Understaning SMTP, MIME, POP3 protocols

Simple Mail Transfer Protocol (SMTP)

Electronic mail (e-mail) is probably the most widely used TCP/IP application. The basic Internet mail protocols provide mail (note) and message exchange between TCP/IP hosts; facilities have been added for the transmission of data that cannot be represented as 7-bit ASCII text.

There are three standard protocols that apply to mail of this kind. Each is *recommended*. The term SMTP is frequently used to refer to the combined set of protocols, since they are so closely inter-related, but strictly speaking SMTP is just one of the three. Normally, it is evident from the context which of the three protocols is being referred to. Whenever some doubt might exist, we refer to the STD or RFC numbers to avoid ambiguity. The three standards are:

- A standard for exchange of mail between two computers (STD 10/RFC 821), which specifies the protocol used to send mail between TCP/IP hosts. This standard is SMTP itself.
- A standard (STD 11) on the format of the mail messages, contained in two RFCs. RFC 822 describes the syntax of mail header fields and defines a set of header fields and their interpretation. RFC 1049 describes how a set of document types other than plain text ASCII can be used in the mail body (the documents themselves are 7-bit ASCII containing imbedded formatting information: PostScript, Scribe, SGML, TEX, TROFF and DVI are all listed in the standard).

  The official protocol name for this standard is MAIL.

- A standard for the routing of mail using the Domain Name System, described in RFC 974. The official protocol name for this standard is DNS-MX.

The STD 10/RFC 821 dictates that data sent via SMTP is 7-bit ASCII data, with the high-order bit cleared to zero. This is adequate in most instances for the transmission of English text messages, but is inadequate for non-English text or non-textual data. There are two approaches to overcoming these limitations:

- Multipurpose Internet Mail Extensions (MIME), defined in RFCs 2045 to 2049, which specifies a mechanism for encoding text and binary data as 7-bit ASCII within the mail envelope defined by RFC 822. MIME is described in 4.8, “Multipurpose Internet Mail Extensions (MIME)” on page 193.
- SMTP Service Extensions, which define a mechanism to extend the capabilities of SMTP beyond the limitations imposed by RFC 821. There are three current RFCs that describe SMTP Service Extensions:
  - A standard for a receiver SMTP to inform a sender SMTP which service extensions it supports (RFC 1869).

    RFC 1869 modifies RFC 821 to allow a client SMTP agent to request that the server respond with a list of the service extensions that it supports at the start of an SMTP session. If the server SMTP does not support RFC 1869, it will respond with an error and the client can either terminate the session or attempt to start a session according to the rules of RFC 821. If the server does support RFC 1869, it can also respond with a list of the service extensions that it supports. A registry of services is maintained by IANA. The initial list defined in RFC 1869 contains those commands listed in RFC 1123 *Requirements for Internet Hosts — Application and Support* as optional for SMTP servers.

    Other service extensions are defined via RFCs in the usual manner. The next two RFCs define specific extensions:

  - A protocol for 8-bit text transmission (RFC 1652) that allows an SMTP server to indicate that it can accept data consisting of 8-bit bytes. A server that reports that this extension is available to a client must leave the high order bit of bytes received in an SMTP message unchanged if requested to do so by the client.

    The MIME and SMTP Service Extension approaches are complementary rather than competing standards. In particular, RFC 1652 is titled *SMTP Service Extension for 8-bit-MIMEtransport*, since the MIME standard allows messages to be declared as consisting of 8-bit data rather than 7-bit data. Such messages cannot be transmitted by SMTP agents that strictly conform to RFC 821, but can be transmitted when both the client and the server conform to RFCs 1869 and 1652. Whenever a client SMTP attempts to send 8-bit data to a server that does not support this extension, the client SMTP must either encode the message contents into a 7-bit representation compliant with the MIME standard or return a permanent error to the user.

    This service extension does not permit the sending of arbitrary binary data because RFC 821 defines the maximum length of a line that an SMTP server is required to accept as 1000 characters. Non-text data could easily have sequences of more than 1000 characters without a <CRLF> sequence.

Note: The service extension specifically limits the use of non-ASCII characters (those with values above decimal 127) to message bodies. They are *not* permitted in RFC 822 message headers.

– A protocol for message size declaration (RFC 1870) that allows a server to inform a client of the maximum size message it can accept. Without this extension, a client can only be informed that a message has exceeded the maximum size acceptable to the server (either a fixed upper limit or a temporary limit imposed by a lack of available storage space at the server) after transmitting the entire message. When this happens, the server discards the failing message. If both client and server support the Message Size Declaration extension, the client may declare an estimated size of the message to be transferred and the server will return an error if the message is too large.

Each of these SMTP Service Extensions is a *draft standard protocol* and each has a status of *elective*.

How SMTP Works

SMTP (that is, STD 11/RFC 821) is based on *end-to-end delivery*; an SMTP client will contact the destination host's SMTP server directly to deliver the mail. It will keep the mail item being transmitted until it has been successfully copied to the recipient's SMTP. This is different from the store-and-forward principle that is common in many mailing systems, where the mail item may pass through a number of intermediate hosts in the same network on its way to the destination and where successful transmission from the sender only indicates that the mail item has reached the first intermediate hop.

In various implementations, there is a possibility to exchange mail between the TCP/IP SMTP mailing system and the locally used mailing systems. These applications are called *mail gateways* or *mail bridges*. Sending mail through a mail gateway can alter the end-to-end delivery specification, since SMTP will only guarantee delivery to the mail-gateway host, not to the real destination host, which is located beyond the TCP/IP network. When a mail gateway is used, the SMTP end-to-end transmission is host-to-gateway, gateway-to-host or gateway-to-gateway; the behavior beyond the gateway is not defined by SMTP. CSNET provides an interesting example of mail gateway service. Started as a low-cost facility to interconnect scientific and corporate research centers, CSNET operates a mail gateway service that allows subscribers to send and receive mail across the Internet using only a dial-up modem. The mail gateway polls the subscribers at regular times, delivers mail that was addressed to them and picks up the outgoing mail. Although this is not a direct end-to-end delivery, it has proven to be a very useful system.

