the additional guarantee that the user will not be shown data that were never intended to be viewed as text.

Appendix B -- Guidelines For Sending Email Data

Internet email is not a perfect, homogeneous system. Mail may become corrupted at several stages in its travel to a final destination. Specifically, email sent throughout the Internet may travel across many networking technologies. Many networking and mail technologies do not support the full functionality possible in the SMTP transport environment. Mail traversing these systems is likely to be modified in such a way that it can be transported.

There exist many widely-deployed non-conformant MTAs in the Internet. These MTAs, speaking the SMTP protocol, alter messages on the fly to take advantage of the internal data structure of the hosts they are implemented on, or are just plain broken.

The following guidelines may be useful to anyone devising a data format (Content-Type) that will survive the widest range of networking technologies and known broken MTAs unscathed. Note that anything encoded in the base64 encoding will satisfy these rules, but that some well-known mechanisms, notably the UNIX uuencode facility, will not. Note also that anything encoded in the Quoted-Printable encoding will survive most gateways intact, but possibly not some gateways to systems that use the EBCDIC character set.

(1) Under some circumstances the encoding used for data may change as part of normal gateway or user agent operation. In particular, conversion from base64 to quoted-printable and vice versa may be necessary. This may result in the confusion of CRLF sequences with line breaks in text bodies. As such, the persistence of CRLF as something other than a line break must not be relied on.

(2) Many systems may elect to represent and store text data using local newline conventions. Local newline conventions may not match the RFC822 CRLF convention -- systems are known that use plain CR, plain LF, CRLF, or counted records. The result is that isolated CR and LF characters are not well tolerated in general; they may

be lost or converted to delimiters on some systems, and hence must not be relied on.

(3) TAB (HT) characters may be misinterpreted or may be automatically converted to variable numbers of spaces. This is unavoidable in some environments, notably those not based on the US-ASCII character set. Such conversion is STRONGLY DISCOURAGED, but it may occur, and mail formats must not rely on the persistence of TAB (HT) characters.

(4) Lines longer than 76 characters may be wrapped or truncated in some environments. Line wrapping and line truncation are STRONGLY DISCOURAGED, but unavoidable in some cases. Applications which require long lines must somehow differentiate between soft and hard line breaks. (A simple way to do this is to use the quoted-printable encoding.)

(5) Trailing "white space" characters (SPACE, TAB (HT)) on a line may be discarded by some transport agents, while other transport agents may pad lines with these characters so that all lines in a mail file are of equal length. The persistence of trailing white space, therefore, must not be relied on.

(6) Many mail domains use variations on the US-ASCII character set, or use character sets such as EBCDIC which contain most but not all of the US-ASCII characters. The correct translation of characters not in the "invariant" set cannot be depended on across character converting gateways. For example, this situation is a problem when sending uuencoded information across BITNET, an EBCDIC system. Similar problems can occur without crossing a gateway, since many Internet hosts use character sets other than US-ASCII internally. The definition of Printable Strings in X.400 adds further restrictions in certain special cases. In particular, the only characters that are known to be consistent across all gateways are the 73 characters that correspond to the upper and lower case letters A-Z and a-z, the 10 digits 0-9, and the following eleven special characters:

"'" (US-ASCII decimal value 39)

"(" (US-ASCII decimal value 40)
")" (US-ASCII decimal value 41)
"+" (US-ASCII decimal value 43)
"," (US-ASCII decimal value 44)
"-" (US-ASCII decimal value 45)
"." (US-ASCII decimal value 46)
"/" (US-ASCII decimal value 47)
":" (US-ASCII decimal value 58)
"=" (US-ASCII decimal value 61)
"?" (US-ASCII decimal value 63)

A maximally portable mail representation, such as the base64 encoding, will confine itself to relatively short lines of text in which the only meaningful characters are taken from this set of 73 characters.

(7) Some mail transport agents will corrupt data that includes certain literal strings. In particular, a period (".") alone on a line is known to be corrupted by some (incorrect) SMTP implementations, and a line that starts with the five characters "From " (the fifth character is a SPACE) are commonly corrupted as well. A careful composition agent can prevent these corruptions by encoding the data (e.g., in the quoted-printable encoding, "=46rom " in place of "From " at the start of a line, and "=2E" in place of "." alone on a line.

Please note that the above list is NOT a list of recommended practices for MTAs. RFC 821 MTAs are prohibited from altering the character of white space or wrapping long lines. These BAD and invalid practices are known to occur on established networks, and implementations should be robust in dealing with the bad effects they can cause.

