INTERNET DRAFT

Mechanisms for Specifying and Describing the Format of Internet Message Bodies

Nathaniel Borenstein, Bellcore
Ned Freed, Innosoft

June 1991

Status of This Memo

This draft document will be submitted to the RFC editor as a protocol specification. Distribution of this memo is unlimited. Please send comments to Nathaniel Borenstein <nsb@thumper.bellcore.com>.

Experimentation with the mechanisms described in this memo is encouraged. It is anticipated that such experimentation will take place during the summer of 1991, after which a new draft will be submitted to the RFC editor. Comments that are intended to affect that future draft should be received no later than September 23, 1991.

Abstract

This document suggests extensions to the RFC 822 message representation protocol to allow multi-part textual and non-textual messages 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. In particular, it is designed to permit and standardize Internet mail mechanisms for representing text in character sets other than US-ASCII, for including formatted multi-font text messages, for including non-textual material such as images and audio fragments, and for generally extending Internet mail to include new types of objects that are tagged in such a way that cooperating mail agents can recognize their types.

Contents

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 of SMTP transport, as defined by RFC 821 [RFC-821]. As the format has seen wider use, a number of limitations have become increasingly problematic 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 email users whose languages require the use of character sets richer than US ASCII [REF-ANSI]. For mail containing audio, video, Asian language text, or even text in most European languages, RFC 822 does not specify enough to permit interoperability.

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 a user to convert any non-textual data that she may wish to send into seven-bit bytes representable as printable ASCII characters before invoking her local mail UA (User Agent program). 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, and many others.

These limitations 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 [REF-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 MHS, where the non-textual information would definitely become useful again.

This memo describes several mechanisms that combine to solve these problems. In particular, it describes:

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

2. A Content-Transfer-Encoding header field, which can be used to describe an auxilliary encoding that was applied to the data in order to allow it to pass through the mail transport layer.

3. A "text" content-type value, which can be used to represent text information in a number of character sets in a standardized manner.

4. A "multipart" content-type value, which can be used to combine several separate body-parts, which may be made of different types of data, into a single message.

5. A "binary" content-type value, which can be used to transmit uninterpreted or partially-interpreted binary data, and hence to implement an email file transfer service.

6. A "message" content-type value, for encapsulating a mail message.

7. Several additional content-type values and subtypes, which can be used by consenting User Agents to interoperate with additional message types such as audio, images, and more.

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

Finally, to specify and promote a minimal level of interoperability, this memo describes a subset of the above mechanisms that defines "conformance" with this memo. That is, it specifies the minimal subset required for an implementation to be called "XXXX-conformant."

2 The Content-Type Header Field

The Content-Type header field was first defined in RFC 1049. This section extends and supersedes that definition. RFC 1049 content-types are all conformant with the new, more general syntax. (In particular, RFC 1049 content-types omitted the subtype/character-set specification, and always had at most two of the parts now called "parameters", which were distinguished by their position as indicating a version number and a resource reference.)

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

The Content-Type header field is used to specify the type of data in a message, by giving a type name, and to provide auxiliary information that may be required for certain types. In addition, a distinguished syntax is defined for specifying subtype information, including character set information in the case of text. After the type name and the optional subtype, the remainder of the header field is simply a set of parameter specifications, as defined for each named type, and an optional comment.

In the Extended BNF notation of RFC-822, we define a Content-type header field value as follows:

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

type := "TEXT" / "TEXT-PLUS" / "MESSAGE" / "AUDIO" / "IMAGE" / "VIDEO" /
        "BINARY" / "APPLICATION"/ "MULTIPART" / "X-" token