Each message has:

- A header, or envelope, the structure of which is strictly defined by RFC 822.

  The mail header is terminated by a null line (that is, a line with nothing preceding the <CRLF> sequence). However, some implementations (for example VM, which does not support zero-length records in files) may interpret this differently and accept a blank line as a terminator.

- Contents

Everything after the null (or blank) line is the message body which is a sequence of lines containing ASCII characters (that is, characters with a value less than 128 decimal).

RFC 821 defines a client/server protocol. As usual, the client SMTP is the one that initiates the session (that is, the sending SMTP) and the server is the one that responds (the receiving SMTP) to the session request. However, since the client SMTP frequently acts as a server for a user mailing program, it is often simpler to refer to the client as the sender SMTP and to the server as the receiver SMTP.

Mail Header Format

The user normally doesn't have to worry about the message header, since it is taken care of by SMTP itself. A short reference is included below for completeness.

RFC 822 contains a complete lexical analysis of the mail header. The syntax is written in a form known as the augmented Backus-Naur Form (BNF). RFC 822 contains a description of augmented BNF, and many RFCs that are related to RFC 822 use this format. RFC 822 describes how to parse a mail header to a *canonical representation*, unfolding continuation lines, deleting insignificant spaces, removing comments and so on. The syntax is powerful, but relatively difficult to parse. A basic description is given here, which should be adequate for the reader to interpret the meaning of simple mail headers that he or she encounters. However, this description is too great a simplification to understand the details workings of RFC 822 mailers; for a full description, refer to RFC 822.

Briefly, the header is a list of lines, of the form:

```
field-name: field-value
```

Fields begin in column 1. Lines beginning with white space characters (SPACE or TAB) are continuation lines that are unfolded to create a single line for each field in the canonical representation. Strings enclosed in ASCII quotation marks indicate single tokens within which special characters such as the colon are not significant. Many important field values (such as those for the To and From fields) are *mailboxes*. The most common forms for these are:

```
octopus@garden.under.the.sea
The Octopus <octopus@garden.under.the.sea>
"The Octopus" <octopus@garden.under.the.sea>
```

The string The Octopus is intended for human recipients and is the name of the mailbox owner. The string octopus@garden.under.the.sea is the machine-readable address of the mailbox. (The angle brackets are used to delimit the address but are not part of it.) One can see that this form of addressing is closely related to the Domain Name System concept. In fact, the client SMTP uses the Domain Name System to determine the IP address of the destination mailbox.

Some frequently used fields are:

keyword	**value**
to	Primary recipients of the message.
cc	Secondary (carbon-copy) recipients of the message.
from	Identity of sender.
reply-to	The mailbox to which responses are to be sent. This field is added by the originator.

return-path Address and route back to the originator. This field is added by the final transport system that delivers the mail.

Subject Summary of the message. This is usually provided by the user.

Mail Exchange

The SMTP design is based on the model of communication shown in Figure 127. As a result of a user mail request, the sender SMTP establishes a two-way connection with a receiver SMTP. The receiver SMTP can be either the ultimate destination or an intermediate (mail gateway). The sender SMTP will generate commands that are replied to by the receiver SMTP.

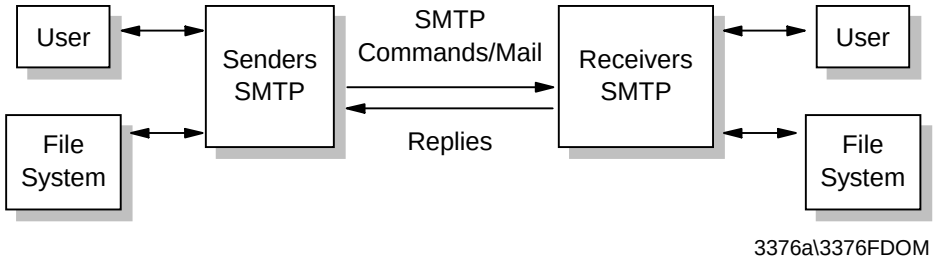

Figure 127. SMTP - Model for SMTP

SMTP Mail Transaction Flow: Although mail commands and replies are rigidly defined, the exchange can easily be followed in Figure 128 on page 189. All exchanged commands/replies/data are text lines, delimited by a <CRLF>. All replies have a numeric code at the beginning of the line.

1. The sender SMTP establishes a TCP connection with the destination SMTP and then waits for the server to send a 220 Service ready message or a 421 Service not available message when the destination is temporarily unable to proceed.

2. HELO (HELO is an abbreviation for hello) is sent, to which the receiver will identify himself or herself by sending back its domain name. The sender-SMTP can use this to verify if it contacted the right destination SMTP.

   If the sender SMTP supports SMTP Service Extensions as defined in RFC 1869, it may substitute an EHLO command in place of the HELO command. A receiver SMTP that does not support service extensions will respond with a 500 Syntax error, command unrecognized message. The sender SMTP should then retry with HELO, or if it cannot transmit the message without one or more service extensions, it should send a QUIT message.

   If a receiver-SMTP supports service extensions, it responds with a multi-line 250 OK message, which includes a list of service extensions that it supports.

3. The sender now initiates the start of a mail transaction by sending a MAIL command to the receiver. This command contains the reverse-path which can be used to report errors. Note that a path can be more than just the user mailbox@host domain name pair. In addition, it can contain a list of routing hosts. Examples of this are when we pass a mail bridge, or when we provide explicit routing information in the destination address. If accepted, the receiver replies with a 250 OK.

