Network Working Group   Nathaniel Borenstein
Internet Draft   Ned Freed
                    <draft-ietf-822ext-mime-imt-04.txt>

             Multipurpose Internet Mail Extensions
                       (MIME) Part Two:

                         Media Types

                         March 1996

                      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 US-ASCII message headers,
but which leaves the message content, or message body, as flat
US-ASCII text. This set of documents, collectively called the
Multipurpose Internet Mail Extensions, or MIME, redefines the
format of messages to allow for

 (1) textual message bodies in character sets other than
     US-ASCII,

 (2) an extensible set of different formats for non-textual
     message bodies,

 (3) multi-part message bodies, and

 (4) textual header information in character sets other than
     US-ASCII.

These documents are 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, these
documents are largely orthogonal to (rather than a revision
of) RFC 822.

The initial document in this set, RFC MIME-IMB, specifies the
various headers used to describe the structure of MIME
messages. This second document defines the general structure
of the MIME media typing system and defines an initial set of
media types. The third document, RFC MIME-HEADERS, describes
extensions to RFC 822 to allow non-US-ASCII text data in
Internet mail header fields. The fourth document, RFC MIME-
REG, specifies various IANA registration procedures for MIME-
related facilities. The fifth and final document, RFC MIME-
CONF, describes MIME conformance criteria as well as providing
some illustrative examples of MIME message formats,
acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521 and 1522, which
themselves were revisions of RFCs 1341 and 1342. An appendix
in RFC MIME-CONF describes differences and changes from
previous versions.

[2](). **Table of Contents**

3. **Introduction**

The first document in this set, RFC MIME-IMB, defines a number
of header fields, including Content-Type. The Content-Type
field is used to specify the nature of the data in the body of
a MIME entity, by giving media type and subtype identifiers,
and by providing auxiliary information that may be required
for certain media 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 media type is used to declare the
general type of data, while the subtype specifies a specific
format for that type of data. Thus, a media 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 media subtype, and as such do
not fundamentally affect the nature of the content. The set
of meaningful parameters depends on the media type and
subtype. Most parameters are associated with a single
specific subtype. However, a given top-level media type may
define parameters which are applicable to any subtype of that
type. Parameters may be required by their defining media type
or subtype or they may be optional. MIME implementations must
also ignore any parameters whose names they do not recognize.

MIME's Content-Type header field and media type mechanism has
been carefully designed to be extensible, and it is expected
that the set of media type/subtype pairs and their associated
parameters will grow significantly over time. Several other
MIME facilities, such as transfer encodings and
message/external-body access types, 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 various areas of extensibility. The
registration process for these areas is described in a
companion document, RFC MIME-REG.

The initial seven standard top-level media type are defined
and described in the remainder of this document.

[4](#). **Definition of a Top-Level Media Type**

The definition of a top-level media 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 entities of this
         top-level type, if any, and

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

[5](#). **Overview Of The Initial Top-Level Media Types**

The five discrete top-level media types are:

  (1)   text -- textual information. The subtype "plain" in
         particular indicates plain text containing no
         formatting commands or directives of any sort. Plain
         text is intended to be displayed "as-is". 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 of text 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, with a further
         revision in 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. An initial
         subtype is defined for the widely-used image format
         JPEG.

   (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 an 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) messaging, 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
         messaging. These issues are discussed later in this
         document.

The two composite top-level media types are:

 (1)   multipart -- data consisting of multiple entities 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 has a default type of "message/rfc822".

 (2)   message -- an encapsulated message. A body of media
       type "message" is itself all or a portion of some kind
       of message object. Such objects may or may not in turn
       contain other entities. The "rfc822" subtype is used
       when the encapsulated 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 transport facilities in one piece. Another
       subtype, "external-body", is defined for specifying
       large bodies by reference to an external data source.

It should be noted that the list of media 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.

6. Discrete Media Type Values

Five of the seven initial media type values refer to discrete
bodies. The content of these types must be handled by non-MIME
mechanisms; they are opaque to MIME processors.

6.1. Text Media Type

The text media 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 text
subtypes, notably including the subtype "text/plain", which
indicates plain text that doesn't contain any formatting
commands or directives.