Appendix C -- A Complex Multipart Example

What follows is the outline of a complex multipart message. This message has five parts to be displayed serially: two introductory plain text parts, an embedded multipart message, a text/enriched part, and a closing encapsulated text message in a non-ASCII character set. The embedded multipart message has two parts to be displayed in parallel, a picture and an audio fragment.

```
  MIME-Version: 1.0
  From: Nathaniel Borenstein <nsb@bellcore.com>
  To: Ned Freed <ned@innosoft.com>
  Date: Fri, 07 Oct 1994 16:15:05 -0700 (PDT)
  Subject: A multipart example
  Content-Type: multipart/mixed;
                boundary=unique-boundary-1

  This is the preamble area of a multipart message.
  Mail readers that understand multipart format
  should ignore this preamble.

  If you are reading this text, you might want to
  consider changing to a mail reader that understands
  how to properly display multipart messages.

  --unique-boundary-1

    ... Some text appears here ...

  [Note that the blank between the boundary and the start
   of the text in this part means no header fields were
   given and this is text in the US-ASCII character set.
   It could have been done with explicit typing as in the
   next part.]

  --unique-boundary-1
  Content-type: text/plain; charset=US-ASCII

  This could have been part of the previous part, but
  illustrates explicit versus implicit typing of body
  parts.
```

```
--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

  ... base64-encoded 8000 Hz single-channel
      mu-law-format audio data goes here ...

--unique-boundary-2
Content-Type: image/gif
Content-Transfer-Encoding: base64

  ... base64-encoded image data goes here ...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>enriched.</italic></bold>
<smaller>as defined in RFC 1563</smaller>

Isn't it
<bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

  ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

Appendix D -- Collected Grammar

This appendix contains the complete BNF grammar for all the syntax specified by this document.

By itself, however, this grammar is incomplete. It refers to several entities that are defined by RFC 822. Rather than reproduce those definitions here, and risk unintentional differences between the two, this document simply refers the reader to RFC 822 for the remaining definitions. Wherever a term is undefined, it refers to the RFC 822 definition.

```
attribute := token

boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

bcharsnospace := DIGIT / ALPHA / "'" / "(" / ")" /
                 "+" / "_" / "," / "-" / "." /
                 "/" / ":" / "=" / "?"

body-part := <"message" as defined in RFC 822, with all
              header fields optional, not starting with the
              specified dash-boundary, and with the
              delimiter not occurring anywhere in the
              message body.  Note that the semantics of a
              part differ from the semantics of a message,
              as described in the text.>

close-delimiter := CRLF dash-boundary "--"

composite-type := "message" / "multipart" / extension-token

content := "Content-Type" ":" type "/" subtype
           *(";" parameter)
           ; Matching of type and subtype is
           ; ALWAYS case-insensitive

dash-boundary := "--" boundary
                 ; boundary taken from Content-Type
                 ; field.
```

```
delimiter := CRLF dash-boundary

description := "Content-Description" ":" *text

discard-text := *(*text CRLF)
                ; To be ignored upon receipt.

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

encapsulation := delimiter [*LWSP-char]
                 CRLF body-part

encoding := "Content-Transfer-Encoding" ":" mechanism

epilogue := discard-text

extension-token := iana-token / ietf-token / x-token

iana-token := <a publicly-defined extension token,
               registered with IANA, as specified in
               RFC REG [REF-REG]>

ietf-token := <a publicly-defined extension token,
               initially registered with IANA and
               subsequently standardized by the IETF>

id := "Content-ID" ":" msg-id

mechanism := "7bit" / "8bit" / "binary" /
             "quoted-printable" / "base64" /
             ietf-token / x-token

multipart-body := preamble dash-boundary
                  [*LWSP-char] CRLF
                  body-part *encapsulation
                  close-delimiter [*LWSP-char]
                  CRLF epilogue

octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
         ; Octet must be used for characters > 127, =,
         ; SPACE, or TAB, and is recommended for any
         ; characters not listed in Appendix B as
         ; "mail-safe".
```

```
parameter := attribute "=" value

preamble := discard-text

ptext := octet / safe-char

quoted-printable := ([*(ptext / SPACE / TAB) ptext] ["="] CRLF)
                    ; Maximum line length of 76 characters
                    ; excluding CRLF

safe-char := <any US-ASCII character except "=",
              SPACE, or TAB>
             ; Characters not listed as "mail-safe" in
             ; Appendix B are also not recommended.