subtype := token

parameter := token / quoted-string

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

tspecial := "(" / ")" "<" / ">" / "@" / ","  / ":" / "/" /
            "\" / <"> / "[" / "]" / ";"
```

The type and subtype values are not case sensitive. TEXT, Text, and TeXt are all equivalent.

An initial set of nine content-types is defined by this memo. 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 usually be accomplished by the creation of new subtypes of these initial types. In the future, more top-level types should be defined by an extension to this standard.

The only constraint on the definition of subtype names is the desire that their uses not conflict. That is, it would be undesirable to have two different communities using "Content-type: binary/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 approval or standardization

2. "Standard" values may be defined by the publication of an Internet RFC, or by registering them with the Internet Assigned Numbers Authority (IANA) at ISI, by email to IANA@ISI.EDU.

The nine standard initial predefined content-types are detailed in the appendices of this memo. They are:

text -- textual information, with character set given by the subtype
text-plus -- mostly textual information, with embedded formatting commands. A simple default type is defined, with possible subtypes including troff, TeX, and so on.
message -- an encapsulated message, with initial subtypes for partial messages and privacy-enhanced messages
multipart -- a message consisting of multiple parts of independent type values, with initial subtype digest.
audio -- a message containing audio data, with initial subtypes a-law and u-law.
image -- a message containing image data, with initial subtypes G3fax, gif, pbm, ppm, and pgm.
video -- a message containing video data.
binary -- a message containing some other form of binary data.
application -- a message containing data to be processed by a mail-based application.

If no Content-type header field is present, "text" is generally to be assumed, with the default (US-ASCII) subtype as specified later in this memo. This is consistent with the default message body type as defined by RFC 822. However, this does not mean that a specification of "Content-type: text/us-ascii" is optional. In the absence of such a header field, it is impossible to be certain that a message is actually text in the US-ASCII character set, because it might well be a message that, using the conventions that predate this memo, includes non-textual data in a manner that cannot be automatically recognized (e.g. a uuencoded compressed UNIX tar file). Although there is no acceptable alternative to treating such untyped messages as "text/us-ascii", implementors should remain aware that unless explicitly so marked, they may in practice be 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. We have simply attempted, in this memo, to give as many standard Content-type definitions as was possible given the current state of our knowledge.

3 The Content-Transfer-Encoding Header Field

Many content-types which are desired to transport via e-mail are represented, in their "natural" format, as 8-bit character or binary data. Such data can not be properly transmitted over existing Internet mail mechanisms because both RFC 821 and RFC 822 restrict mail messages to 7-bit US-ASCII data with 1000 character lines.

It is necessary, therefore, to extend the definition of the data types allowed in the RFC 821 and RFC 822 framework, and to define a standard mechanism for encoding such data in an acceptable manner.

This memo 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 to represent the message body in an acceptable manner.

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, 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. Second, certain Content-Types may require different types of transfer encoding under different circumstances. For example, many PostScript messages might consist entirely of short lines of 7-bit data and hence require little or no encoding. Other PostScript messages (especially those using Level 2 PostScript's binary encoding mechanism) may only be resonably 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.

It should be noted, also, that there is considerable interest and effort being expended on extending mail transport to permit 8-bit or binary data. If such extensions ever become commonplace, the Content-Transfer-Encoding mechanism will quickly become irrelevant, and it is therefore desirable not to "overload" Content-Transfer-Encoding with additional mechanisms that might still be useful in such a future. For this reason, Content-Transfer-Encoding is restricted in its scope to refer to nothing but the 7-bit encoding question. Matters such as the basic format in which information is "encoded" are to be handled by other mechanisms.

Unlike Content-types, which are expected to proliferate, it is expected that there will never be more than a few different Content-Transfer-Encoding values, both because there is less need for variation and because the effect of variation in Content-Transfer-Encoding would be more problematic. However, establishing only a single Content-

Transfer-Encoding mechanism does not seem possible. In particular, there is a tradeoff between the desire for a compact and efficient encoding of 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.

A third encoding, for *compressed* ("super-dense") data, is also viewed by many as desirable. This memo does not specify a "compressed" encoding, due largely to the uncertain legal state of the UNIX "compress" command and a lack of certainty, during the drafting of this memo, regarding the right way to define a standard compression algorithm. It is hoped that a compressed Content-Transfer-Encoding will be defined in a future RFC. Any compression algorithm for such a use should be unambiguously defined and without legal encumbrances. (Alternate mechanisms for compression have also been proposed, and might be defined in ways that are compatible with this memo.)

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 as defined by RFC 821 (SMTP). This field has not been defined by any previous RFC. The field’s value is a single atom specifying the type of encoding, as enumerated below. Formally:

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

These values are not case sensitive. That is, Base64 and BASE64 and bAsE64 are all equivalent. An encoding type of 7BIT implies that the message is already in a seven-bit mail-ready representation. This value is assumed if the Content-Transfer-Encoding header field is not present. If the message is stored or transported via a mechanism that permits 8-bit data, a Content-Transfer-Encoding of "8bit" should be used. If the message is stored or transported via a mechanism that permits arbitary binary data, a Content-Transfer-Encoding of "binary" may nonetheless be used. In particular, "8bit" or "binary" must be used in the case where there is a possibility that the message may "leak" into a more restricted (7-bit) transport environment. (DISCUSSION: The distinction between the Content-Transfer-Encoding values of "binary," "8bit," and "7bit" may seem unimportant in an 8-bit binary environment, but clear labeling will be of enormous value to gateways between 8-bit and 7-bit systems. The difference between "8bit" and "binary" is that "8bit" implies adherence to SMTP limits on line length and CR/LF semantics, whereas "binary" does not.)

Implementors may, if necessary, define new Content-Transfer-Encoding values, but should prefix them with "x-" to indicate their 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 discouraged, as it seems likely to hinder interoperability with little potential benefit.

If a Content-Transfer-Encoding header field appears as part of a message header, it applies to the entire message body, whether or not that body is of type "multipart." If it is of type multipart, the encoding applies recursively to all of the encapsulated parts, including their encapsulated headers *and* the encapsulation boundaries. If a Content-Transfer-Encoding header field appears as part of an encapsulation's headers, it applies only to the body of the encapsulated part. If the encapsulated part is itself of type "multipart", the encoding applies recursively to all of the encapsulated parts within that encapsulated part.

It should be noted that, because email is character-oriented, 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 should first be converted to a 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 should be padded with zeroes. This RFC does not provide a mechanism for noting the addition of such padding; this information could either be encoded into the data stream or noted in some additional header field.

The following sections will define the two standard encoding mechanisms.

3.1 Quoted-Printable Content-Transfer-Encoding

The Quoted-Printable encoding is intended to represent data that largely contains octets less than 127. It encodes the data in such a way that the resulting octets are both unlikely to be modified by mail transport, and, when read as ASCII text, are largely recognisable by humans. A message which is entirely ASCII may also be encoded in Quoted-Printable to insure it's survival in an environment which is anticipated to traverse a character translating gateway such as those onto BITNET.

In this encoding, ASCII characters 33 (EXCLAMATION POINT) through 57 (DIGIT 9), inclusive, 59 (SEMICOLON) through 126 (TILDE), inclusive, may be represented as themselves. All other characters, including characters 32 (SPACE), 58 (COLON), 127 (DEL), and all control characters, are to be represented as determined by the following rules:

> Rule #1: Any 8 bit value may be represented by a ":" followed by a two digit hexadecimal representation of the character's 8-bit value. Thus, for example, character 12 (control-L, or formfeed) can be represented by ":0C", and the colon character (58) itself can be represented by ":3A". Rule #1 is optional for characters 10 (control-J, or linefeed), 13 (control-M, or return), and 32 (SPACE) through 126 (TILDE), and is required for all other characters.

> Rule #2: The literal colon character must itself be quoted by a colon (i.e., as "::") if Rule #1 is not used. Note that this is not ambiguous with regard to Rule #1, because ":" is not part of the hexadecimal alphabet.

Rule #3: A colon at the end of a line may be used to indicate a non-significant line break. That is, if one needs to include a long line without line breaks, a message encoded with the quoted-printable encoding should include "soft" line breaks in which the line break is preceded by a colon. Thus if the "raw" form of the line is a single line that says:

```
Now’s the time for all men to come to the aid of their country.
```

This could be represented, in the quoted-printable encoding, as

```
Now’s the time :
for all men 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 quoted-printable encoding REQUIRES that lines be broken so that they are no more than 78 characters long, using soft line breaks when necessary.