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. 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.

NOTE: The proper interpretation of line breaks when a body is
displayed depends on the media type. In particular, while it
is appropriate to treat a line break as a transition to a new
line when displaying a text/plain body, this treatment is
actually incorrect for other subtypes of text like
text/enriched [RFC-1563]. Similarly, whether or not line
breaks should be added during display operations is also a
function of the media type. It should not be necessary to add
any line breaks to display text/plain correctly, whereas
proper display of text/enriched requires the appropriate
addition of line breaks.

6.1.2. 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 to 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 MIME-IMB. 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.

Note that if the specified character set includes 8bit 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 [RFC-821].

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 as a media
type parameter in the Content-Type header field. "US-ASCII"
does not indicate an arbitrary -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 from
this document 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 media 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 8bit 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)." Aside from these conventions, any use
of the control characters or DEL in a body must occur within
the context 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 Internet 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 7bit or 8bit 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
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, composition 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 resulting entity correctly.

6.1.3. **Plain Subtype**

The simplest and most important subtype of text is "plain".
This indicates plain text that does not contain any formatting
commands or directives. Plain text is intended to be displayed
"as-is", that is, no formatting operations of any sort other
than support for the indicated character set should be
necessary for proper display. The default media type of
"text/plain; charset=us-ascii" for Internet mail 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.

6.1.4. **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".

6.2. **Image Media Type**

A media type of "image" indicates that the body contains an
image. The subtype names the specific image format. These
names are not case sensitive. An initial subtype is "jpeg" for
the JPEG format using JFIF encoding [JPEG].

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 MIME-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 secure and robust general-purpose image viewing
application, if such an application is available.

NOTE: Using of a generic-purpose image viewing application
this way inherits the security problems of the most dangerous
type supported by the application.

6.3. Audio Media Type

A media 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 8bit 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.

6.4. Video Media Type

A media 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.

[6.5]. Application Media Type

The "application" media 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 some type of
application program. This is information which must be
processed by an application before it is viewable or usable by
a user. Expected uses for the application media type include
file transfer, spreadsheets, data for mail-based scheduling
systems, and languages for "active" (computational) material.
(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 media
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 additional
material based on that dialog. More generally, there have
been several "active" messaging languages developed in which
programs in a suitably specialized language are transported to
a remote location and automatically run in the recipient's
environment.

Such applications may be defined as subtypes of the
"application" media 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.

6.5.1. **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 8bit 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 an
application/octet-stream entity 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, 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 message body as input.

6.5.2. **PostScript Subtype**

A media 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 media 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 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 MIME
readers.

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

  (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 Internet mail, 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.

6.5.3. 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".

7. Composite Media 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.

7.1. **Multipart Media Type**

In the case of multipart entities, in which one or more
different sets of data are combined in a single body, a
"multipart" media type field must appear in the entity's
header. The body must then contain one or more body parts,
each preceded by a boundary delimiter line, and the last one
followed by a closing boundary delimiter line. After its
boundary delimiter line, each body part then consists 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 an entity and hence 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 usually 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 may be ignored in body parts. Although
they should generally be retained if at all possible, 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
JPEG 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/jpeg" 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
parts actually are messages, a "digest" subtype is also

defined.)

As stated previously, each body part is preceded by a boundary
delimiter line that contains the boundary delimiter. The
boundary delimiter MUST NOT appear inside any of the
encapsulated parts, on a line by itself or as the prefix of
any line. This implies that it is crucial that the composing
agent be able to choose and specify a unique boundary
parameter value that does not contain the boundary parameter
value of an enclosing multipart as a prefix.

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 those of an unrecognized
subtype.

As stated in the definition of the Content-Transfer-Encoding
field [MIME-IMB], no encoding other than "7bit", "8bit", or
"binary" is permitted for entities of type "multipart". The
multipart boundary delimiters and header fields are always
represented as 7bit 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.

7.1.1. **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 RFC MIME-CONF.

The Content-Type field for multipart entities requires one
parameter, "boundary". The boundary delimiter line is then
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, optional linear
whitespace, and a terminating CRLF.

