Network Working Group
Request for Comments: 1341
N. Borenstein, Bellcore
N. Freed, Innosoft
June 1992

MIME (Multipurpose Internet Mail Extensions):

Mechanisms for Specifying and Describing the Format of Internet Message Bodies

Status of this Memo

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

Abstract

RFC 822 defines a message representation protocol which specifies considerable detail about message headers, but which leaves the message content, or message body, as flat 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 and RFC 1049, but extends and revises that work. 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 objects 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. It is recognized that such extensions are necessary, and they are the subject of a companion document [RFC -1342].

A table of contents appears at the end of this document.

1 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 [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 seven-bit ASCII. This forces users to convert any non-textual data that they may wish to send into seven-bit bytes representable as printable 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 1113, 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 that either X.400 non-textual body parts should be converted to (not encoded in) an ASCII format, or that they should 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’s UA 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 is 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.

    2.a. 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.b. 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.c. 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.

    2.d. A "message" Content-Type value, for encapsulating a mail message.

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

    2.f. An "audio" Content-Type value, for transmitting audio or voice data.

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

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 optional header fields that can be used to further describe the data in a message body, the Content-ID and Content-Description header fields.

MIME has been carefully designed as an extensible mechanism, and it is expected that the set of content-type/subtype pairs and their associated parameters will grow significantly with time. Several other MIME fields, notably including character set names, 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 defines a registration process which uses the Internet Assigned Numbers Authority (IANA) as a central registry for such values. Appendix F provides details about how IANA registration is accomplished.

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.

2 Notations, Conventions, and Generic BNF Grammar

This document is being published in two versions, one as plain ASCII text and one as PostScript. The latter is recommended, though the textual contents are identical. An Andrew-format copy of this document is also available from the first author (Borenstein).

Although the mechanisms specified in this document are all described in prose, most are also described formally in the modified 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 modified BNF notation.

Some of the modified 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 combining the collected grammar appendix of this document with that of RFC 822.

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

The term "character set", wherever it is used in this document, refers to a coded character set, in the sense of ISO character set standardization work, and must not be misinterpreted as meaning "a set of characters."

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 of type "message".

The term "body part", in this document, means one of the parts of 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 overal structure of a MIME message is indeed recursive.

In this document, all numeric and octet values are given in decimal notation.

It must be noted that 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: This document has been carefully formatted for ease of reading. The PostScript version of this document, in particular, places notes like this one, which may be skipped by the reader, in a smaller, italicized, font, and indents it as well. In the text version, only the indentation is preserved, so if you are reading the text version of this you might consider using the PostScript version instead. However, all such notes will be indented and preceded by "NOTE:" or some similar introduction, even in the text version.

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 be skipped by those who are focused entirely on building a compliant implementation, but may be of use to those who wish to understand why this document is written as it is.

For ease of recognition, all BNF definitions have been placed in a fixed-width font in the PostScript version of this document.

3 The 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:

```
MIME-Version := text
```

Thus, future format specifiers, which might replace or extend "1.0", are (minimally) constrained by the definition of "text", which appears in RFC 822.

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-compliant.

4 The 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 set of meaningful parameters differs for the different types. The ordering of parameters is *not* significant. Among the defined parameters is a "charset" parameter by which the character set used in the body may be declared. Comments are allowed in accordance with RFC 822 rules for structured header fields.

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 audio, image, text, and video, should not contain embedded information that is really of a different type. Such compound types should be represented using the "multipart" or "application" types.

Parameters are modifiers of the content-subtype, and do not fundamentally affect the requirements of the host system. Although most parameters make sense only with certain content-types, others are "global" in the sense that they might apply to any subtype. For example, the "boundary" parameter makes sense only for the "multipart" content-type, but the "charset" parameter might make sense with several content-types.

An initial set of seven Content-Types is defined by this document. This set of top-level names 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 an extension to this standard. If another primary 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.

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

```
Content-Type := type "/" subtype *[";" parameter]

type :=       "application"   / "audio"
            / "image"         / "message"
            / "multipart"     / "text"
            / "video"         / x-token

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

subtype := token

parameter := attribute "=" value

attribute := token

value := token / quoted-string

token := 1*<any 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 "=".

Note also that a subtype specification is MANDATORY. There are no default subtypes.

The type, subtype, and parameter names are not case sensitive. For example, TEXT, Text, and TeXt are all equivalent. Parameter values are normally case sensitive, but certain parameters are interpreted to be case-insensitive, depending on the intended use. (For example, multipart boundaries are case-sensitive, but the "access-type" for message/External-body is not case-sensitive.)

Beyond this syntax, the only 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 Appendix F. Where intended for public use, the formats they refer to must also be defined by a published specification, and possibly offered for standardization.

The seven standard initial predefined Content-Types are detailed in the bulk of this document. They are:

text -- textual information. The primary subtype, "plain", 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. 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 *readable* word processor format. A very simple and portable subtype, richtext, is defined in this document.

multipart -- data consisting of multiple parts of independent data types. Four initial subtypes are defined, including the primary "mixed" subtype, "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".

message -- an encapsulated message. A body of Content-Type "message" is itself a fully formatted RFC 822 conformant message which may contain its own different Content-Type header field. The primary subtype is "rfc822". The "partial" subtype is defined for partial messages, to permit the fragmented transmission of bodies that are thought to be too large to be passed through mail transport facilities. Another subtype, "External-body", is defined for specifying large bodies by reference to an external data source.

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

audio -- audio data, with initial subtype "basic". Audio requires an audio output device (such as a speaker or a telephone) to "display" the contents.

video -- video data. Video requires the capability to display moving images, typically including specialized hardware and software. The initial subtype is "mpeg".

application -- some other kind of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The primary 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. Two additional subtypes, "ODA" and "PostScript", are defined for transporting ODA and PostScript documents in bodies. Other expected uses for "application" include spreadsheets, data for mail-based scheduling systems, and languages for "active" (computational) email. (Note that active email entails several securityconsiderations, which are discussed later in this memo, particularly in the context of application/PostScript.)

Default RFC 822 messages are typed by this protocol as plain text in the US-ASCII character set, which can be explicitly specified as "Content-type: text/plain; charset=us-ascii". If no Content-Type is specified, either by error or by an older user agent, this default is assumed. 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. In the absence of a MIME-Version specification, plain US-ASCII text must still be assumed, 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 the 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 The Content-Transfer-Encoding Header Field

Many Content-Types which could usefully be 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 restricts mail messages to 7-bit US-ASCII data with 1000 character lines.

It is necessary, therefore, to define a standard mechanism for re-encoding such data into a 7-bit short-line format. This document specifies that such encodings will be indicated by a new "Content-Transfer-Encoding" header field. The Content-Transfer-Encoding field is used to indicate the type of transformation that has been used in order to represent the body in an acceptable manner for transport.

Unlike Content-Types, a proliferation of Content-Transfer-Encoding values is undesirable and unnecessary. However, establishing only a single Content-Transfer-Encoding mechanism 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 data. For this reason, at least two encoding mechanisms are necessary: a "readable" encoding and a "dense" encoding.

The Content-Transfer-Encoding field is designed to specify an invertible mapping between the "native" representation of a type of data and a representation that can be readily exchanged using 7 bit mail transport protocols, such as those defined by RFC 821 (SMTP). This field has not been defined by any previous standard. The field’s value is a single token specifying the type of encoding, as enumerated below. Formally:

```
Content-Transfer-Encoding := "BASE64" / "QUOTED-PRINTABLE" /
                             "8BIT"   / "7BIT" /
                             "BINARY" / x-token
```

These values are not case sensitive. That is, Base64 and BASE64 and bAsE64 are all equivalent. An encoding type of 7BIT requires that the body is already in a seven-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.

The values "8bit", "7bit", and "binary" all imply that NO encoding has been performed. However, they are potentially useful as indications of the kind of data contained in the object, and therefore of the kind of encoding that might need to be performed for transmission in a given transport system. "7bit" means that the data is all represented as short lines of US-ASCII data. "8bit" means that the lines are short, but there may be non-ASCII characters (octets with the high-order bit set). "Binary" means that not only may non-ASCII characters be present, but also that the lines are not necessarily short enough for SMTP transport.

The difference between "8bit" (or any other conceivable bit-width token) and the "binary" token is that "binary" does not require adherence to any limits on line length or to the SMTP CRLF semantics, while the bit-width tokens do require such adherence. If the body contains data in any bit-width other than 7-bit, the appropriate bit-width Content-Transfer-Encoding token must be used (e.g., "8bit" for unencoded 8 bit wide data). If the body contains binary data, the "binary" Content-Transfer-Encoding token must be used.

NOTE: The distinction between the Content-Transfer-Encoding values of "binary," "8bit," etc. may seem unimportant, in that all of them really mean "none" -- that is, there has been no encoding of the data for transport. However, clear labeling will be of enormous value to gateways between future mail transport systems with differing capabilities in transporting data that do not meet the restrictions of RFC 821 transport.

As of the publication of this document, there are no standardized Internet transports for which it is legitimate to include unencoded 8-bit or binary data in mail bodies. Thus there are no circumstances in which the "8bit" or "binary" Content-Transfer-Encoding is actually legal on the Internet. However, in the event that 8-bit or binary mail transport becomes a reality in Internet mail, or when this document is used in conjunction with any other 8-bit or binary-capable transport mechanism, 8-bit or binary bodies should be labeled 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 explicitly and strongly discouraged**, as it seems likely to hinder interoperability with little potential benefit. Their use is 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" or "message", the Content-Transfer-Encoding is not permitted to have any value other than a bit width (e.g., "7bit", "8bit", etc.) or "binary".

It should be noted that email is character-oriented, so that the mechanisms described here are mechanisms for encoding arbitrary byte 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 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 Content-Type, which has a "padding" parameter.

The encoding mechanisms defined here explicitly encode all data in 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 should be interpreted to mean that the body is a base64 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 Content-Type that recursively includes other Content-Type fields, notably the "multipart" and "message" Content-Types.** 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.

NOTE ON ENCODING RESTRICTIONS: Though the prohibition against using content-transfer-encodings on data of type multipart or message 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 objects 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 little or no encoding. 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 should 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 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 as Appendix H.

5.1 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 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 ASCII text, the encoded form of the data remains largely recognizable by humans. A body which is entirely 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:

> Rule #1: (General 8-bit representation) Any octet, except those indicating a line break according to the newline convention of the canonical 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 value 12 (ASCII form feed) can be represented by "=0C", and the value 61 (ASCII EQUAL SIGN) can be represented by "=3D". Except when the following rules allow an alternative encoding, this rule is mandatory.
>
> Rule #2: (Literal representation) Octets with decimal values of 33 through 60 inclusive, and 62 through 126, inclusive, MAY be represented as the ASCII characters which correspond to those octets (EXCLAMATION POINT through LESS THAN, and GREATER THAN through TILDE, respectively).

Rule #3: (White Space): Octets with values of 9 and 32 MAY be represented as 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 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.

Rule #4 (Line Breaks): A line break in a text body part, independent of what its representation is following the canonical representation of the data being encoded, must be represented by a (RFC 822) line break, which is a CRLF sequence, in the Quoted-Printable encoding. If isolated CRs and LFs, or LF CR and CR LF sequences are allowed to appear in binary data according to the canonical form, they must be represented using the "=0D", "=0A", "=0A=0D" and "=0D=0A" notations respectively.

Note that many implementation may elect to encode the local representation of various content types directly. 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.

Rule #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. (In theory, an EBCDIC gateway could decode a quoted-printable body and re-encode it using base64, but such gateways do not yet exist.) 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 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 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.

5.2 Base64 Content-Transfer-Encoding

The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that is not 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 based on the one used in Privacy Enhanced Mail applications, as defined in RFC 1113. The base64 encoding is adapted from RFC 1113, with one change: base64 eliminates the "*" mechanism for embedded clear text.

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 byte, and the eighth bit will be the low-order bit in the first 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 output stream (encoded bytes) 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. Output character positions which are not required to represent actual input data are set to 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.

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 should 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.

6 Additional Optional Content- Header Fields

6.1 Optional 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 labeled using the "Content-ID" header field, which is syntactically identical to the "Message-ID" header field:

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

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

6.2 Optional 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.

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