Rule #4: SPACE (32) characters may generally be represented as themselves, but should NOT be so represented at the end of a line, because some MTA’s are known to remove "white space" from the end of a line. In such cases, the characters MUST be represented as in rule #1 (as ":20") or as themselves, followed by a soft line break followed by a real line break. Of course, these characters may be so represented within a line as well, if this is desired, though this is less readable. Note that in decoding a quoted-printable message, any trailing white space on a line should be deleted, as it will necessarily have been added by intermediate transport agents.

Rule #5: A CR LF pair normally constitutes a line break and should be represented by a line break in the quoted-printable encoding if that is its meaning. Isolated CRs, LFs, and LF CR sequences must be represented using the :0D, :0A, and :0A:0D notations respectively. CR LF sequences that are not intended to represent a line break should be encoded as :0D:0A to reflect this usage. In other words, the concept "end of line" is represented, in the quoted-printable encoding, by CR LF, although this may be modified in local storage formats. Literal occurrences of CR or LF that do not occur as CRLF or are not intended to represent end-of-line markers must be represented in hexadecimal.

Since the hyphen character ("-") is represented as itself in the Quoted-Printable encoding, the usual care must be taken, when encapsulating a quoted-printable encoded message or body part in a multipart message, to ensure that the encapsulation boundary does not appear anywhere in the message. See the definition of multipart messages later in this memo.

It should be noted that the quoted-printable encoding represents something of a compromise between readability and reliability in transport. Message 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 message 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. For more information about how to ensure that messages are safe against the vagaries of mail gateways, see Appendix I.

3.2 Base64 Content-Transfer-Encoding

The Base64 Content-Transfer-Encoding is designed to represent arbitrary 8 bit data in a form that is not humanly readable. The encoding and decoding algorithms are simple, but the encoded data is 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 two changes: base64 elminates the "*" mechanism for embedded clear text and defines a new syntax for portable end-of-line markers, using the comma character.

A 66-character subset of International Alphabet IA5 is used, enabling 6 bits to be represented per printable character. (The extra 65th and 66th characters "=" and "," are used to signify special processing functions.) This subset has the important property that it is represented identicially in IA5 and ASCII, and all characters in the subset are part of the so-called invariant subset 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 imposed on a binary transport encoding for mail.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right across a 24-bit input group is formed by concatenating 3 8-bit input groups, this is then treated as 4 concatenated 6-bit groups. When encoding a bit stream via the base64 encoding, the bit stream should 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 with 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 RFC (e.g., "-").

Table 1

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	(eol)	,
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 in the encoded version of the data should be ignored by decoding software.

Special processing is performed if fewer than 24 bits are available at the end of a message or encapsulated part of a message. A full encoding quantum is always completed at the end of a message. 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 canonically encoded output 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.

One addition is made to the RFC 1113 specification of this encoding: The comma character (",", ASCII 44) may be used to represent an "end-of-line" or "end-of-record" marker. If line-oriented data are encoded using base64, it is desirable to restore end-of-line markers according to the local convention. The RFC 1113 specification, as given above, offers no way to differentiate between a binary file including a CRLF sequence and a portable end-of-line marker. This memo augments that mechanism to permit such differentiation, as follows. To represent an end-of-line marker:

1. Treat the byte stream preceding the end-of-line as terminating with at the end of the line -- that is, pad with "=" characters as appropriate to complete the representation of the line.

2. Insert a comma character.

3. Resume the encoding starting a new 24-bit input group with the first character on the next line.

Thus, while encoding the sequence "a-b-c-CR-LF-a-b-c" (or, in hexadecimal, "61 62 63 0D 0A 61 62 63") yields the octets which are represented in ASCII as "YWJjDQphYmM=", encoding "a-b-c" followed by an end-of-line followed by "a-b-c" (in hex, "61 62 63" end-of-line "61 62 63") yields "YWJj,YWJj" They will be translated back into the same thing if the local end-of-line convention is ASCII "CRLF" (hexadecimal "0D 0A"), but they will be translated back differently if the end-of-line convention is anything other than ASCII CRLF (hexadecimal "0D 0A"). (Note: The utliity of the portable end-of-line feature is somewhat limited. Most line-oriented data are best represented by the quoted-printable encoding. A few cases, however, might benefit from this mechanism, notably line-oriented textual data in character sets that bear no resemblance to ASCII.)

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

4 Additional Optional Content- Header Fields

4.1 Optional Content-ID Header Field

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

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

4.2 Optional Content-Description Header Field

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

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

5 The Predefined Content-type Values

This memo defines nine initial content-type values and an extension mechanism for private or experimental types. Further types must be defined and published by a new RFC. It is expected that most innovation in new types of mail take place as subtypes of the nine types defined here.

5.1 The TEXT Content-type and the US-ASCII Character Set

The text content-type is intended for sending textual email. It is the default content-type. Subtype names are used, for text, to indicate character sets. The default content-type for internet mail is "text", and the default subtype (character set) is "US-ASCII".

Alternately, a different character set subtype may be specified, in which case the body text is in the specified character set. A recommended list of predefined subtype names can be found at the end of this section. Note that if the specified character set includes 8-bit data, the Content-Transfer-Encoding header field is required in order to transmit the message via 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 elminate 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 is not an arbitrary seven-bit character code, but indicates that the message body uses character coding that uses the exact correspondence of codes to characters specified in ASCII. National use variations of ISO646 [REF-ISO646] are not ASCII, and neither an explicit "ASCII" character set, nor "US-ASCII", nor the default (omission of a character set) should be used when characters are coded using them. (Discussion: RFC821 very explicitly specifies "ASCII", and references an earlier version of the American Standard cited in [REF-ANSI]. Whether that specification, rather than a reference to an International Standard, was done deliberately or out of convenience or ignorance, is no longer interesting: 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 ISO646, or using code-switching procedures (e.g., those of ISO2022), 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 Appendix III. Note that the control characters (0-31) and delete (127) have no defined meaning apart from the combination <CR><LF> (ASCII values 13 and 10) indicating a new line. Two of the characters have de facto meanings in wide use: <FF> (ASCII 12) as the first character of a line means "start this line on the beginning of a new page"; and <TAB> (ASCII 9) means "move the cursor to the next available position 8n+1 after the next postion". Apart from this any use of the control characters or DEL in a message 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 memo."