4. The second step of the actual mail exchange consists of providing the server SMTP with the destinations for the message. There can be more than one recipient.) This is done by sending one or more RCPT TO:<forward-path>

commands. Each of them will receive a reply 250 OK if the destination is known to the server, or a 550 No such user here if it isn't.

5. When all RCPT commands are sent, the sender issues a DATA command to notify the receiver that the message contents are following. The server replies with 354 Start mail input, end with <CRLF>.<CRLF>. Note the ending sequence that the sender should use to terminate the message data.
6. The client now sends the data line by line, ending with the 5-character sequence <CRLF>.<CRLF> line upon which the receiver acknowledges with a 250 OK or an appropriate error message if anything went wrong.
7. We now have several possible actions:
   - The sender has no more messages to send. He or she will end the connection with a QUIT command, which will be answered with a 221 Service closing transmission channel reply.
   - The sender has no more messages to send, but is ready to receive messages (if any) from the other side. He or she will issue the TURN command. The two SMTPs now switch their role of sender/receiver and the sender (previously the receiver) can now send messages by starting with step 3 above.
   - The sender has another message to send, and simply goes back to step 3 to send a new MAIL command.

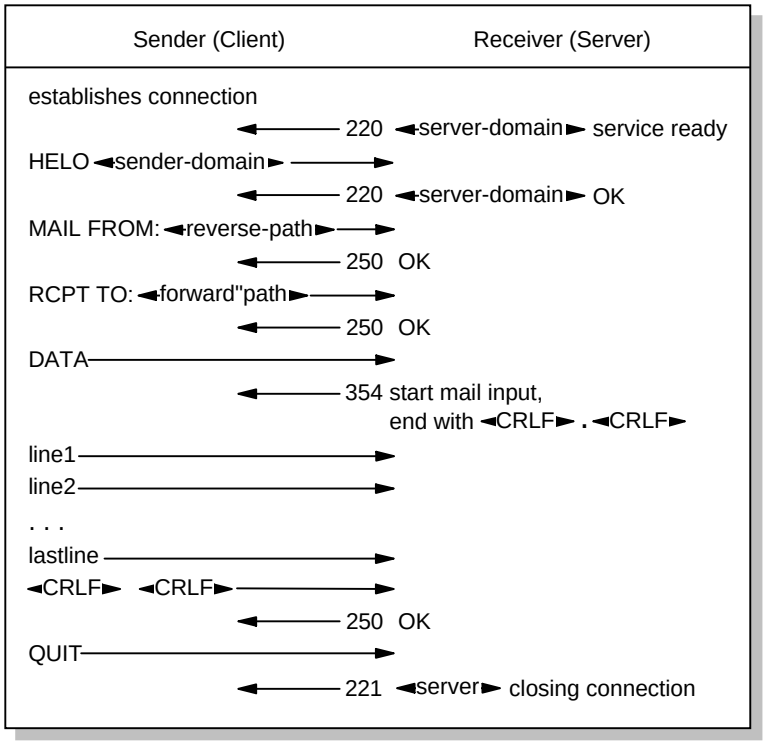

Figure 128. SMTP - Normal SMTP Data Flow. One mail message is delivered to one destination mailbox.

The SMTP Destination Address (Mailbox Address): Its general form is local-part@domain-name and can take several forms:

user@host For a direct destination on the same TCP/IP network.

user%remote-host@gateway-host

For a user on a non-SMTP destination remote-host, via the mail gateway gateway-host.

@host-a,@host-b:user@host-c

For a relayed message. This contains explicit routing information. The message will first be delivered to host-a, who will resend (relay) the message to host-b. Host-b will then forward the message to the real destination host-c. Note that the message is stored on each of the intermediate hosts, so we don't have an end-to-end delivery in this case.

In the above description, only the most important commands were mentioned. All of them are commands that must be recognized in each SMTP implementation. Other commands exist, but most of those are only optional; that is, the RFC standard does not require them to be implemented everywhere. However, they implement very interesting functions such as relaying, forwarding, mailing lists, etc.

For a full list of command verbs, see RFC 821 *Simple Mail Transfer Protocol* and RFC 1123 *Requirements for Internet Hosts — Application and Support*. For details of SMTP service extensions, see RFC 1869 *SMTP Service Extensions*, RFC 1652 *SMTP Service Extension for 8-bit-MIMEtransport* and RFC 1870 *SMTP Service Extension for Message Size Declaration*.

Example: In the following scenario, user abc at host vm1.stockholm.ibm.com sends a note to users xyz, opq and rst at host delta.aus.edu. The lines preceded by R: are lines sent by the receiver; the S: lines are sent by the sender.

```
R: 220 delta.aus.edu Simple Mail Transfer Service Ready
S: HELO stockholm.ibm.com
R: 250 delta.aus.edu

S: MAIL FROM:<abc@stockholm.ibm.com>
R: 250 OK

S: RCPT TO:<xyz@delta.aus.edu>
R: 250 OK
S: RCPT TO:<opq@delta.aus.edu>
R: 550 No such user here
S: RCPT TO:<rst@delta.aus.edu>
R: 250 OK

S: DATA
R: 354 Start mail input, end with <CRLF>.<CRLF>
S: Date: 23 Jan 89  18:05:23
S: From: Alex B. Carver <abc@stockholm.ibm.com>
S: Subject: Important meeting
S: To:  <xyz@delta.aus.edu>
S: To:  <opq@delta.aus.edu>
S: cc:  <rst@delta.aus.edu>
S:
S: Blah blah blah
S: etc.....
S: .
R: 250 OK

S: QUIT
R: 221 delta.aus.edu Service closing transmission channel
```

Figure 129. SMTP - An Example Scenario

Note that the message header is part of the data being transmitted.