subtype := extension-token

token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
            or tspecials>

tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values

type := discrete-type / composite-type

value := token / quoted-string

version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT

x-token := <The two characters "X-" or "x-" followed, with
            no  intervening white space, by any token>
```

Appendix E -- Summary of the Seven Content-types

Content type: text

Subtypes defined by this document: plain

Important parameters: charset

Encoding notes: quoted-printable generally preferred if an encoding is needed and the character set is mostly a US-ASCII superset.

Security considerations: Rich text formats such as TeX and Troff often contain mechanisms for executing arbitrary commands or file system operations, and should not be used automatically unless these security problems have been addressed. Even plain text may contain control characters that can be used to exploit the capabilities of "intelligent" terminals and cause security violations. User interfaces designed to run on such terminals should be aware of and try to prevent such problems.

Content type: image

Subtypes defined by this document: jpeg, gif

Important parameters: none

Encoding notes: base64 generally preferred

Content type: audio

Subtypes defined by this document: basic

Important parameters: none

Encoding notes: base64 generally preferred

Content type: video

Subtypes defined by this document: mpeg

Important parameters: none

Encoding notes: base64 generally preferred

Content type: application

Subtypes defined by this document: octet-stream, postscript

Important parameters: type, padding

Deprecated parameters: name and conversions were defined in RFC 1341, and have since been deleted.

Encoding notes: base64 preferred for unreadable subtypes.

Security considerations: This type is intended for the transmission of data to be interpreted by locally-installed programs. Severe security problems could result if this type is used to transmit binary programs or programs in general-purpose interpreted languages, such as LISP programs or shell scripts, without taking special precautions. Authors of mail-reading agents are cautioned against giving their systems the power to execute mail-based application data without carefully considering the security implications. While it is certainly possible to define safe application formats and even safe interpreters for unsafe formats, each interpreter should be evaluated separately for possible security problems.

Content type: multipart

Subtypes defined by this document: mixed, alternative, digest, parallel.

Important parameters: boundary

Encoding notes: No content-transfer-encoding other than "7bit", "8bit", or "binary" are permitted.

Content type: message

Subtypes defined by this document: rfc822, partial, external-body

Important parameters: id, number, total, access-type, expiration, size, permission, name, site, directory, mode, server, subject

Encoding notes: Only "7bit" is permitted for "message/partial" or "message/external-body", and only "7bit", "8bit", or "binary" are permitted for other subtypes of "message".

Appendix F -- Canonical Encoding Model

There was some confusion, in earlier drafts of this memo, regarding the model for when email data was to be converted to canonical form and encoded, and in particular how this process would affect the treatment of CRLFs, given that the representation of newlines varies greatly from system to system. For this reason, a canonical model for encoding is presented below.

The process of composing a MIME entity can be modeled as being done in a number of steps. Note that these steps are roughly similar to those steps used in PEM [RFC1421] and are performed for each "innermost level" body:

(1) Creation of local form.

The body to be transmitted is created in the system's native format. The native character set is used, and where appropriate local end of line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the content type.

(2) Conversion to canonical form.

The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific content type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various content types. If character set conversion is involved, however, care must be taken to understand the semantics of the content-type, which may have strong implications for

any character set conversion, e.g. with regard to syntactically meaningful characters in a text subtype other than "plain".

For example, in the case of text/plain data, the text must be converted to a supported character set and lines must be delimited with CRLF delimiters in accordance with RFC 822. Note that the restriction on line lengths implied by RFC 822 is eliminated if the next step employs either quoted-printable or base64 encoding.

(3) Apply transfer encoding.

A Content-Transfer-Encoding appropriate for this body is applied. Note that there is no fixed relationship between the content type and the transfer encoding. In particular, it may be appropriate to base the choice of base64 or quoted-printable on character frequency counts which are specific to a given instance of a body.

(4) Insertion into entity.

The encoded object is inserted into a MIME entity with appropriate headers. The entity is then inserted into the body of a higher-level entity (message or multipart) if needed.

It is vital to note that these steps are only a model; they are specifically NOT a blueprint for how an actual system would be built. In particular, the model fails to account for two common designs:

(1) In many cases the conversion to a canonical form prior to encoding will be subsumed into the encoder itself, which understands local formats directly. For example, the local newline convention for text bodies might be carried through to the encoder itself along with knowledge of what that format is.

(2) The output of the encoders may have to pass through one or more additional steps prior to being transmitted as a message. As such, the output of the encoder may not be conformant with the formats specified by RFC 822.

In particular, once again it may be appropriate for the converter's output to be expressed using local newline conventions rather than using the standard RFC 822 CRLF delimiters.

Other implementation variations are conceivable as well. The vital aspect of this discussion is that, in spite of any optimizations, collapsings of required steps, or insertion of additional processing, the resulting messages must be consistent with those produced by the model described here. For example, a message with the following header fields:

```
  Content-type: text/foo; charset=bar
  Content-Transfer-Encoding: base64
```

must be first represented in the text/foo form, then (if necessary) represented in the "bar" character set, and finally transformed via the base64 algorithm into a mail-safe form.

Appendix G -- Changes from RFC 1521

This document is a revision of RFC 1521. For the convenience of those familiar with RFC 1521, the changes from that document are summarized in this appendix. For further history, note that Appendix H in RFC 1521 specified how that document differed from its predecessor, RFC 1341.

(1) This document has been completely reformatted. This was done to improve the quality of the plain text version of this document, which is required to be the reference copy.

(2) BNF describing the overall structure of MIME message and part headers has been added. This is a documentation change only -- the underlying syntax has not changed in any way.

(3) The specific BNF for the seven content types in MIME has been removed. This BNF was incorrect, incomplete, amd inconsistent with the type-indendependent BNF. And since the type-independent BNF already fully specifies the syntax of the various MIME headers, the type-specific BNF was, in the final analysis, completely unnecessary and caused more problems than it solved.

(4) The more specific "US-ASCII" character set name has replaced the use of the term ASCII in many parts of this specification.

(5) The informal concept of a primary subtype has been removed.

(6) The term "object" was being used inconsistently. This term has been replaced with the more precise terms "body", "body part", and "entity" where appropriate.

(7) The BNF for the multipart content-type has been rearranged to make it clear that the CRLF preceeding the boundary marker is actually part of the marker itself rather than the preceeding body part.

(8) In the rules on reassembling "message/partial" MIME entities, "Subject" is added to the list of headers to take from the inner message, and the example is modified to clarify this point.

(9) In the discussion of the application/postscript type, an additional paragraph has been added warning about possible interoperability problems caused by embedding of binary data inside a PostScript MIME entity.

(10) Added a clarifying note to the basic syntax rules for Content-Type to make it clear that the following two forms:

```
Content-type: text/plain; charset=us-ascii (comment)