Beyond US-ASCII, one can imagine an enormous proliferation of character sets. It is the opinion of the authors of this memo 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, there is no clear choice for such a universal representation, and existing practice in several communities seems to point to the continuing 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 consituent base exists. We recommend the use of ISO-10646 wherever possible.

The defined subtypes of text, which name alternate character sets, are:

US-ASCII -- as defined above.

ISO-8859-X -- where "X" is to be replaced, as necessary, for the national use variants of ISO-8859 [REF-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. The ISO-8859 character sets will be rigorously defined, for use in mail and other applications, by a forthcoming RFC.

Note that the character set used should always be explicitly specified in the Content-type field.

The following three subtypes of text are expected to be defined by forthcoming documents. Their use is not recommended in advance of those publications:

ISO-10646 -- as defined in [REF-ISO-10646]. This standard is not, as of this writing, finalized, and therefore its use for email cannot be fully specified.

ISO-2022 -- ISO-2022 -- ISO-2022, as defined in [REF-ISO-2022], is problematic for mail use because it actually specifies ways of designating and accessing character sets, rather than, itself, being a character set. Its use in mail will probably be strongly desired by communities who are already using it locally to handle multiple sets of characters and multi-byte characters. It appears necessary to explicitly specify the ISO-2022 methods that will be permitted in text mail so as to avoid the need for private agreements about, e.g., the specific character sets being used in messages. It is expected that those interested in ISO-2022 mail will devise and publish such a specfication in the future.

QUOTED-READABLE -- A format for representing text in multiple character sets, as defined in [REF-RFC-QR].

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

The intent of "text" is to represent "unformatted" text in an appropriate character set. Formatted text, such as multi-font text, should use the "text-plus" content-type.

5.2 The "Multipart" Content-Type

In the case of multiple part messages, a "multipart" Content-type field should appear in the RFC 822 message header. The message body is then assumed to contain multiple parts separated by encapsulation boundaries. Each of the parts is defined, syntactically, as a header area, a blank line, and a body area, similar to the RFC 822 syntax for a message. However body parts are NOT to be interpreted as actually being RFC 822 messages. To begin with, NO header fields are actually required in body parts. A body part that starts with a blank line, therefore, is a body part for which all default values are to be assumed. In such a case, of course, the absence of a Content-type header field implies that the encapsulation is 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, and may be discarded by gateways. They are permitted to appear in body parts only for ease of conversion between messages and body parts. Of course, "X-" field may be created for experimental or private purposes, with the recognition that the information they contain may be lost at some gateways.

It must be understood that body parts are NOT messages. For example, a gateway between Internet and X.400 mail must be able to tell the difference between a body part that consists of an image and a bodypart that consists of an encapsulated message, the body of which is an image. In order to represent the latter, the body part should have "Content-type: message", and its body (after the blank line) should be the encapsulated message, with its own "Content-type: image" header field. Body parts use the same syntax as messages because there are many legitimate cases in which a body part might be converted into a message, or vice versa. The identical syntax makes such conversions easy, but must be understood by implementors. (For the special case in which all parts are actually messages, a "digest" subtype is also defined.)

As stated previously, each pair of consecutive body parts are separated 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 boundary that will separate the parts.

The Content-type field for multipart messages requires two supplementary fields. The first is used to specify a version number and must be either "1-S" and "1-P". The two versions have identical syntax, but the "-P" is intended as a hint, to receivers, that the parts are intended to be viewed in parallel rather than sequentially. Implementations that can not show the parts in parallel, or that choose not to do so, are free to treat all multipart messages of version "1-P" as if they were version "1-S". However, all implementations must check the version number, to ensure graceful behavior in the event that an incompatible future version of multipart messages is defined later. Future version numbers will always start with an integer for the primary version number, followed by a hyphen and (possibly) some additional text.

The second parameter, which is always required for multipart messages, is used to specify the format of the encapsulation boundary. The encapsulation boundary is defined as a line consisting entirely of two hyphen characters ("-", decimal code 45) followed by the second parameter of the Content-type header field with any leading or trailing white space removed. (DISCUSSION: The specification that white space be removed is intended to eliminate the possible introduction of ambiguity caused by the addition or deletion of white space by message transport agents. 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.)

Thus, a typical multipart content-type header field might look like this:

```
Content-type: multipart; 1-S; gc0p4Jq0M2Yt08jU534c0p
```

This indicates that the message consists of several parts, each itself structured as an RFC 822 message, which are intended to be viewed one-at-a-time, and that the parts are separated by the line

--gc0p4Jq0M2Yt08jU534c0p

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

The encapsulation boundary following the last body-part should be 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--

It should be noted that there appears to be room for additional information prior to the first encapsulation boundary and following the final such boundary. For several reasons, however, it is specified that these areas should be left blank, and that implementations should ignore anything that appears before the first boundary or after the last one. (The most important reasons are the lack of proper typing of these parts and lack of clear semantics for handling these parts at gateways, particularly X.400 gateways.)

The use of "Content-Type: Multipart" as a message part within another "Content-Type: Multipart" is explicitly allowed. In such cases, for obvious reasons, care must be taken to ensure that each nested mulitpart message should use a different boundary delimiter. See Appendix II for an example of nested multipart messages.

The use of content-type "Multipart" with only a single included part may be useful in certain contexts, and is explicitly permitted.