SMTP and the Domain Name System

If the network is using the domain concept, an SMTP cannot simply deliver mail sent to TEST.IBM.COM by opening a TCP connection to TEST.IBM.COM. It must first query the name server to find out to which host (again a domain name) it should deliver the message.

For message delivery, the name server stores resource records (RRs) known as MX RRs. They map a domain name to two values:

- A preference value. As multiple MX resource records may exist for the same domain name, a preference (priority) is assigned to them. The lowest preference value corresponds to the most preferred record. This is useful whenever the most preferred host is unreachable; the sending SMTP then tries to contact the next (less preferred) host.
- A host name.

It is also possible that the name server responds with an empty list of MX RRs. This means that the domain name is in the name server's authority, but has no MX assigned to it. In this case, the sending SMTP may try to establish the connection with the host name itself.

An important recommendation is given in RFC 974. It recommends that after obtaining the MX records, the sending SMTP should query for Well-Known Services (WKS) records for this host, and should check that the referenced host has SMTP as a WKS-entry.

Note: This is only an option of the protocol but is already widely implemented.

Here is an example of MX resource records:

```
fsc5.stn.mlv.fr.        IN    MX 0  fsc5.stn.mlv.fr.
                        IN    MX 2  psfred.stn.mlv.fr.
                        IN    MX 4  mvs.stn.mlv.fr.
                        IN    WKS   152.9.250.150 TCP (SMTP)
```

In the above example, mail for fsc5.stn.mlv.fr should by preference, be delivered to the host itself, but in case the host is unreachable, the mail might also be delivered to psfred.stn.mlv.fr or to mvs.stn.mlv.fr (if psfred.stn.mlv.fr is unreachable, too).

Addressing Mailboxes on Server Systems

When a user employs a server system for all mail functions, the mailbox address seen by other SMTP users refers exclusively to the mail server system. For example if two OS/2 systems are named:

```
hayes.itso.ral.ibm.com
```

and

```
itso180.itso.ral.ibm.com
```

with the first one being used as an UltiMail client and the second as an UltiMail server, the mailbox address might be:

```
hayes@itso180.itso.ral.ibm.com
```

This mailbox address would appear in the From: header field of all outgoing mail and in the SMTP commands to remote servers issued by the UltiMail server system.

When the user uses a POP server, however, the mailbox address on outbound mail items contains the workstation's hostname (for example, steve@hayes.itso.ral.ibm.com). In this case, the sender should include a Reply-To: field in the mail header to indicate that replies should *not* be sent to the originating mailbox. For example, the mail header might look like this:

```
Date: Fri, 10 Feb 95 15:38:23
From: steve@hayes.itso.ral.ibm.com
To: "Steve Hayes" <tsgsh@gford1.warwick.uk.ibm.com>
Reply-To: hayes@itso180.itso.ral.ibm.com
Subject: Test Reply-To: header field
```

The receiving mail agent is expected to send replies to the Reply-To: address and not the From: address.

Using the Domain Name System to Direct Mail: An alternative approach to using the Reply-To: header field is to use the Domain Name System to direct mail to the correct mailbox. The administrator for the domain name server with authority for the domain containing the user's workstation and the name server can add MX resource records to the Domain Name System to direct mail appropriately, as described in 4.7.2, “SMTP and the Domain Name System” on page 191. For

example, the following MX records indicate to client SMTPs that if the SMTP server on hayes.itso.ral.ibm.com is not available, there is a mail server on itso.180.ral.ibm.com (9.24.104.180) that should be used instead.

```
itso180.itso.ral.ibm.com.  IN     WKS  9.24.104.180 TCP (SMTP)

hayes.itso.ral.ibm.com.    IN     MX 0 hayes.itso.ral.ibm.com.
                           IN     MX 1 itso180.itso.ral.ibm.com.
```

References

A detailed description of the SMTP, MAIL and DNS-MX standards can be found in the following RFCs:

- *RFC 821 — Simple Mail Transfer Protocol*
- *RFC 822 — Standard for the format of ARPA Internet text messages*
- *RFC 974 — Mail Routing and the Domain System*
- *RFC 1049 — A Content Type Header Field for Internet messages*
- *RFC 1652 — SMTP Service Extension for 8-bit-MIMEtransport*

Multipurpose Internet Mail Extensions (MIME)

MIME is a *draft-standard protocol*. Its status is *elective*.

Electronic mail (as described in 4.7, “Simple Mail Transfer Protocol (SMTP)” on page 184) is probably the most widely used TCP/IP application. However, SMTP (that is, an STD 10/RFC 821-compliant mailing system) is limited to 7-bit ASCII text with a maximum line length of 1000 characters, which results in a number of limitations:

- SMTP cannot transmit executable files or other binary objects. There are a number of ad hoc methods of encapsulating binary items in SMTP mail items, for example:
  - Encoding the file as pure hexadecimal
  - The UNIX UUencode and UUdecode utilities that are used to encode binary data in the UUCP mailing system to overcome the same limitations of 7-bit transport
  - The Andrew Toolkit representation

  None of these can be described as a de facto standard. UUencode is perhaps the most pervasive due to the pioneering role of UNIX systems in the Internet.
- SMTP cannot transmit text data that includes national language characters since these are represented by code points with a value of 128 (decimal) or higher in all character sets based on ASCII.
- SMTP servers may reject mail messages over a certain size. Any given server may have permanent and/or transient limits on the maximum amount of mail data it can accept from a client at any given time.
- SMTP gateways that translate from ASCII to EBCDIC and vice versa do not use a consistent set of code page mappings, resulting in translation problems.
- Some SMTP implementations or other mail transport agents (MTAs) in the Internet do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
  - Removal of trailing white space characters (TABs and SPACEs)

- Padding of all lines in a message to the same length
- Wrapping of lines longer than 76 characters
- Changing of new line sequences between different conventions. (For instance <CR> characters may be converted to <CRLF> sequences.)
- Conversion of TAB characters to multiple SPACEs.