The description is presumed to be given in the US-ASCII character set, although the mechanism specified in [RFC-1342] may be used for non-US-ASCII Content-Description values.

7 The 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 G.

7.1 The Text Content-Type

The text Content-Type is intended for sending material which is principally textual in form. It is the default Content-Type. A "charset" parameter may be used to indicate the character set of the body text. The primary subtype of text is "plain". This indicates plain (unformatted) text. **The default Content-Type for Internet mail 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/richtext".

7.1.1 The charset parameter

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

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

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.

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 Appendix F, although the standardization of their use requires the usual IAB review and approval. 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 seven-bit character code, but specifies that the body uses character coding that uses the exact correspondence of codes to characters specified in ASCII. National use variations of ISO 646 [ISO-646] are NOT 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 [US-ASCII]. Note that the control characters including DEL (0-31, 127) have no defined meaning apart from the combination CRLF (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. It is our hope that ISO 10646 or some other effort will eventually define a single world character set which can then be specified for use in Internet mail, but in the advance of that definition we cannot specify the use of ISO 10646, Unicode, or any other character set whose definition is, as of this writing, incomplete.

The defined charset values are:

US-ASCII -- as defined in [US-ASCII].

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.

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 as described in Appendix F, 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 labeled as being in that subset. This will increase the chances that the recipient will be able to view the mail correctly.

7.1.2 The Text/plain subtype

The primary 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.

7.1.3 The Text/richtext subtype

In order to promote the wider interoperability of simple formatted text, this document defines an extremely simple subtype of "text", the "richtext" subtype. This subtype was designed to meet the following criteria:

1. The syntax must be extremely simple to parse, so that even teletype-oriented mail systems can easily strip away the formatting information and leave only the readable text.

2. The syntax must be extensible to allow for new formatting commands that are deemed essential.

3. The capabilities must be extremely limited, to ensure that it can represent no more than is likely to be representable by the user's primary word processor. While this limits what can be sent, it increases the likelihood that what is sent can be properly displayed.

4. The syntax must be compatible with SGML, so that, with an appropriate DTD (Document Type Definition, the standard mechanism for defining a document type using SGML), a general SGML parser could be made to parse richtext. However, despite this compatibility, the syntax should be far simpler than full SGML, so that no SGML knowledge is required in order to implement it.

The syntax of "richtext" is very simple. It is assumed, at the top-level, to be in the US-ASCII character set, unless of course a different charset parameter was specified in the Content-type field. All characters represent themselves, with the exception of the "<" character (ASCII 60), which is used to mark the beginning of a formatting command. Formatting instructions consist of formatting commands surrounded by angle brackets ("<>", ASCII 60 and 62). Each formatting command may be no more than 40 characters in length, all in US-ASCII, restricted to the alphanumeric and hyphen ("-") characters. Formatting commands may be preceded by a forward slash or solidus ("/", ASCII 47), making them negations, and such negations must always exist to balance the initial opening commands, except as noted below. Thus, if the formatting command "<bold>" appears at some point, there must later be a "</bold>" to balance it. There are only three exceptions to this "balancing" rule: First, the command "<lt>" is used to represent a literal "<" character. Second, the command "<nl>" is used to represent a required line break. (Otherwise, CRLFs in the data are treated as equivalent to a single SPACE character.) Finally, the command "<np>" is used to represent a page break. (NOTE: The 40 character limit on formatting commands does not include the "<", ">", or "/" characters that might be attached to such commands.)

Initially defined formatting commands, not all of which will be implemented by all richtext implementations, include:

Bold -- causes the subsequent text to be in a bold font.
Italic -- causes the subsequent text to be in an italic font.
Fixed -- causes the subsequent text to be in a fixed width font.
Smaller -- causes the subsequent text to be in a smaller font.

Bigger -- causes the subsequent text to be in a bigger font.
Underline -- causes the subsequent text to be underlined.
Center -- causes the subsequent text to be centered.
FlushLeft -- causes the subsequent text to be left justified.
FlushRight -- causes the subsequent text to be right justified.
Indent -- causes the subsequent text to be indented at the left margin.
IndentRight -- causes the subsequent text to be indented at the right margin.
Outdent -- causes the subsequent text to be outdented at the left margin.
OutdentRight -- causes the subsequent text to be outdented at the right margin.
SamePage -- causes the subsequent text to be grouped, if possible, on one page.
Subscript -- causes the subsequent text to be interpreted as a subscript.
Superscript -- causes the subsequent text to be interpreted as a superscript.
Heading -- causes the subsequent text to be interpreted as a page heading.
Footing -- causes the subsequent text to be interpreted as a page footing.
ISO-8859-X (for any value of X that is legal as a "charset" parameter) -- causes the subsequent text to be interpreted as text in the appropriate character set.
US-ASCII -- causes the subsequent text to be interpreted as text in the US-ASCII character set.
Excerpt -- causes the subsequent text to be interpreted as a textual excerpt from another source. Typically this will be displayed using indentation and an alternate font, but such decisions are up to the viewer.
Paragraph -- causes the subsequent text to be interpreted as a single paragraph, with appropriate paragraph breaks (typically blank space) before and after.
Signature -- causes the subsequent text to be interpreted as a "signature". Some systems may wish to display signatures in a smaller font or otherwise set them apart from the main text of the message.
Comment -- causes the subsequent text to be interpreted as a comment, and hence not shown to the reader.
No-op -- has no effect on the subsequent text.
lt -- <lt> is replaced by a literal "<" character. No balancing </lt> is allowed.
nl -- <nl> causes a line break. No balancing </nl> is allowed.
np -- <np> causes a page break. No balancing </np> is allowed.

Each positive formatting command affects all subsequent text until the matching negative formatting command. Such pairs of formatting commands must be properly balanced and nested. Thus, a proper way to describe text in ***bold italics*** is:

```
<bold><italic>the-text</italic></bold>
```

or, alternately,

```
<italic><bold>the-text</bold></italic>
```

but, in particular, the following is illegal richtext:

```
<bold><italic>the-text</bold></italic>
```

NOTE: The nesting requirement for formatting commands imposes a slightly higher burden upon the composers of richtext bodies, but potentially simplifies richtext displayers by allowing them to be stack-based. The main goal of richtext is to be simple enough to make multifont, formatted email widely readable, so that those with the capability of sending it will be able to do so with confidence. Thus slightly increased complexity in the composing software was deemed a reasonable tradeoff for simplified reading software. Nonetheless, implementors of richtext readers are encouraged to follow the general Internet guidelines of being conservative in what you send and liberal in what you accept. Those implementations that can do so are encouraged to deal reasonably with improperly nested richtext.

Implementations must regard any unrecognized formatting command as equivalent to "No-op", thus facilitating future extensions to "richtext". Private extensions may be defined using formatting commands that begin with "X-", by analogy to Internet mail header field names.

It is worth noting that no special behavior is required for the TAB (HT) character. It is recommended, however, that, at least when fixed-width fonts are in use, the common semantics of the TAB (HT) character should be observed, namely that it moves to the next column position that is a multiple of 8. (In other words, if a TAB (HT) occurs in column n, where the leftmost column is column 0, then that TAB (HT) should be replaced by 8-(*n* mod 8) SPACE characters.)

Richtext also differentiates between "hard" and "soft" line breaks. A line break (CRLF) in the richtext data stream is interpreted as a "soft" line break, one that is included only for purposes of mail transport, and is to be treated as white space by richtext interpreters. To include a "hard" line break (one that must be displayed as such), the "<nl>" or "<paragraph> formatting constructs should be used. In general, a soft line break should be treated as white space, but when soft line breaks immediately follow a <nl> or a </paragraph> tag they should be ignored rather than treated as white space.

Putting all this together, the following "text/richtext" body fragment:

```
<bold>Now</bold> is the time for <italic>all</italic>
good men
 <smaller>(and <lt>women>)</smaller> to
<ignoreme></ignoreme> come

to the aid of their
<nl>
beloved <nl><nl>country. <comment> Stupid quote!
</comment> -- the end
```

represents the following formatted text (which will, no doubt, look cryptic in the text-only version of this document):

Now is the time for *all* good men (and <women>) to come to the aid of their beloved

country. -- the end

Richtext conformance: A minimal richtext implementation is one that simply converts "<lt>" to "<", converts CRLFs to SPACE, converts <nl> to a newline according to local newline convention, removes everything between a <comment> command and the next balancing </comment> command, and removes all other formatting commands (all text enclosed in angle brackets).

NOTE ON THE RELATIONSHIP OF RICHTEXT TO SGML: Richtext is decidedly not SGML, and must not be used to transport arbitrary SGML documents. Those who wish to use SGML document types as a mail transport format must define a new text or application subtype, e.g., "text/sgml-dtd-whatever" or "application/sgml-dtd-whatever", depending on the perceived readability of the DTD in use. Richtext is designed to be compatible with SGML, and specifically so that it will be possible to define a richtext DTD if one is needed. However, this does not imply that arbitrary SGML can be called richtext, nor that richtext implementors have any need to understand SGML; the description in this document is a complete definition of richtext, which is far simpler than complete SGML.

NOTE ON THE INTENDED USE OF RICHTEXT: It is recognized that implementors of future mail systems will want rich text functionality far beyond that currently defined for richtext. The intent of richtext is to provide a common format for expressing that functionality in a form in which much of it, at least, will be understood by interoperating software. Thus, in particular, software with a richer notion of formatted text than richtext can still use richtext as its basic representation, but can extend it with new formatting commands and by hiding information specific to that software system in richtext comments. As such systems evolve, it is expected that the definition of richtext will be further refined by future published specifications, but richtext as defined here provides a platform on which evolutionary refinements can be based.

IMPLEMENTATION NOTE: In some environments, it might be impossible to combine certain richtext formatting commands, whereas in others they might be combined easily. For example, the combination of <bold> and <italic> might produce ***bold italics*** on systems that support such fonts, but there exist systems that can make text bold or italicized, but not both. In such cases, the most recently issued recognized formatting command should be preferred.

One of the major goals in the design of richtext was to make it so simple that even text-only mailers will implement richtext-to-plain-text translators, thus increasing the likelihood that multifont text will become "safe" to use very widely. To demonstrate this simplicity, an extremely simple 35-line C program that converts richtext input into plain text output is included in Appendix D.

7.2 The Multipart Content-Type

In the case of multiple part messages, 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 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 field implies that the encapsulation is plain US-ASCII text. 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 should 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.

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 an image. In order to represent the latter, the body part must have "Content-Type: message", and its body (after the blank line) must be the encapsulated message, with its own "Content-Type: image" 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 the 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 7-bit ASCII in any case, 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.

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 body part headers embedded in the bodies of messages of type "multipart."

7.2.1 Multipart: The common syntax

All subtypes of "multipart" share a common syntax, defined in this section. 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 code 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.

Thus, a typical multipart Content-Type header field might look like this:

```
Content-Type: multipart/mixed;
      boundary=gc0p4Jq0M2Yt08jU534c0p
```

This indicates that the entity consists of several parts, each itself with a structure that is syntactically identical to an RFC 822 message, except that the header area might be completely empty, and that the parts are each preceded by the line