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

```
body := 1*encapsulation close-delimiter

encapsulation := delimiter CRLF message

delimiter := "--" <delimiter from Content-type resource>

close-delimiter := delimiter "--"

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

The above description defines the default subtype of the multipart type, "mixed", which may be explicitly specified with a content-type of "multipart/mixed". Other subtypes are possible, but should be defined to be syntactically compatible with the "mixed" subtype. Unrecognized subtypes should be treated as being of subtype "mixed." (DISCUSSION: Conspicuously missing from the multipart type is a notion of *structure*. In general, it seems premature to try to standardize 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 (e.g. with a content-type of "Application/x-my-structure-subtype") 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 subtype, but will treat it as the default subtype (multipart/mixed) and will thus be able to show the user the parts that are recognized.)

This memo defines one particular subtype of multipart, the "digest" subtype. This type is syntactically identical to multipart, but the semantics are different. In particular, in a digest, all of the parts are assumed to be of type "Message". That is, each part is implicitly prefixed by a line that says "Content-type: message" followed by a blank line. This is provided in order to allow a more readable digest format that is largely compatible (except for the quoting convention) with RFC 934.

5.3 The "Text-Plus" Content-Type and "Richtext" subtype

There are many formats for representing what might be known as "extended text" -- text with embedded formatting and presentation information. An interesting characteristic of most 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 non-readable data as images or audio messages. In the absence of appropriate interpreting software, it is reasonable to show extended text to the user, while it is not reasonable to do so with binary data.

To represent such data, this memo defines a "text-plus" content-type. Plausible subtypes of text-plus are typically given by the common name of the representation format, e.g. "text-plus/Troff" or "text-plus/TeX". Character sets are not specified as subtypes; in general it is assume that rich text formats will have their own mechanisms for representing alternate or multiple character sets. However, a subtype can be defined to permit such a specification, e.g. "text-plus/troff; charset=ISO-8859-1". Initial subtypes include troff, TeX, and PostScript

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

1. The syntax is 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 is easily extended to allow for new formatting commands that are deemed essential.

3. The capabilities are 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 it can be properly displayed.

4. The syntax is 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, richtext is several orders of magnitude simpler than full SGML, and no SGML knowledge is required in order to understand the richtext specification.)

The syntax of "richtext" is very simple. It is assumed, at the top-level, to be in the US-ASCII character set. All characters represent themselves, with the exception of the "<" character (ASCII 60), which is used to begin a formatting sequence. Formatting sequences consists of formatting commands surrounded by angle brackets ("<>", ASCII 60 and 62). Each formatting command may be no more than 40 characters in length. Formatting commands that begin with a forward slash or solidus (ASCII 47) are negations, and such negations must always exist to balance the initial opening commands. Thus, if the formatting sequence "<bold>" appears at some point, there must later be a "</bold>" to balance it. There are only two exceptions to this "balancing" rule: First, the command "<lt>" may be used to represent a literal "<" character. Second, the command "<nl> may be used to represent a line break. (NOTE: These are intended to be mnemonic: "lt" stands for "less than", and "nl" stands for "new line".)

Initially defined formatting commands are:

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 both margins.
Subscript -- causes the subsequent text to be interpreted as a subscript.
Superscript -- causes the subsequent text to be interpreted as a superscript.
ISO-10646 -- causes the subsequent text to be interpreted as text in the ISO-10646 character set.
ISO-8859-X (for any registered value of X) -- 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. Although this is the default character set, it might be usefully nested inside another character set.
Excerpt -- causes the subsequent text to be interpreted as a textual excerptfrom another message. Typically this will be displayed using indentation and an alternate font, but such decisions are up to the viewer.
Comment -- causes the subsequent text to be interpreted as a comment, and hence not shown to the reader. (Comments may be used, among other things, for annotating richtext documents with information that will be useful upon translation into some richer document format.>
No-op -- has no effect on the subsequent text.

Each formatting command affects all subsequent text until the matching </token>. Such pairs of tokens must be properly balanced. Thus, the proper way to describe text in ***bold italics*** is:

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

and, in particular, the following is illegal richtext:

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

Implementations should regard any unrecognized formatting token as equivalent to "No-op", thus facilitating future extensions to "richtext". Private extensions may defined using formatting tokens that begin with "X-", by analogy to Internet mail headers.

Richtext also differentiates betweeen "hard" and "soft" line breaks. A line break (CR LF) 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 should be displayed as such), the "<nl>" formatting token should be used.

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

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

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 memo):

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

country. -- the end

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> token and the next following </comment> token, and removes all other formatting tokens (all text enclosed in angle brackets).

NOTE ON THE RELATIONSHIP OF RICHTEXT TO SGML: Richtext is decidedly not SGML, and should not be used to transport arbitrary SGML documents. Those who wish to use SGML document types as a mail transport format should define a new text-plus subtype, e.g. "text-plus/sgml-dtd-whatever". Richtext is designed to be compatible with SGML, and specifically so that it will be possible to define a richtext DTD if that is desired. 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 memo is a complete definition of richtext.

One of the major goals in the design of richtext is to make it so simple that even text-only mailers would implement richtext-to-plain-text translators, thus increasing the likeihood that multifont text will become "safe" to use very widely. To demonstrate this simplicity, what follows is a 31-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; (c = getc(stdin)) != '>'; ++i) {
                token[i] = isupper(c) ? tolower(c) : c;
            }
            token[i] = NULL;
            if (!strcmp(token, "lt")) {
                putc('<', stdout);
            } else if (!strcmp(token, "nl")) {
                putc('\n', stdout);
            } else if (!strcmp(token, "comment")) {
                while (strcmp(token, "/comment")) {
                    while ((c = getc(stdin)) != '<') ;
                    for (i=0; (c = getc(stdin)) != '>'; ++i) {
                        token[i] = isupper(c) ? tolower(c) : c;
                    }
                    token[i] = NULL;
                }
            } /* Ignore all other tokens */
        } else if (c != '\n') {
            putc(c, stdout);
        }
    }
    putc('\n', stdout); /* for good measure */
}
```

5.4 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 hereby defined.