MIME is a standard that includes mechanisms to solve these problems in a manner that is highly compatible with existing RFC 822 standards. Because mail messages are frequently forwarded through mail gateways, it is not possible for an SMTP client to distinguish between a server that manages the destination mailbox and one that acts as a gateway to another network. Since mail that passes through a gateway may be tunnelled through further gateways, some or all of which may be using a different set of messaging protocols, it is not possible in general for a sending SMTP to determine the lowest common denominator capability common to all stages of the route to the destination mailbox. For this reason, MIME assumes the worst: 7-bit ASCII transport, which may not strictly conform to or be compatible with RFC 821. It does not define any extensions to RFC 821, but limits itself to extensions within the framework of RFC 822. Thus, a MIME message is one which can be routed through any number of networks that are loosely compliant with RFC 821 or are capable of transmitting RFC 821 messages.

MIME is a *draft-standard protocol* with a status of *elective*. It is described in five parts:

- Protocols for including objects other than US ASCII text mail messages within the bodies of messages conforming to RFC 822. These are described in RFC 2045.
- General structure of the MIME media typing system and defines an initial set of media types. This is described in RFC 2046.
- A protocol for encoding non-US ASCII text in the header fields of mail messages conforming to RFC 822. This is described in RFC 2047.
- Various IANA registration procedures for MIME-related facilities. This is described in RFC 2048.
- MIME conformance criteria. This is described in RFC 2049.

Although RFC 2045 provides a mechanism suitable for describing non-textual data from X.400 messages in a form that is compatible with RFC 822, it does not say how X.400 message parts are to be mapped to MIME message parts. The conversion between X.400 and MIME is defined in RFCs 1494, 2156 and 1496 which update the protocols for the conversion between RFC 822 and X.400.

The MIME standard was designed with the following general order of priorities:

1. Compatibility with existing standards such as RFC 822.

   There are two areas where compatibility with previous standards is not complete.

   - RFC 1049 (which is part of STD 11) described a Content-Type: field used to indicate the type of (ASCII text) data in a message body. PostScript or SGML would allow a user mail agent to process it accordingly. MIME retains this field, but changes the values that are defined for it. Since the correct response for a mail agent on encountering an unknown value in this

field is basically to ignore it, this does not raise any major compatibility concerns.

- RFC 934 discussed encapsulation of messages in the context of message forwarding and defined encapsulation boundaries: lines indicating the beginning and end of an encapsulated message. MIME retains broad compatibility with RFC 934, but does not include the quoting mechanism used by RFC 934 for lines in encapsulated messages that could otherwise be misinterpreted as boundaries.[9]

The most important compatibility issue is that the standard form of a MIME message is readable with an RFC 821-compliant mail reader. This is, of course, the case. In particular the default encoding for MIME message bodies is no encoding at all, just like RFC 822.

2. Robustness across existing practice. As noted above, there are many widely deployed MTAs in the Internet that do not comply with STD 10/RFC 821. The encoding mechanisms specified in RFC 2045 are designed to always circumvent the most common of these (folding of lines as short as 76 characters and corruption of trailing white space characters) by only transmitting short lines with no trailing white space characters, and allowing encoding of any data in a mail safe fashion.

   Note: MIME does *not* require mail items to be encoded; the decision is left to the user and/or the mail program. For binary data transmitted across (7-bit) SMTP, encoding is invariably required, but for data consisting mostly of text, this may not be the case.

   The preferred encoding mechanism for mostly text data is such that, at a minimum, it is mail-safe with any compliant SMTP agent on an ASCII system and at maximum is mail-safe with all known gateways and MTAs. The reason why MIME does not require maximum encoding is that the encoding hampers readability when the mail is transmitted to non-MIME compliant systems.

3. Ease of extension. RFC 2045 categorizes elements of mail bodies into seven *content-types*, which have *subtypes*. The content-type/subtype pairs in turn have parameters that further describe the object concerned. The RFC defines a mechanism for registering new values for these and other MIME fields with the Internet Assigned Numbers Authority (IANA). This process is itself updated by RFC 2048.

   For the current list of all MIME values, consult *STD 2 — Assigned Internet Numbers*. The remainder of this chapter describes only the values and types given in RFC 2045.

One consequence of this approach is that, to quote RFC 2045, “some of the mechanisms [used in MIME] may seem somewhat strange or even baroque at first. In particular, compatibility was always favored over elegance.”

Because RFC 822 defines the syntax of message headers (and deliberately allows for additions to the set of headers it describes) but not the composition of message bodies, the MIME standard is largely compatible with RFC 822, particularly the RFC

---

[9] The reason for this departure is that MIME allows for deeply nested encapsulation, but encodes text in such a way as to reversibly spill text lines at or before column 76 to avoid the lines being spilled irreversibly by non-conforming SMTP agents. The RFC 934 quoting mechanism can result in lines being lengthened with each level of encapsulation, possibly past column 76.

2045 part that defines the structure of message bodies and a set of header fields that are used to describe that structure.

MIME can be seen as a high-level protocol; since it works entirely within the boundaries of STD 10 and STD 11, it does not involve the transport layer (or lower layers) of the protocol stack at all.

How MIME Works

A MIME-compliant message must contain a header field with the following verbatim text:

```
MIME-Version: 1.0
```

As is the case with RFC 822 headers, the case of MIME header field names are never significant but the case of field values may be, depending on the field name and the context. For the MIME fields described below, the values are case-insensitive unless stated otherwise.