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 boundary parameter values 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 boundary delimiter MUST occur at the beginning of a line,
i.e., following a CRLF, and the initial CRLF is considered to
be attached to the boundary delimiter line rather than part of
the preceding part. The boundary may be followed by zero or

more characters of linear whitespace. It is then terminated by
either 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. If no Content-Type field is present it is
assumed to be of message/rfc822 in a multipart/digest and
text/plain otherwise.

NOTE: The CRLF preceding the boundary delimiter 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 boundary
delimiter line, the first of which is part of the preceding
body part, and the second of which is part of the
encapsulation boundary.

Boundary delimiters must not appear within the encapsulated
material, and must be no longer than 70 characters, not
counting the two leading hyphens.

The boundary delimiter line following the last body part is a
distinguished delimiter that indicates that no further body
parts will follow. Such a delimiter line is identical to the
previous delimiter lines, with the addition of two more
hyphens after the boundary parameter value.

   --gc0pJq0M:08jU534c0p--

NOTE TO IMPLEMENTORS: Boundary string comparisons must
compare the boundary value with the beginning of each
candidate line. An exact match of the entire candidate line
is not required; it is sufficient that the boundary appear in
its entirety following the CRLF.

There appears to be room for additional information prior to
the first boundary delimiter line and following the final
boundary delimiter line. These areas should generally be left
blank, and implementations must ignore anything that appears
before the first boundary delimiter line 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 boundary delimiters must not appear in the body
parts being encapsulated, a user agent must exercise care to
choose a unique boundary parameter value. The boundary
parameter value in the example above could have been the
result of an algorithm designed to produce boundary delimiters
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" boundary delimiters
for a recipient with an old user agent, but would require more
attention to the possibility that the boundary delimiter might
appear at the beginning of some line in the encapsulated part.
The simplest boundary delimiter line possible is something
like "---", with a closing boundary delimiter line 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 composition agents 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 media 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 RFC MIME-CONF for an example of nested
multipart entities.

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

NOTE: Experience has shown that a multipart media type with a
single body part is useful for sending non-text media types.
It has the advantage of providing the preamble as a place to
include decoding instructions. In addition, a number of SMTP
gateways move or remove the MIME headers, and a clever MIME
decoder can take a good guess at multipart boundaries even in
the absence of the Content-Type header and thereby successful
decode the message.

The only mandatory global parameter for the multipart media
type is the boundary parameter, which consists of 1 to 70
characters from a set of characters known to be very robust
through mail gateways, and NOT ending with white space. (If a
boundary delimiter line 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 the value of
                   ; boundary parameter of the
                   ; Content-Type field.

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

  transport-padding := *LWSP-char
                       ; Composers MUST NOT generate
                       ; non-zero length transport
                       ; padding, but receivers MUST
                       ; be able to handle padding
                       ; added by message transports.

  encapsulation := delimiter transport-padding
                  CRLF body-part

  delimiter := CRLF dash-boundary

  close-delimiter := delimiter "--"
```

```
  preamble := discard-text

  epilogue := discard-text

  discard-text := *(*text CRLF) *text
                  ; May be ignored or discarded.

  body-part := MIME-part-headers [CRLF *OCTET]
               ; Lines in a body-part must not start
               ; with the specified dash-boundary and
               ; the delimiter must not appear anywhere
               ; in the body part.  Note that the
               ; semantics of a body-part differ from
               ; the semantics of a message, as
               ; described in the text.