A content-type of "message" with no subtype indicates that the body or body part is an encapsulated message, with the syntax of an RFC 822 message, as extended by this memo.

The special subtype "pem" may be used to indicate that the body or body part is a message conforming to the Privacy Enhanced Mail protocol [RFC-1113].

The special subtype "partial" may be used to indicate that the body or body part is a fragment of a larger message. Three subfields must be specified in the content-type field: The first 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 can be similar to 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 memo.) The second, an integer, is the part number. The third, another integer, is the total number of parts. Thus, part 2 of a 3-part message might have the following header field:

```
Content-type: Message/Partial;
      "oc=jpbe0M2Yt4s@thumper.bellcore.com; 2; 3"
```

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. (EXPLANATION: The purpose of the MESSAGE/PARTIAL type is to allow large objects to be delivered as several separate pieces of mail and automatically reassembled by the receiving user agent. This may be desirable when intermediate transport agents limit the size of messages that can be sent.)

Additionally, all the character set subtypes of text are defined as subtypes of "message." If a character set subtype is given, it applies to the bodies, though not the names, of each of the encapsulated message’s header fields *except* the Content-XXX header fields, which must be entirely in US-ASCII. Thus it can be used to represent address and subject information in non-ASCII character sets. The character set subtype does NOT apply to the body of the encapsulated message. Thus, to encapsulate a message with non-ASCII characters in both the header fields and in the body, you would need something like the following:

```
From: <ASCII form>
Subject:  <ASCII form>
Content-type:  message/iso-8859-2

From: <iso-8859-2-form>
```

```
Subject: <iso-8859-2-form>
Content-type: text/iso-8859-2

Message body in iso-8859-2 character set.
```

5.5 The Binary Content-Type

A content-type of "binary" may be used to Indicate that the body or body part is binary data. A subtype may be specified, but none are defined here. The parameters for type binary are a set of attribute/value pairs, of the form "NAME=VALUE", separated by the usual semicolons. The set of possible attributes to be defined includes, but is not limited to:

> *NAME* -- a suggested name for the binary data as a file.
>
> *TYPE* -- the type of binary data
>
> *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 should be specified in the order they were applied, and separated by commas.

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

```
Content-type:  binary; name=foo.tar.Z.uu; type=tar;
        "conversions=compress,uuencode"
```

However, the use of such mechanisms as uuencode and compress is *explicitly 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 binary mail of an unrecognized type 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 header field (e.g. the "type=" subfield of a binary content-type) is found and executed using the mail body as input. The recommended action for an implementation that receives binary mail of an unrecognized type is to simply decode any Content-Transfer-Encoding and put the data in a file for the end-user.

Among the subtypes that have been suggested as suitable subtypes of "binary" are such document representation formats as "DVI" and "ODA".

5.6 The Application Content-Type Value

The "application" content-type is to be used for mail-based applications. The notion of mail-based application is an application that defines a standard format for representing intermediate data that is to be manipulated by cooperating user agents. 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 more based on that dialog.

Such applications may be defined as subtypes of the "application" content-type. There is no default subtype for application, and this memo defines only one subtype, the "external-reference" subtype.

The External-Reference subtype indicates that the body or body part is not included, presumably because too much data is involved for the underlying mail transport mechanism to handle. The subfields are, as in the case of the "binary" content-type, attribute-value pairs. In this case, the subfields describe a mechanism for accessing the external binary data. The set of possible attributes includes, but is not limited to:

> *FILENAME* -- The name of a file that contains the external data.
>
> *SITE* -- one or more domain names, comma separated, 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", 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.
>
> REAL-TYPE -- The real content-type of the data, once retrieved.
>
> EXPIRATION -- The date (in RFC 822 date syntax) after which the existence of the external data is not guaranteed.

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 makes 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. However, the external-reference mechanism is not intended to be limited to file retrieval. One can imagine, for example, using a LISTSERV mechanism, or using unique identifiers and a video server for external references to video clips. However, this memo explicitly defines only the FILENAME and SITE attributes for retrieval purposes, as this is the only retrieval method that is currently widely applicable. Other attributes may be defined as needed.

The "REAL-TYPE" attribute may be used to specify a new content-type header field to be applied to the data once retrieved, as the data are assumed to be only the body of a message, not including any header information. Note that because of the syntax of parameters, they may be quoted by enclosing an entire parameter in double quotes. Thus an external reference to an image in G3FAX format might have the following content-type header field:

```
Content-Type: application/external-reference;
     name=/usr/local/images/contact.g3;
     site=thumper.bellcore.com;
     real-type=image/g3fax;
     expiration="Fri, 14 Jun 1991 19:13:14 -0400
     (EDT)"
```

If a message is of content-type "application/external-reference", then the actual body of the message is ignored.

The distinction between the application and binary content-types is more a difference of intent than of syntax. The application content-type is used to indicate data that are intended to be interpreted by a mail-based user agent of some sort. The binary content-type is intended for the transport of arbitrary binary data, typically data that are used independently of a mail system, and for which mail transport is used as a convenient alternative to other file and data transport mechanisms.

5.7 The Audio, Image, and Video Content-Type Values

This memo defines several morecontent-type values that are defined only incompletely here, and await further practical experience before their values can be more completely specified. These are clearly experimental in nature, and are partially defined here in order to encourage experimenters to move in a common direction, regcognizing that future additional standardization will be needed.

AUDIO -- Indicates that the body or body part contains audio data. The subtype specifies the audio representation format. It is expected that such subtypes will be defined by future standards. In the meantime, vendor formats may be marked by subtypes such as "audio/x-sun", "audio/x-next", and "audio/x-mac".