Content-type: text/plain; charset="us-ascii"
```

are completely equivalent.

(11) The following sentence has been removed from the discussion of the MIME-Version header: "However, conformant software is encouraged to check the version number and at least warn the user if an unrecognized MIME-version is encountered."

(12) A typo was fixed that said "application/external-body" instead of "message/external-body".

(13) The definition of a character set has been reorganized to make the requirements clearer.

(14) The definitions of "7bit" and "8bit" have been tightened so that use of bare CR, LF, and NUL characters are no longer allowed.

(15) The definition of canonical text in MIME has been tightened so that line breaks must be represented by a CRLF sequence. CR and LF characters are not allowed outside of this usage. The definition of quoted-printable encoding has been altered accordingly.

(16) Prose was added to clarify the use of the "7bit", "8-bit", and "binary" transfer-encodings on multipart or message entities encapsulating "8bit" or "binary" data.

(17) In Appendix A, "multipart/digest" support was added to the list of requirements for minimal MIME conformance. Also, the requirement for "message/rfc822" support were strengthened to clarify the importance of recognizing recursive structure.

(18) The various restrictions on subtypes of "message" are now specified entirely on a subtype by subtype basis.

(19) The definition of "message/rfc822" was changed to indicate that at least one of the "From", "Subject", or "Date" headers must be present.

(20) The required handling of unrecognized subtypes as "application/octet-stream" has been made more explicit in both the type definitions sections and the conformance guidelines.

(21) Examples using text/richtext were changed to text/enriched.

(22) The BNF definition of subtype has been changed to make it clear that either an IANA registered subtype or a nonstandard "X-" subtype must be used in a Content-Type header field.

(23) The use of escape and shift mechanisms in the US-ASCII and ISO-8859-X character sets this specification defines has been clarified: Such mechanisms should never be used in conjunction with these character sets and their effect if they are used is undefined.

(24) The definition of the AFS access-type for message/external-body has been removed.

(25) Entities that are simply registered for use and those that are standardized by the IETF are now distinguished in the MIME BNF.

(26) The handling of the combination of multipart/alternative and message/external-body is now specifically addressed.

Appendix H -- References

[ATK]
Borenstein, Nathaniel S., Multimedia Applications Development with the Andrew Toolkit, Prentice-Hall, 1990.

[GIF]
Graphics Interchange Format (Version 89a), Compuserve, Inc., Columbus, Ohio, 1990.

[ISO-2022]
International Standard -- Information Processing -- ISO 7-bit and 8-bit Coded Character Sets -- Code Extension Techniques, ISO 2022:1986.

[ISO-8859]
International Standard -- Information Processing -- 8-bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1, ISO 8859-1:1987. Part 2: Latin alphabet No. 2, ISO 8859-2, 1987. Part 3: Latin alphabet No. 3, ISO 8859-3, 1988. Part 4: Latin alphabet No. 4, ISO 8859-4, 1988. Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988. Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987. Part 7: Latin/Greek alphabet, ISO 8859-7, 1987. Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988. Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

[ISO-646]
International Standard -- Information Processing -- ISO 7-bit Coded Character Set For Information Interchange, ISO 646:1983.

[MPEG]
Video Coding Draft Standard ISO 11172 CD, ISO IEC/TJC1/SC2/WG11 (Motion Picture Experts Group), May, 1991.

[PCM]
CCITT, Fascicle III.4 - Recommendation G.711, "Pulse Code Modulation (PCM) of Voice Frequencies", Geneva, 1972.

[POSTSCRIPT]
Adobe Systems, Inc., PostScript Language Reference Manual, Addison-Wesley, 1985.

[POSTSCRIPT2]
Adobe Systems, Inc., PostScript Language Reference Manual, Addison-Wesley, Second Edition, 1990.

[RFC-783]
Sollins, K.R., "TFTP Protocol (revision 2)", RFC-783, MIT, June 1981.

[RFC-821]
Postel, J.B., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[RFC-822]
Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[RFC-934]
Rose, M., and E. Stefferud, "Proposed Standard for Message Encapsulation", RFC 934, Delaware and NMA, January 1985.

[RFC-959]
Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, USC/Information Sciences Institute, October 1985.

[RFC-1049]
Sirbu, M., "Content-Type Header Field for Internet Messages", STD 11, RFC 1049, CMU, March 1988.

[RFC-1154]
Robinson, D. and R. Ullmann, "Encoding Header Field for Internet Messages", RFC 1154, Prime Computer, Inc., April 1990.

[RFC-1341]
Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, Bellcore, Innosoft, June 1992.

[RFC-1342]
Moore, K., "Representation of Non-Ascii Text in Internet Message Headers", RFC 1342, University of Tennessee, June 1992.

[RFC-1344]
Borenstein, N., "Implications of MIME for Internet Mail Gateways", RFC 1344, Bellcore, June 1992.

[RFC-1345]
Simonsen, K., "Character Mnemonics & Character Sets", RFC 1345, Rationel Almen Planlaegning, June 1992.

[RFC-1421]
Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encryption and Authentication Procedures", RFC 1421, IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1422]
Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", RFC 1422, IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1423]
Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers", IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1424]
Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV -- Key Certification and Related Services", IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1521]
Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September, 1993.

[RFC-1522]
Moore, K., "Representation of Non-ASCII Text in Internet Message Headers", RFC 1522, University of Tennessee, September 1993.

[RFC-1524]
Borenstein, N., "A User Agent Configuration Mechanism for Multimedia Mail Format Information", RFC 1524, Bellcore, September 1993.

[RFC-1563]
Borenstein, N., "The text/enriched MIME Content-type", RFC 1563, Bellcore, January, 1994.

[RFC-1652]
Klensin, J., (WG Chair), Freed, N., (Editor), Rose, M., Stefferud, E., and Crocker, D., "SMTP Service Extension for 8bit-MIME transport", RFC 1652, United Nations Universit, Innosoft, Dover Beach Consulting, Inc., Network Management Associates, Inc., The Branch Office, February 1993.

[RFC-1700]
Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[RFC-MIME-HEADERS]
Moore, K., "Representation of Non-Ascii Text in Internet Message Headers", RFC MIME-HEADERS, University of Tennessee, ?.

[RFC-REG]
Postel, J., "Media Type Registration Procedure", RFC REG, ?.

[US-ASCII]
Coded Character Set -- 7-Bit American Standard Code for Information Interchange, ANSI X3.4-1986.

[X400]
Schicker, Pietro, "Message Handling Systems, X.400", Message Handling Systems and Distributed Applications, E. Stefferud, O-j. Jacobsen, and P. Schicker, eds., North-Holland, 1989, pp. 3-41.