```
--gc0p4Jq0M2Yt08jU534c0p
```

Note that the encapsulation boundary must occur at the beginning of a line, i.e., following a CRLF, and that that initial CRLF is considered to be part of 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 considered part of 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, should 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.

The requirement that the encapsulation boundary begins with a CRLF implies that the body of a multipart entity must itself begin with a CRLF before the first encapsulation line -- that is, if the "preamble" area is not used, the entity headers must be followed by TWO CRLFs. This is indeed how such entities should be composed. A tolerant mail reading program, however, may interpret a body of type multipart that begins with an encapsulation line NOT initiated by a CRLF as also being an encapsulation boundary, but a compliant mail sending program must not generate such entities.

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:

```
--gc0p4Jq0M2Yt08jU534c0p--
```

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 should ignore anything that appears before the first boundary or after the last one.

NOTE: These "preamble" and "epilogue" areas are 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.

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>
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 compliant readers.
--simple boundary

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

This is explicitly typed plain 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 must use 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 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 should 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:

```
multipart-body := preamble 1*encapsulation
                  close-delimiter epilogue

encapsulation := delimiter CRLF body-part

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

```
                                  ; when content-type is multipart
                                  ; There must be no space
                                  ; between "--" and boundary.

close-delimiter := delimiter "--" ; Again, no space before "--"

preamble := *text                 ; to be ignored upon receipt.

epilogue := *text                 ; to be ignored upon receipt.

body-part = <"message" as defined in RFC 822,
          with all header fields optional, and with the
          specified delimiter not occurring anywhere in
          the message body, either on a line by itself
          or as a substring anywhere.  Note that the
          semantics of a part differ from the semantics
          of a message, as described in the text.>
```

NOTE: Conspicuously missing from the multipart type is a notion of *structure*d, 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.

7.2.2 The Multipart/mixed (primary) subtype

The primary subtype for multipart, "mixed", is intended for use when the body parts are independent and intended to be displayed serially. Any multipart subtypes that an implementation does not recognize should be treated as being of subtype "mixed".

7.2.3 The Multipart/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. User agents should recognize that the content of the various parts are interchangeable. The user agent should either choose the "best" type based on the user’s environment and preferences, or offer the user the available alternatives. In general, choosing the best type means displaying only the LAST part that can be displayed. This 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>
```

```
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/richtext

.... richtext version of same message goes here ...
--boundary42
Content-Type: text/x-whatever

.... fanciest formatted version of same message goes here ...
--boundary42--
```

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

In general, user agents that compose multipart/alternative entities should 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 mutlipart/alternative entities are viewed using a non-MIME-compliant mail reader. While this approach does impose some burden on compliant 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 explicitly be given the choice.

7.2.4 The Multipart/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
MIME-Version: 1.0
Subject:  Internet Digest, volume 42
Content-Type: multipart/digest;
      boundary="---- next message ----"


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

From: someone-else
Subject: my opinion

...body goes here ...

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

From: someone-else-again
Subject: my different opinion

... another body goes here...

------ next message ------
```

7.2.5 The Multipart/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, all of the parts are intended to be presented in parallel, i.e., simultaneously, on hardware and software that are capable of doing so. Composing agents should be aware that many mail readers will lack this capability and will show the parts serially in any event.

7.3 The Message Content-Type

It is frequently desirable, in sending mail, to encapsulate another mail message. For this common operation, a special Content-Type, "message", is defined. The primary subtype, message/rfc822, has no required parameters in the Content-Type field. Additional subtypes, "partial" and "External-body", do have required parameters. These subtypes are explained below.

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.

As stated in the definition of the Content-Transfer-Encoding field, no encoding other than "7bit", "8bit", or "binary" is permitted for messages or parts of type "message". 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-ASCII text in the headers of an encapsulated message can be specified using the mechanisms described in [RFC-1342].

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."

7.3.1 The Message/rfc822 (primary) subtype

A Content-Type of "message/rfc822" indicates that the body contains an encapsulated message, with the syntax of an RFC 822 message.

7.3.2 The Message/Partial subtype

A subtype of message, "partial", is defined in order to allow large objects to be delivered as several separate pieces of mail and automatically reassembled by the receiving user agent. (The concept is similar to IP fragmentation/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 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 RFC 822 format message, which may have its own Content-Type header field, and thus may contain any other data type.

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 headers from the initial enclosing entity (part one), except those that start with "Content-" and "Message-ID", must be copied, in order, to the new message.

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

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

For 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
Subject: Audio mail
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
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
Subject: Audio mail
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
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...
```

It should be noted that, because some message transfer agents may choose to automatically fragment large messages, and because such agents may use 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.

It should also be noted that 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.

7.3.3 The Message/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 message body or body part 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

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 auxilliary information for some such messages, as indeed it is when the access-type is "mail-server". Of the access-types defined by this document, the phantom body is used only when the access-type is "mail-server". In all other cases, the phantom body is ignored.

The only always-mandatory parameter for message/external-body is "access-type"; all of the other parameters may be mandatory or optional depending on the value of access-type.

> **ACCESS-TYPE** -- One or more case-insensitive words, comma-separated, indicating supported access mechanisms by which the file or data may be obtained. Values include, but are not limited to, "FTP", "ANON-FTP", "TFTP", "AFS", "LOCAL-FILE", and "MAIL-SERVER". Future values, except for experimental values beginning with "X-", must

be registered with IANA, as described in Appendix F .

In addition, the following two parameters are optional for ALL access-types:

EXPIRATION -- The date (in the RFC 822 "date-time" syntax, as extended by RFC 1123 to permit 4 digits in the date field) after which the existence of the external data is not guaranteed.

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.

PERMISSION -- A 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.

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

7.3.3.1 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:

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

SITE -- A machine from which the file may be obtained, using the given protocol

Before the data is retrieved, using these protocols, the user will generally need to be asked to provide a login id and a password for the machine named by the site parameter.

In addition, the following optional parameters may also appear when the access-type is FTP or ANON-FTP:

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

MODE -- A transfer mode for retrieving the information, e.g. "image".

7.3.3.2 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.

7.3.3.3 The "local-file" and "afs" access-types

An access-type of "local-file" indicates that the actual body is accessible as a file on the local machine. An access-type of "afs" indicates that the file is accessible via the global AFS file system. In both cases, only a single parameter is required:

> **NAME** -- The name of the file that contains the actual body data.

The following optional parameter may be 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:

> **SITE** -- A domain specifier for a machine or set of machines that are known to have access to the data file. 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.

7.3.3.4 The "mail-server" access-type

The "mail-server" access-type indicates that the actual body is available from a mail server. The mandatory parameter for this access-type is:

> **SERVER** -- The email address of the mail server from which the actual body data can be obtained.

Because mail servers accept a variety of syntax, 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 may be 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 should include the phantom body in the body of the message it sends to the mail server address to retrieve the relevant data.

7.3.3.5 Examples and Further Explanations

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.

If an entity is of type "message/external-body", then the body of the entity will contain the header fields of the encapsulated message. The body itself is to be found in the external location. This means that if the body of the "message/external-body" message contains two consecutive CRLFs, everything after those pairs is NOT part of the message itself. For most message/external-body messages, this trailing area must simply be ignored. However, it is a convenient place for additional data that cannot be included in the content-type header field. In particular, if the "access-type" value is "mail-server", then the trailing area must contain commands to be sent to the mail server at the address given by NAME@SITE, where NAME and SITE are the values of the NAME and SITE parameters, respectively.

The embedded message header fields which appear in the body of the message/external-body data can be used to declare the Content-type of the external body. Thus a complete message/external-body message, referring to a document in PostScript format, might look like this:

```
From: Whomever
Subject: whatever
MIME-Version: 1.0
Message-ID: id1@host.com
Content-Type: multipart/alternative; boundary=42


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

```
Content-type: application/postscript

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

Content-type: application/postscript

--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

get rfc-xxxx doc

--42--
```

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.

7.4 The Application Content-Type

The "application" Content-Type is to be used for 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 should 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 three subtypes: octet-stream, ODA, and PostScript.

In general, 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. Such usages must be registered with IANA, as described in Appendix F.

7.4.1 The Application/Octet-Stream (primary) subtype

The primary subtype of application, "octet-stream", may be used to indicate that a body contains binary data. The set of possible parameters includes, but is not limited to:

NAME -- a suggested name for the binary data if stored as a file.

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

CONVERSIONS -- the set of operations that have been performed on the data before putting it in the mail (and before any Content-Transfer-Encoding that might have been applied). If multiple conversions have occurred, they must be separated by commas and specified in the order they were applied -- that is, the leftmost conversion must have occurred first, and conversions are undone from right to left. Note that NO conversion values are defined by this document. Any conversion values that that do not begin with "X-" must be preceded by a published specification and by registration with IANA, as described in Appendix F.

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

The values for these attributes are left undefined at present, but may require specification in the future. An example of a common (though UNIX-specific) usage might be:

```
Content-Type:  application/octet-stream;
     name=foo.tar.Z; type=tar;
     conversions="x-encrypt,x-compress"
```

However, it should be noted that the use of such conversions is *explicitly discouraged* due to a lack of portability and standardization. The use of uuencode is particularly discouraged, in favor of the Content-Transfer-Encoding mechanism, which is both more standardized and more portable across mail boundaries.

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.

7.4.2 The Application/PostScript subtype

A Content-Type of "application/postscript" indicates a PostScript program. The language is defined in [POSTSCRIPT]. It is recommended that Postscript as sent through email should use Postscript document structuring conventions if at all possible, and correctly.

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, 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.

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 reenabling full-function versions of these operators.

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. The ability to exit will probably be unavailable in secure PostScript implementations. Message-receiving and -displaying software should, if possible, disable the ability to make retained changes to the PostScript environment. Eliminate the startjob and exitserver commands. If these commands cannot be eliminated, at least set the password associated with them to a hard-to-guess value.

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, if possible, disable the ability to change system and device parameters. If these operators cannot be disabled, at least set the password associated with them to a hard-to-guess value.

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.

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.

It is possible to write PostScript that consumes huge amounts of various system resources. It is also possible to write PostScript programs that loop infinitely. 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.

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.

7.4.3 The Application/ODA subtype

The "ODA" subtype of application is used to indicate that a body contains information encoded according to the Office Document Architecture [ODA] standards, using the ODIF representation format. For application/oda, the Content-Type line should also specify an attribute/value pair that indicates the document application profile (DAP), using the key word "profile". Thus an appropriate header field might look like this:

```
Content-Type:  application/oda; profile=Q112
```

Consult the ODA standard [ODA] for further information.

7.5 The Image Content-Type

A Content-Type of "image" indicates that the bodycontains an image. The subtype names the specific image format. These names are case insensitive. 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 Appendix F.

7.6 The 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 audio encoded using 8-bit ISDN u-law [PCM]. When this subtype is present, a sample rate of 8000 Hz and a single channel is assumed.

7.7 The 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".

7.8 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.

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 text/richtext, message/partial, and message/external-body.

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
Randall Atkinson
Philippe Brandon
Kevin Carosso
Uhhyung Choi
Cristian Constantinof
Mark Crispin
Dave Crocker
Terry Crowley
Walt Daniels
Frank Dawson
Hitoshi Doi
Kevin Donnelly
Keith Edwards
Chris Eich
Johnny Eriksson
Craig Everhart
Patrik Fältström
Erik E. Fair
Roger Fajman
Alain Fontaine
James M. Galvin
Philip Gladstone
Thomas Gordon
Phill Gross
James Hamilton
Steve Hardcastle-Kille
David Herron
Bruce Howard
Bill Janssen
Olle Järnefors
Risto Kankkunen
Phil Karn
Alan Katz
Tim Kehres
Neil Katin
Kyuho Kim
Anders Klemets
John Klensin
Valdis Kletniek
Jim Knowles
Stev Knowles
Bob Kummerfeld
Pekka Kytolaakso
Stellan Lagerström
Vincent Lau
Donald Lindsay
Timo Lehtinen
John R. MacMillan
Rick McGowan
Leo Mclaughlin
Goli Montaser-Kohsari
Keith Moore
Tom Moore
Erik Naggum
Mark Needleman
John Noerenberg
Mats Ohrman
Julian Onions
Michael Patton
David J. Pepper
Blake C. Ramsdell
Luc Rooijakkers
Marshall T. Rose
Jonathan Rosenberg
Jan Rynning
Harri Salminen
Michael Sanderson
Masahiro Sekiguchi
Mark Sherman
Keld Simonsen
Bob Smart
Peter Speck
Henry Spencer
Einar Stefferud
Michael Stein
Klaus Steinberger
Peter Svanberg
James Thompson
Steve Uhler
Stuart Vance
Erik van der Poel
Guido van Rossum
Peter Vanderbilt
Greg Vaudreuil
Ed Vielmetti
Ryan Waldron
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 -- Minimal 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. Encode any data sent that is not in seven-bit mail-ready representation using one of these transformations and include the appropriate Content-Transfer-Encoding header field, unless the underlying transport mechanism supports non-seven-bit data, as SMTP does not.