IMAGE --Indicates that the body or body part contains an image. The subtype names the specific image format. A few such case insensitive values are "G3Fax" for Group Three Fax, "jpeg" for the JPEG format, and "pbm", "pgm", and "ppm" for the "portable bitmap" formats for black and white, grey scale, or color images.

VIDEO -- Indicates that the body or body part 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 and possible parameter values are left undefined by this memo.

5.8 Experimental ("X-") Content-Type Values

A content-type value beginning with the characters "X-" and not defined here or in another RFC is a private value, to be used by consenting mail systems by mutual agreement. Any format without a rigorous and public definition should be named with an "X-" prefix. Older versions of the widely-used Andrew system use the "X-BE2" name, so new systems should probably choose a different name.

6 Conformance With this Memo

The mechanisms described in this memo are open-ended. It is definitely not expected that all implementations will implement 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 "XXXX-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.

An XXXX-conformant mail user agent must:

1. Recognize the Content-Transfer-Encoding header field, and decode data encoded with either the quoted-printable or base64 implementations. (If a compressed encoding is ever agreed to, it should also become part of all conformant user agents.)

2. Recognize and interpret the Content-type header field, and avoid showing an unsuspecting user raw data that has a content-type field other than text.

3. Explicitly handle the following content-type values, to at least the following extents:

Text:

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

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

-- Recognize the "ISO-8859-1" subtype to the extent of being able to display those characters that are common to ISO-8859-1 and US-ASCII.

-- Never compose text mail without including a "Content-type" header specifying the appropriate subtype (character set).

Text-plus:

-- For unrecognized subtypes, show or offer to show the user the "raw" version of the data. An ability to convert "text-plus/richtext" to plain text is encouraged, but not required for conformance.

Message:

--Recognize and display at least the default (simple) encapsulation.

Multipart:

-- Recognize and display the default (mixed) subtype, although parallel parts may be serialized.

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

Binary:

-- Offer the ability to remove any Content-Transfer-Encoding and put the resulting information in a user file.

4. Upon encountering any unrecognized content-type, an implementation should treat it as if it had a content-type of "binary" with no parameter sub-arguments. How such data is 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 might include audio, image, video, or application, should also be treated in this way.

A user agent that meets the above conditions is said to be XXXX-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 they will at least be able to treat the data as undifferentiated binary, and will not simply splash it onto the screen of unsuspecting users. Of course, there is another sense in which it is always "safe" to send XXXX-conformant format data, which is that it 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 XXXX-conformant have the additional guarantee that the user will not be shown data that were never intended to be viewed as text.

Appendix I -- Guidelines For Sending Data Via Email

Internet email is not yet a perfect, homogenous 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 MTA's in the Internet. These MTA's, speaking the SMTP protocol, alter messages on the fly to take advantage of 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 MTA's 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 gateways to systems that use the EBCDIC character set.

(1) Delimiters other than CR-LF pairs may be used in the local representation of a message on some systems. The persistence of CR-LF pairs should not be relied on.

(2) 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 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 users of US-ASCII format should not rely on the persistence of TAB characters.

(4) Lines longer than 78 characters may be wrapped or truncated in some environments. Line wrapping and line truncation are STRONGLY DISCOURAGED, but unavoidable in some cases. Applications which depend on lines not being wrapped should use mechanisms other than unencoded US-ASCII bodyparts to transmit messages.

(5) Trailing "white space" characters (SPACE, TAB, etc.) 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. In particular, the only characters that are known to be consistent across all gateways are the 62 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 taken from this set of 73 characters.

Please note that the above list is NOT a list of recommended practices for MTA’s. RFC 821 MTA’s are prohibited from altering the character of white space, or wraping long lines. These BAD and illegal practices are know to occur on established networks, and implementions should be robust in dealing with the bad effects they can cause.

Appendix II -- A Complex Multipart Example

What follows is the outline of a complex multipart message. This message has three parts to be displayed serially: an introductory plain text part, an embedded multipart message, 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.

```
From: ...
Subject: ...
Content-type: multipart; 1-s; tweedledum

This is a multipart message.
Since I've not specified another character set,
this "prefix" area is in US ASCII.
--tweedledum

...Some more text appears here...
[Note that the preceding blank line means
no header fields were given and this is text,
with charset US ASCII.]
--tweedledum
Content-type: multipart; 1-p; tweedledee

This is a multipart message.
If you are reading this text, you might want to
consider changing to a user agent that understands
how to properly display multipart messages.
--tweedledee
Content-type: x-NeXT
Content-Transfer-Encoding: base64

... base64-encoded NeXT-format audio data goes here....
--tweedledee
Content-type: image/G3FAX
Content-Transfer-Encoding: Base64

... base64-encoded FAX data goes here....
--tweedledee--
--tweedledum
Content-type: message/ISO-8859-1

From: (name in ISO-8859-1)
Subject: (subject in ISO-8859-1)
Content-type: Text/ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

... Closing text in ISO-8859-1 goes here ...
--tweedledum--
```