The general syntax for MIME header fields is the same as that for RFC 822, so the following field is valid since parenthetical phrases are treated as comments and ignored.

```
MIME-Version: 1.0 (this is a comment)
```

The following five header fields are defined for MIME:

MIME-Version
: As noted above, this must have the value 1.0.

Content-Type
: This describes how the object within the body is to be interpreted. The default value is text/plain; charset=us-ascii, which indicates unformatted 7-bit ASCII text data (which is a message body by the RFC 822 definition).

Content-Transfer-Encoding
: This describes how the object within the body was encoded so that it could be included in the message in a mail-safe form.

Content-Description
: A plain text description of the object within the body, which is useful when the object is not readable (for example, audio data).

Content-ID
: A world-unique value specifying the content of this part of this message.

The first two of these fields are described in more detail in the following sections.

The Content-Type Field

The body of the message is described with a *Content-Type* field of the form:

```
Content-Type: type/subtype ;parameter=value ;parameter=value
```

The allowable parameters are dependent on the type and subtype. Some type/subtype pairs have no parameters, some have optional ones, some have mandatory ones and some have both. The subtype parameter *cannot* be omitted, but the whole field can, in which case the default value is text/plain.

There are seven standard content-types:

text

A single subtype is defined:

plain Unformatted text. The character set of the text may be specified with the charset parameter. The following values are permitted:

us-ascii The text consists of ASCII characters in the range 0 to 127 (decimal). This is the default (for compatibility with RFC 822).

iso-8859-x where x is in the range 1 to 9 for the different parts of the ISO-8859 standard. The text consists of ISO characters in the range 0 to 255 (decimal). All of the ISO-8859 character sets are ASCII-based with national language characters and so on in the range 128 to 255. Note that, if the text contains no characters with values above 127, the character set should be specified as us-ascii because it can be adequately represented in that character set.

Further subtypes may be added to describe other readable text formats (such as word processor formats) which contain formatting information for an application to enhance the appearance of the text, provided that the correct software is not required to determine the meaning of the text.

multipart

The message body contains multiple objects of independent data types. In each case, the body is divided into parts by lines called encapsulation boundaries. The contents of the boundary are defined with a parameter in the content-type field, for example:

```
Content-Type: multipart/mixed; boundary="1995021309105517"
```

The boundary should not appear in any of the parts of the message. It is case-sensitive and consists of 1-70 characters from a set of 75 which are known to be very robust through mail gateways, and it may not end in a space. (The example uses a 16-digit decimal timestamp.) Each encapsulation boundary consists of the boundary value prefixed by a <CRLF> sequence and two hyphens (for compatibility with RFC 934). The final boundary that marks the end of the last part also has a suffix of two hyphens. Within each part there is a MIME header, which like ordinary mail headers is terminated by the sequence <CRLF><CRLF> but may be blank. The header fields define the content of the encapsulated message.

Four subtypes are defined:

mixed The different parts are independent but are to be transmitted together. They should be presented to the recipient in the order that they appear in the mail message.

parallel This differs from the mixed subtype only in that no order is ascribed to the parts and the receiving mail program can, for example, display all of them in parallel.

alternative The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the best version to the user.

digest This is a variant on multipart/mixed where the default type/subtype is message/rfc822 (see below) instead of text/plain. It is used for the common case where multiple RFC 822 or MIME messages are transmitted together.

An example of a complex multipart message is shown in Figure 130.

```
MIME-Version: 1.0
From: Steve Hayes <steve@hayessj.bedfont.uk.ibm.com>
To:   Matthias Enders <enders@itso180.itso.ral.ibm.com>
Subject: Multipart message
Content-type: multipart/mixed; boundary="1995021309105517"

This section is called the preamble.  It is after the header but before the
first boundary.  Mail readers which understand multipart messages must
ignore this.
--1995021309105517

The first part.  There is no header, so this is text/plain with
charset=us-ascii by default.  The immediately preceding <CRLF> is part of
the <CRLF><CRLF> sequence that ends the null header.  The one at the end is
part of the next boundary, so this part consists of five lines of text with
four <CRLF>s.
--1995021309105517
Content-type: text/plain; charset=us-ascii
Comments: this header explicitly states the defaults

One line of text this time, but it ends in a line break.

--1995021309105517
Content-Type: multipart/alternative; boundary=_
Comments:     An encapsulated multipart message!

Again, this preamble is ignored.  The multipart body contains a still image
and a video image encoded in Base64.  See4.8.3.5, "Base64 Encoding" on page 204
One feature is that the character "_" which is allowed in multipart
boundaries never occurs in Base64 encoding so we can use a very simple
boundary!
--_
Content-type: text/plain
```

Figure 130 (Part 1 of 2). MIME - A Complex Multipart Example

```
This message contains images which cannot be displayed at your terminal.
This is a shame because they're very nice.

--_
Content-type: image/jpeg
Content-transfer-encoding: base64
Comments: This photograph is to be shown if the user's system cannot display
          MPEG videos.  Only part of the data is shown in this book because
          the reader is unlikely to be wearing MIME-compliant spectacles.

Qk1OAAAAAAAAAE4EAABAAAAAQAEAAPAAAAABAAgAAAAAAAAAAAAAAAAAAAAAAABAAAAAQAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAB4VjQSAAAAAAAAgAAAkgAAAJKAAKoAAACqAIAAqpIAAMHBwQDJyckA
/9uqAKpJAAD/SQAAAG0AAFVtAACqbQAA/20AAAAkAABVkgAAqiQAAP+SAAAAtgAAVbYAAKq2AAD/
<base64 data continues for another 1365 lines>
--_
Content-type: video/mpeg
Content-transfer-encoding: base64

AAABswoAeBn//+CEAAABsgAAAOgAAAG4AAAAAAAAAQAAT/////wAAAGy//8AAAEBQ/ZlIwwBGWCX
+pqMiJQDjAKywS/1NRrtXcTCLgzVQymqqHAf0sL1sMgMq4SWLCwOTYRdgyAyrhNYsLhhF3DLjAGg
BdwDXBv3yMV8/4tzrp3zsAWIGAJg1IBKTeFFI2IsgutIdfuSaAGCTsBVnWdz8afdMMAMgKgMEkPE
<base64 data continues for another 1839 lines>
--_--
That was the end of the nested multipart message.  This is the epilogue.
Like the preamble it is ignored.
--1995021309105517--
And that was the end of the main multipart message.  That's all folks!
```