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, show or offer to show the user the "raw" version of the data. An ability at least to convert "text/richtext" to plain text, as shown in Appendix D, is encouraged, but not required for conformance.

Message:

--Recognize and display at least the primary (822) encapsulation.

Multipart:

-- Recognize the primary (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.

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

Application:

-- Offer the ability to remove either of the two types of Content-Transfer-Encoding defined in this document and put the resulting information in a user file.

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. Unrecognized predefined types, which in a MIME-conformant mailer might still include audio, image, or video, should also be treated in this way.

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 -- General 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 body parts. As such, the persistence of CRLF as something other than a line break should 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 should 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 ASCII character set. Such conversion is STRONGLY DISCOURAGED, but it may occur, and mail formats should 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 should 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, should not be relied on.

(6) Many mail domains use variations on the 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 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:

"’" (ASCII code 39)
"(" (ASCII code 40)
")" (ASCII code 41)
"+" (ASCII code 43)
"," (ASCII code 44)
"-" (ASCII code 45)
"." (ASCII code 46)
"/" (ASCII code 47)
":" (ASCII code 58)
"=" (ASCII code 61)
"?" (ASCII code 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.

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 illegal practices are known to occur on established networks, and implementions 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 richtext 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>
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 preceding blank line means
no header fields were given and this is text,
with charset US ASCII.  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
    u-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/richtext

This is <bold><italic>richtext.</italic></bold>
<nl><nl>Isn’t it
<bigger><bigger>cool?</bigger></bigger>

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

From: *(name 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 -- A Simple Richtext-to-Text Translator in C

One of the major goals in the design of the richtext subtype of the text Content-Type is to make formatted text so simple that even text-only mailers will implement richtext-to-plain-text translators, thus increasing the likelihood that multifont text will become "safe" to use very widely. To demonstrate this simplicity, what follows is an extremely simple 44-line C program that converts richtext input into plain text output:

```
#include <stdio.h>
#include <ctype.h>
main() {
    int c, i;
    char token[50];

    while((c = getc(stdin)) != EOF) {
        if (c == '<') {
            for (i=0; (i<49 && (c = getc(stdin)) != '>'
                           && c != EOF); ++i) {
                token[i] = isupper(c) ? tolower(c) : c;
            }
            if (c == EOF) break;
            if (c != '>') while ((c = getc(stdin)) != '>'
                           && c != EOF) {;}
            if (c == EOF) break;
            token[i] = '\0';
            if (!strcmp(token, "lt")) {
                putc('<', stdout);
            } else if (!strcmp(token, "nl")) {
                putc('\n', stdout);
            } else if (!strcmp(token, "/paragraph")) {
                fputs("\n\n", stdout);
            } else if (!strcmp(token, "comment")) {
                int commct=1;
                while (commct > 0) {
                    while ((c = getc(stdin)) != '<'
                           && c != EOF) ;
                    if (c == EOF) break;
                    for (i=0; (c = getc(stdin)) != '>'
                             && c != EOF; ++i) {
                        token[i] = isupper(c) ?
                                   tolower(c) : c;
                    }
                    if (c== EOF) break;
                    token[i] = NULL;
                    if (!strcmp(token, "/comment")) --commct;
                    if (!strcmp(token, "comment")) ++commct;
                }
            } /* Ignore all other tokens */
        } else if (c != '\n') putc(c, stdout);
    }
    putc('\n', stdout); /* for good measure */
}
```

It should be noted that one can do considerably better than this in displaying richtext data on a dumb terminal. In particular, one can replace font information such as "bold" with textual emphasis (like *this* or _T_H_I_S_). One can also properly handle the richtext formatting commands regarding indentation, justification, and others. However, the above program is all that is *necessary* in order to present richtext on a dumb terminal.

Appendix E -- 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

body-part = <"message" as defined in RFC 822,
          with all header fields optional, and with the
          specified delimiter not occurring anywhere in
          the message body, either on a line by itself
          or as a substring anywhere.>

boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

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

close-delimiter := delimiter "--"

Content-Description := *text

Content-ID := msg-id

Content-Transfer-Encoding :=    "BASE64"  / "QUOTED-PRINTABLE" /
                                "8BIT"    / "7BIT" /
                                "BINARY"  / x-token

Content-Type := type "/" subtype *[";" parameter]

delimiter := CRLF "--" boundary   ; taken from Content-Type field.
                                  ; when content-type is multipart
                                  ; There should be no space
                                  ; between "--" and boundary.

encapsulation := delimiter CRLF body-part

epilogue := *text                 ; to be ignored upon receipt.

MIME-Version := 1*text

multipart-body := preamble 1*encapsulation close-delimiter epilogue
```

```
parameter := attribute "=" value

preamble := *text                 ; to be ignored upon receipt.

subtype := token

token := 1*<any CHAR except SPACE, CTLs, or tspecials>

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


type :=        "application"   / "audio"    ; case-insensitive
            / "image"         / "message"
            / "multipart"     / "text"
            / "video"         / x-token

value := token / quoted-string

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

Appendix F -- IANA Registration Procedures

MIME has been carefully designed to have extensible mechanisms, and it is expected that the set of content-type/subtype pairs and their associated parameters will grow significantly with time. Several other MIME fields, notably character set names, access-type parameters for the message/external-body type, conversions parameters for the application type, and possibly even Content-Transfer-Encoding values, 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 defines a registration process which uses the Internet Assigned Numbers Authority (IANA) as a central registry for such values.