  OCTET := <any 0-255 octet value>
```

IMPORTANT: The free insertion of linear-white-space and RFC
822 comments 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 exist that 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. It is recommended
that those wishing to provide more structured or integrated
multipart messaging facilities should define subtypes of
multipart that are syntactically identical but define
relationships between the various parts. For example, subtypes
of multipart could be defined that include a distinguished
part which in turn is used to specify the relationships

between the other parts, probably referring to them by their
Content-ID field. Old implementations will not recognize the
new subtype if this approach is used, but will treat it as
multipart/mixed and will thus be able to show the user the
parts that are recognized.

7.1.2. 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 entity may not have any terminating boundary marker,
and can turn up operationally due to mail system malfunctions.

It is essential that such entities 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.

7.1.3. 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".

7.1.4. Alternative Subtype

The multipart/alternative type is syntactically identical to
multipart/mixed, but the semantics are different. In
particular, each of the body 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 references, 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 a
message 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 systems 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 viewer. While this approach does impose some
burden on conformant MIME viewers, 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 a message 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.

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.

7.1.5. **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.

Note: Though it is possible to specify a Content-Type value
for a body part in a digest which is other than
"message/rfc822", such as a text/plain part containing a
description of the material in the digest, actually doing so
is undesireble. The "multipart/digest" Content-Type is
intended to be used to send collections of messages. If a
"text/plain" part is needed, it should be included as a
seperate part of a "multipart/mixed" message.

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/mixed;
                 boundary="---- main boundary ----"

   ------ main boundary ----

     ...Introductory text or table of contents...

   ------ main boundary ----
   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 ------

   ------ main boundary ------
```

7.1.6. 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.

7.1.7. **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".

7.2. **Message Media Type**

It is frequently desirable, in sending mail, to encapsulate
another mail message. A special media 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. These operations are explicitly
forbidden for the encapsulated headers embedded in the bodies
of messages of type "message."

7.2.1. RFC822 Subtype

A media 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.

It should be noted that, despite the use of the numbers "822",
a message/rfc822 entity isn't restricted to material in strict
conformance to RFC822. Such entities can also include enhanced
information as defined in this document. In other words, a
message/rfc822 message could well be a News article or a MIME
message.

No encoding other than "7bit", "8bit", or "binary" is
permitted for the body of a "message/rfc822" entity. 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.

7.2.2. Partial Subtype

The "partial" subtype is defined to allow large entities to be
delivered as several separate pieces of mail and automatically
reassembled by a 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. The media type "message/partial"
thus indicates that the body contains a fragment of a larger
entity.

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 8bit 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 7bit transport environment,
there would be no way to properly encode them for the 7bit
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 entities of type message/partial
must always have a content-transfer-encoding of 7bit (the
default). In particular, even in environments that support
binary or 8bit transport, the use of a content-transfer-
encoding of "8bit" or "binary" is explicitly prohibited for
MIME entities of type message/partial. This in turn implies
that the inner message must not use "8bit" or "binary"
encoding.

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.

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 fragments 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 fragment number, which
indicates where this fragment fits into the sequence of
fragments. The third, "total", another integer, is the total
number of fragments. This third subfield is required on the
final fragment, and is optional (though encouraged) on the
earlier fragments. Note also that these parameters may be
given in any order.

Thus, the second piece of a 3-piece 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 the third piece MUST specify the total number of
fragments:

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

Note that fragment numbering begins with 1, not 0.

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

7.2.2.1. **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 pieces 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
        message, 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)   The header fields in the enclosed message which start
        with "Content-", plus the "Subject", "Message-ID",
        "Encrypted", and "MIME-Version" fields, 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 the
"Subject", "Message-ID", "Encrypted", and "MIME-
Version" fields) will be ignored and dropped.

(3)   All of the header fields from the second and any
subsequent enclosing messages are discarded by the
reassembly process.