Figure 130 (Part 2 of 2). MIME - A Complex Multipart Example

message

The body is an encapsulated message, or part of one. Three subtypes are defined:

rfc822 The body itself is an encapsulated message with the syntax of an RFC 822 message. It is required that at least one of From:, Subject: or Date: must be present.

Note: rfc822 refers to the syntax of the encapsulated message envelopes and does not preclude MIME messages for example.

partial This type is used to allow fragmentation of large mail items in a similar way to IP fragmentation. Because SMTP agents may impose upper limits on maximum mail sizes, it may be necessary to send large items as fragments. The intent of the message/partial mail items is that the fragmentation is transparent to the recipient. The receiving user agent should re-assemble the fragments to create a new message with identical semantics to the original. There are three parameters for the Content-Type: field:

id= A unique identifier common to all parts of the message.

number= The sequence number of this part, with the first part being numbered 1.

total= The total number of parts. This is optional on all but the last part. The last part is identified by the fact that it has the same value for the number and total parameters.

The original message is always a message according to RFC 822 rules. The first part is syntactically equivalent to a message/rfc822 message (that is the body itself contains message headers), and the subsequent parts are syntactically equivalent to text/plain messages. When re-building the message, the RFC 822 header fields are taken from the top-level message, not from the enclosed message, with the exception of those fields that cannot be copied from the inner message to the outer when fragmentation is performed (for example, the Content-Type: field).

Note: It is explicitly permitted to fragment a message/partial message further. This allows mail gateways to freely fragment messages in order to ensure that all parts are small enough to be transmitted. If this were not the case, the mail agent performing the fragmentation would have to know the smallest maximum size limit that the mail items would encounter en route to the destination.

external-body This type contains a pointer to an object that exists elsewhere. It has the syntax of the message/rfc822 type. The top-level message header defines how the external object is to be accessed, using the access-type: parameter of the Content-Type: field and a set of additional parameters that are specific to the access type. The intent is for the mail reader to be able to synchronously access the external object using the specified access type. The following access types are defined:

ftp File Transfer Protocol. The recipient will be expected to supply the necessary user ID and password. For security reasons, these are never transmitted with the message.

tftp Trivial File Transfer Protocol.

anon-ftp Anonymous FTP.

local-file The data is contained in a file accessible directly via the recipient's local file system.

mail-server The data is accessible via a mail server. Unlike the others, this access is necessarily asynchronous.

When the external object has been received, the desired message is obtained by appending the object to the message header encapsulated within the body of the message/external-body message. This encapsulated message header defines how the resulting message is to be interpreted. (It is required to have a Content-ID: and will normally have a Content-Type: field.) The encapsulated message body is not used (the real message body is elsewhere, after all) and it is therefore termed the phantom body. There is one exception to this: if the access-type is mail-server the phantom body contains the mail server commands necessary to extract the real message body. This is because mail server syntaxes vary widely so it is much simpler to use the otherwise redundant phantom body than to codify a syntax for encoding

arbitrary mail server commands as parameters on the Content-Type: field.

image

The body contains image data requiring a graphical display or some other device such as a printer to display it. Two subtypes are defined initially:

jpeg The image is in JPEG format, JFIF encoding.

gif GIF format.

video

The body contains moving image data (possibly with synchronized audio) requiring an intelligent terminal or multimedia workstation to display it. A single subtype is defined initially:

mpeg MPEG format.

audio

The body contains image data requiring a speaker and sound card (or similar hardware) to display it. A single subtype is defined initially:

basic A lowest common denominator format in the absence of any de facto standards for audio encoding. Specifically, it is single-channel 8-bit ISDN mu-law encoding at a sample rate of 8kHz.

application

This type is intended for types that do not fit into other categories, and particularly for data to be processed by an application program before being presented to the user, such as spreadsheet data. It is also intended for application programs that are intended to be processed as part of the mail reading process (for example, see the PostScript type below). This type of usage poses serious security risks unless an implementation ensures executable mail messages are run in a safe or *padded cell* environment.

Two subtypes are defined initially:

PostScript Adobe Systems PostScript (Level 1 or Level 2).

Security Issues: Although PostScript is often thought of as a format for printer data, it is a programming language and the use of a PostScript interpreter to process application/PostScript types poses serious security problems. Any mail reader that automatically interprets PostScript programs is equivalent, in principle, to one that automatically runs executable programs it receives. RFC 2045 outlines the issues involved.

octet-stream This subtype indicates general binary data consisting of 8-bit bytes. It is also the subtype that a mail reader should assume on encountering an unknown type or subtype. Any parameters are permitted, and RFC mentions two: a type= parameter to inform the recipient of the general type of the data and padding= to indicate a bit stream encoded in a byte stream. (The padding value is the number of trailing zero bits added to pad the stream to a byte boundary.)

Implementations are recommended to offer the user the option of using the data as input to a user program or of storing it in a file. (There is no standard for the default name of such a file, although

RFC 2045 does mention a “Content-Disposition:” field to be defined in a later RFC.)