In general, parameters in the content-type header field are used to convey supplemental information for various content types, and their use is defined when the content-type and subtype are defined. New parameters should not be defined as a way to introduce new functionality.

In order to simplify and standardize the registration process, this appendix gives templates for the registration of new values with IANA. Each of these is given in the form of an email message template, to be filled in by the registering party.

F.1 Registration of New Content-type/subtype Values

Note that MIME is generally expected to be extended by subtypes. If a new fundamental top-level type is needed, its specification should be published as an RFC or submitted in a form suitable to become an RFC, and be subject to the Internet standards process.

```
To:  IANA@isi.edu
Subject:  Registration of new MIME content-
type/subtype

MIME type name:

(If the above is not an existing top-level MIME type,
please explain why an existing type cannot be used.)

MIME subtype name:

Required parameters:

Optional parameters:

Encoding considerations:
```

Security considerations:

Published specification:

(The published specification must be an Internet RFC or RFC-to-be if a new top-level type is being defined, and must be a publicly available specification in any case.)

Person & email address to contact for further information:

F.2 Registration of New Character Set Values

To: IANA@isi.edu
Subject: Registration of new MIME character set value

MIME character set name:

Published specification:

(The published specification must be an Internet RFC or RFC-to-be or an international standard.)

Person & email address to contact for further information:

F.3 Registration of New Access-type Values for Message/external-body

To: IANA@isi.edu
Subject: Registration of new MIME Access-type for
Message/external-body content-type

MIME access-type name:

Required parameters:

Optional parameters:

Published specification:

(The published specification must be an Internet RFC or RFC-to-be.)

Person & email address to contact for further information:

F.4 Registration of New Conversions Values for Application

```
To:  IANA@isi.edu
Subject:  Registration of new MIME Conversions value
for Application content-type

MIME Conversions name:

Published specification:
```

(The published specification must be an Internet RFC or RFC-to-be.)

```
Person & email address to contact for further
information:
```

Appendix G -- Summary of the Seven Content-types

Content-type: **text**

Subtypes defined by this document: plain, richtext

Important Parameters: charset

Encoding notes: quoted-printable generally preferred if an encoding is needed and the character set is mostly an 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: **multipart**

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

Important Parameters: boundary

Encoding notes: No content-transfer-encoding is permitted.

Content-type: **message**

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

Important Parameters: id, number, total

Encoding notes: No content-transfer-encoding is permitted.

Content-type: **application**

Subtypes defined by this document: octet-stream, postscript, oda

Important Parameters: profile

Encoding notes: base64 generally preferred for octet-stream or other unreadable subtypes.

Security considerations: This type is intended for the transmission of data to be interpreted by locally-installed programs. If used, for example, to transmit executable binary programs or programs in general-purpose interpreted languages, such as LISP programs or shell scripts, severe security problems could result. In general, 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: **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

Appendix H -- 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 message part can be modelled as being done in a number of steps. Note that these steps are roughly similar to those steps used in RFC1113:

Step 1. Creation of local form.

The body part 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 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.

Step 2. Conversion to canonical form.

The entire body part, 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 part 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.

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 RFC822. Note that the restriction on line lengths implied by RFC822 is eliminated if the next step employs either quoted-printable or base64 encoding.

Step 3. Apply transfer encoding.

A Content-Transfer-Encoding appropriate for this body part 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 body part.

Step 4. Insertion into message.

The encoded object is inserted into a MIME message with appropriate body part headers and boundary markers.

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 bodyparts 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 compliant with the formats specified by RFC822. 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 RFC822 CRLF delimiters.

Other implementation variations are conceivable as well. The only important aspect of this discussion is that the resulting messages are consistent with those produced by the model described here.

References

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

[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] 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.

[ODA] ISO 8613; Information Processing: Text and Office System; Office Document Architecture (ODA) and Interchange Format (ODIF), Part 1-8, 1989.

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

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

[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.

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

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

[RFC-822] Crocker, D. Standard for the format of ARPA Internet text messages. August, 1982, UDEL, RFC-822.

[RFC-934] Rose, M.T.; Stefferud, E.A. Proposed standard for message encapsulation. January, 1985, Delaware and NMA, RFC-934.

[RFC-959] Postel, J.B.; Reynolds, J.K. File Transfer Protocol. October, 1985, USC/Information Sciences Institute, RFC-959.

[RFC-1049] Sirbu, M.A. Content-Type header field for Internet messages. March, 1988, CMU, RFC-1049.

[RFC-1113] Linn, J. Privacy enhancement for Internet electronic mail: Part I - message encipherment and authentication procedures. August, 1989, IAB Privacy Task Force, RFC-1113.

[RFC-1154] Robinson, D.; Ullmann, R. Encoding header field for Internet messages. April, 1990, Prime Computer, Inc., RFC-1154.

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

Security Considerations

Security issues are discussed in Section 7.4.2 and in Appendix G. 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 applicaton/postscript content-type in Section 7.4.2 may serve as a model for considering other content-types with remote execution capabilities.

Authors' Addresses

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

Nathaniel S. Borenstein
MRE 2D-296, Bellcore
445 South St.
Morristown, NJ 07962-1910

Phone: +1 201 829 4270
Fax: +1 201 829 7019
Email: nsb@bellcore.com

Ned Freed
Innosoft International, Inc.
250 West First Street
Suite 240
Claremont, CA 91711

Phone: +1 714 624 7907
Fax: +1 714 621 5319
Email: ned@innosoft.com

THIS PAGE INTENTIONALLY LEFT BLANK.

Please discard this page and place the following table of contents after the title page.

Table of Contents