Appendix III -- The US-ASCII Character Set

The following table explicitly defines the default character set for Internet mail, "US-ASCII":

```
     0 nul                    @  64 Commercial at
     1 soh                    A  65 Latin capital letter a
     2 stx                    B  66 Latin capital letter b
     3 etx                    C  67 Latin capital letter c
     4 eot                    D  68 Latin capital letter d
     5 enq                    E  69 Latin capital letter e
     6 ack                    F  70 Latin capital letter f
     7 bel                    G  71 Latin capital letter g
     8 bs                     H  72 Latin capital letter h
     9 ht                     I  73 Latin capital letter i
    10 lf                     J  74 Latin capital letter j
    11 vt                     K  75 Latin capital letter k
    12 np                     L  76 Latin capital letter l
    13 cr                     M  77 Latin capital letter m
    14 so                     N  78 Latin capital letter n
    15 si                     O  79 Latin capital letter o
    16 dle                    P  80 Latin capital letter p
    17 dc1                    Q  81 Latin capital letter q
    18 dc2                    R  82 Latin capital letter r
    19 dc3                    S  83 Latin capital letter s
    20 dc4                    T  84 Latin capital letter t
    21 nak                    U  85 Latin capital letter u
    22 syn                    V  86 Latin capital letter v
    23 etb                    W  87 Latin capital letter w
    24 can                    X  88 Latin capital letter x
    25 em                     Y  89 Latin capital letter y
    26 sub                    Z  90 Latin capital letter z
    27 esc                    [  91 Left square bracket
    28 fs                     \  92 Reverse solidus
    29 gs                     ]  93 Right square bracket
    30 rs                     ˆ  94 Circumflex accent
    31 us                     _  95 Low line
    32 Space                  ‘  96 Grave accent
 !  33 Exclamation mark       a  97 Latin small letter a
 "  34 Quotation mark         b  98 Latin small letter b
 #  35 Number sign            c  99 Latin small letter c
 $  36 Dollar sign            d 100 Latin small letter d
 %  37 Percent sign           e 101 Latin small letter e
 &  38 Ampersand              f 102 Latin small letter f
 ’  39 Apostrophe             g 103 Latin small letter g
 (  40 Left parenthesis       h 104 Latin small letter h
 )  41 Right parenthesis      i 105 Latin small letter i
 *  42 Asterisk               j 106 Latin small letter j
 +  43 Plus sign              k 107 Latin small letter k
 ,  44 Comma                  l 108 Latin small letter l
 -  45 Hyphen, minus sign     m 109 Latin small letter m
 .  46 Full stop              n 110 Latin small letter n
 /  47 Solidus                o 111 Latin small letter o
 0  48 Digit zero             p 112 Latin small letter p
 1  49 Digit one              q 113 Latin small letter q
 2  50 Digit two              r 114 Latin small letter r
 3  51 Digit three            s 115 Latin small letter s
 4  52 Digit four             t 116 Latin small letter t
 5  53 Digit five             u 117 Latin small letter u
 6  54 Digit six              v 118 Latin small letter v
 7  55 Digit seven            w 119 Latin small letter w
 8  56 Digit eight            x 120 Latin small letter x
 9  57 Digit nine             y 121 Latin small letter y
 :  58 Colon                  z 122 Latin small letter z
 ;  59 Semicolon              { 123 Left curly bracket
 <  60 Less-than sign         | 124 Vertical line
 =  61 Equals sign            } 125 Right curly bracket
 >  62 Greater-than sign      ˜ 126 Tilde
 ?  63 Question mark            127 Del
```

Summary

Using the 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 broken, and care has been taken to avoid problems caused by additional restrictions imposed by the characteristics of some Internet mail transport mechanisms (see Appendix I). The "multipart" and "message" content-types allow mixing and heirarchical structuring of objects of different types in a single message. Further content-tyes allow a standardized mechanism for tagging messages or mesage parts as audio, image, or several other kinds of data. Additional optional header fields provide conventional 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.

Contacts

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

Nathaniel Borenstein <nsb@thumper.bellcore.com>
Ned Freed <ned@innosoft.com>

Acknowledgements

This memo is the result of the collective effort of a large number of people, at several IETF meetings and on the IETF-SMTP and IETF-822 mailing lists. Although any enumeration seems doomed to suffer from egregious omissions, the following are among the many contributors to this effort: Harald Alvestrand, Randall Atkinson, Kevin Carosso, Mark Crispin, Dave Crocker, Terry Crowley, Walt Daniels, Frank Dawson, Hitoshi Doi, Kevin Donnelly, Johnny Eriksson, Craig Everhart, Roger Fajman, Alain Fontaine, Philip Gladstone, Thomas Gordon, Phill Gross, David Herron, Bruce Howard, Bill Janssen, Risto Kankkunen, Phil Karn, Tim Kehres, Neil Katin, Steve Kille, Anders Klemets, John Klensin, Valdis Kletniek, Stev Knowles, Bob Kummerfeld, Vincent Lau, Timo Lehtinen, John MacMillan, Rick McGowan, Leo Mclaughlin, Goli Montaser-Kohsari, Keith Moore, Mark Needleman, John Noerenberg, Mats Ohrman, David J. Pepper, Jonathan Rosenberg, Jan Rynning, Mark Sherman, Keld Simonsen, Bob Smart, Einar Stefferud, Michael Stein, Peter Svanberg, Steve Uhler, Stuart Vance, Erik van der Poel, Peter Vanderbilt, Greg Vaudreuil, Brian Wideen, Glenn Wright, and David Zimmerman. The authors apologize for any omissions from this list, which were certainly unintentional.

References

[REF-ISO646] International Standard--Information Processing--ISO 7-bit coded character set for information interchange, ISO 646:1983.

[REF-ISO-2022] International Standard--Information Processing--ISO 7-bit and 8-bit coded character sets--Code extension techniques, ISO 2022:1986.

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

[REF-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-821] Postel, J.B. Simple Mail Transfer Protocol. August, 1982, Network Information Center, RFC-821.

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

[RFC-934] Rose, M.T.; Stefferud, E.A. Proposed standard for message encapsulation. January, 1985, Network Information Center, RFC-934.

[RFC-1049] Sirbu, M.A. Content-type header field for Internet messages. March, 1988, Network Information Center, RFC-1049.

[RFC-1113] Linn, J. Privacy enhancement for Internet electronic mail: Part I - message encipherment and authentication procedures [Draft]. August, 1989, Network Information Center, RFC-1113.

[RFC-1154] Robinson, D.; Ullmann, R. Encoding header field for internet messages. April, 1990, Network Information Center, RFC-1154.

[REF-RFC-QR] Prindeville, Phillipe-Andrew’, and Keld Simonsen, "A Portable, Extensible Message Encoding Format for Alphabetic Scripts", Internet RFC, in preparation.

[REF-ISO-10646] Draft International Standard -- Information Technology -- Universal Coded Character Set (UCS), ISO/IEC DIS 10646:1990.

[REF-ISO-8859] **********

Table of Contents