7.2.2.2. Fragmentation and Reassembly Example

If an audio message is broken into two pieces, the first piece
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 ...
```

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.

[7.2.3](#). **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 a MIME 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.jpeg"

  Content-type: image/jpeg
  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
7bit (the default). In particular, even in environments that
support binary or 8bit transport, the use of a content-
transfer-encoding of "8bit" or "binary" is explicitly
prohibited for entities of type message/external-body.

7.2.3.1. **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 MIME-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.

7.2.3.2. 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.

7.2.3.3. 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 mail address.

7.2.3.4. 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.

7.2.3.5. **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 addr-spec 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 in the content-type
header field. Instead, it is provided as the "phantom body"
when the media 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 mail servers must use the
same Content-ID field on the returned message that was used in
the original message/external-body entities, to facilitate
such matching.

7.2.3.6. **External-Body Security Issues**

Message/external-body entities give rise to two important
security issues:

(1)   Accessing data via a message/external-body reference
       effectively results in the message recipient performing
       an operation that was specified by the message
       originator. It is therefore possible for the message
       originator to trick a recipient into doing something
       they would not have done otherwise. For example, an
       originator could specify a action that attempts
       retrieval of material that the recipient is not
       authorized to obtain, causing the recipient to
       unwittingly violate some security policy. For this
       reason, user agents capable of resolving external
       references must always take steps to describe the
       action they are to take to the recipient and ask for
       explicit permisssion prior to performing it.

       The 'mail-server' access-type is particularly
       vulnerable, in that it causes the recipient to send a
       new message whose contents are specified by the
       original message's originator. Given the potential for
       abuse, any such request messages that are constructed
       should contain a clear indication that they were
       generated automatically (e.g. in a Comments: header
       field) in an attempt to resolve a MIME
       message/external-body reference.

(2)   MIME will sometimes be used in environments that
       provide some guarantee of message integrity and
       authenticity. If present, such guarantees may apply
       only to the actual direct content of messages -- they
       may or may not apply to data accessed through MIME's
       message/external-body mechanism. In particular, it may
       be possible to subvert certain access mechanisms even
       when the messaging system itself is secure.

       It should be noted that this problem exists either with
       or without the availabilty of MIME mechanisms. A
       casual reference to an FTP site containing a document
       in the text of a secure message brings up similar
       issues -- the only difference is that MIME provides for
       automatic retrieval of such material, and users may
       place unwarranted trust is such automatic retrieval
       mechanisms.

7.2.3.7. **Examples and Further Explanations**

When the external-body mechanism is used in conjunction with
the multipart/alternative media type it extends the
functionality of multipart/alternative to include the case
where the same entity is provided in the same format but via
different accces mechanisms. When this is done the originator
of the message must order the parts 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 entities of type
message/external-body within the body parts of an enclosing
multipart/alternative entity.

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
media 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 media
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 and Subject fields are overridden, but the From field is
preserved.

Note that since the external bodies are not transported along
with the external body reference, they need not conform to
transport limitations that apply to the reference itself. In
particular, Internet mail transports may impose 7bit and line
length limits, but these do not automatically apply to binary
external body references. 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.

7.2.4. Other Message Subtypes

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

Future subtypes of message intended for use with email should
be restricted to "7bit" encoding. A type other than message
should be used if restriction to "7bit" is not possible.

8. Experimental Media Type Values

A media type value beginning with the characters "X-" is a
private value, to be used by consenting 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. In many cases, a subtype of
application will be more appropriate than a new top-level
type.

[9](#). Summary

The five discrete media types provide provide a standardized
mechanism for tagging entities as audio, image, or several
other kinds of data. The composite "multipart" and "message"
media types allow mixing and hierarchical structuring of
entities of different types in a single message. 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 media types are
defined for general use by consenting user agents, notably
message/partial, and message/external-body.

[10](#). Security Considerations

Security issues are discussed in the context of the
application/postscript type, the message/external-body type,
and in RFC MIME-REG. Implementors should pay special
attention to the security implications of any media 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 media types with remote execution
capabilities.

11. Authors' Addresses

For more information, the authors of this document are best
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: +1 818 919 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
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

Email: Greg.Vaudreuil@Octel.Com

                    Appendix A -- 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 by
name to several syntax rules 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.

  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
               body part. Note that the semantics of a
               part differ from the semantics of a message,
               as described in the text.>

  close-delimiter := delimiter "--"

  dash-boundary := "--" boundary
                   ; boundary taken from the value of
                   ; boundary parameter of the
                   ; Content-Type field.

  delimiter := CRLF dash-boundary

  discard-text := *(*text CRLF)
                   ; May be ignored or discarded.

```
encapsulation := delimiter transport-padding
                CRLF body-part

epilogue := discard-text

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

preamble := discard-text

transport-padding := *LWSP-char
                     ; Composers MUST NOT generate
                     ; non-zero length transport
                     ; padding, but receivers MUST
                     ; be able to handle padding
                     ; added by message transports.
```