Security Issues: RFC strongly recommends against an implementation executing an application/octet-stream part automatically or using it as input to a program specified in the mail header. To do so would expose the receiving system to serious security risks and could impact the integrity of any networks that the system is connected to.

Obviously, there are many types of data that do not fit into any of the subtypes above. Co-operating mail programs may, in keeping with the rules of RFC 822, use types and/or subtypes beginning with X- as private values. No other values are permitted unless they have first been registered with the Internet Assigned Numbers Authority (IANA). See RFC 2048 for more details. The intention is that few, if any, additional types will be needed, but that many subtypes will be added to the set.

The Content-Transfer-Encoding Field

As already noted, SMTP agents and mail gateways can severely constrain the contents of mail messages that can be transmitted safely. The MIME types described above list a rich set of different types of objects which can be included in mail messages and the majority of these do not fall within these constraints. Therefore, it is necessary to encode data of these types in a fashion that can be transmitted, and to decode them on receipt. RFC 2045 defines two forms of encoding that are mail safe. The reason for two forms rather than one is that it is not possible, given the small set of characters known to be mail safe, to devise a form that can both encode text data with minimal impact to the readability of the text and yet can encode binary data that consists of characters distributed randomly across all 256 byte values compactly enough to be practical.

These two encodings are used only for bodies and not for headers. Header encoding is described in 4.8.4, “Using Non-ASCII Characters in Message Headers” on page 206. The Content-Transfer-Encoding: field defines the encoding used. Although cumbersome, this field name emphasizes that the encoding is a feature of the transport process and not an intrinsic property of the object being mailed. Although there are only two encodings defined, this field can take on *five* values. (As usual, the values are case-insensitive.) Three of the values actually specify that no encoding has been done; where they differ is that they imply different reasons why this is the case. This is a subtle but important point. MIME is not restricted to SMTP as a transport agent, despite the prevalence of (broadly) SMTP-compliant mail systems on the Internet. It therefore allows a mail agent to transmit data that is not mail-safe by the standards of SMTP (that is STD 10/RFC 821). If such a mail item reaches a gateway to a more restrictive system, the encoding mechanism specified allows the gateway to decide on an item-by-item basis whether the body must be encoded to be transmitted safely.

The five encodings are:

- 7-bit (the default if the Content-Transfer-Encoding: header is omitted)
- 8-bit
- Binary
- Quoted-Printable

- Base64

These are described in the sections that follow.

7-bit Encoding

7-bit encoding means that no encoding has been done and the body consists of lines of ASCII text with a length of no more than 1000 characters. It is therefore known to be mail-safe with any mail system that *strictly* conforms with STD 10/RFC 821. This is the default, since these are the restrictions which apply to pre-MIME STD 11/RFC 822 messages.

Note: 7-bit encoding does *not* guarantee that the contents are truly mail safe for two reasons. First, gateways to EBCDIC networks have a smaller set of mail-safe characters, and secondly because of the many non-conforming SMTP implementations. The Quoted-Printable encoding is designed to overcome these difficulties for text data.

8-bit Encoding

8-bit encoding implies that lines are short enough for SMTP transport, but that there may be non-ASCII characters (that is, octets with the high-order bit set). Where SMTP agents support the SMTP Service Extension for 8-bit-MIMEtransport, described in RFC 1652, 8-bit encoding is possible. Otherwise, SMTP implementations should set the high-order bit to zero, so 8-bit encoding is not valid.

Binary Encoding

Binary encoding indicates that non-ASCII characters may be present and that the lines may be too long for SMTP transport. (That is, there may be sequences of 999 or more characters without a CRLF sequence.) There are currently no standards for the transport of unencoded binary data by mail based on the TCP/IP protocol stack, so the only case where it is valid to use binary encoding in a MIME message sent on the Internet or other TCP/IP based network is in the header of an external-body part (see the message/external-body type above). Binary encoding would be valid if MIME were used in conjunction with other mail transport mechanisms, or with a hypothetical SMTP Service Extension that did support long lines.

4.8.3.4 Quoted-Printable Encoding

This is the first of the two real encodings and it is intended to leave text files largely readable in their encoded form.

- It represents non-mail safe characters by the hexadecimal representation of their ASCII characters.
- It introduces reversible (soft) line breaks to keep all lines in the message to a length of 76 characters or less.

Quoted-Printable encoding uses the equal sign as a quote character to indicate both of these cases. It has five rules which are summarized as follows:

1. Any character except one that is part of a new line sequence (that is, a X'0D0A' sequence on a text file) can be represented by =XX, where XX are two uppercase hexadecimal digits. If none of the other rules apply, the character must be represented like this.
2. Any character in the range X'21' to X'7E' except X'3D' (“=”) can be represented as the ASCII character.

3. ASCII TAB (X'09') and SPACE (X'20') can be represented as the ASCII character except when it is the last character on the line.
4. A line break must be represented by a <CRLF> sequence (X'0D0A'). When encoding binary data, X'0D0A' is not a line break and should be coded, according to rule 1, as =0D=0A.
5. Encoded lines cannot be longer than 76 characters (excluding the <CRLF>). If a line is longer than this, a soft line break must be inserted at or before column 75. A soft line break is the sequence =<CRLF> (X'3D0D0A').

This scheme is a compromise between readability, efficiency and robustness. Since rules 1 and 2 use the phrase “may be encoded,” implementations have a fair degree of latitude on how many characters are quoted. If as few characters are quoted as possible within the scope of the rules, then the encoding will work with well-behaved ASCII SMTP agents. Adding the following set of ASCII characters to the list of those to be quoted is adequate for well-behaved EBCDIC gateways:

```
! " # $ @ [ \ ] ^ ` { ¦ } ~
```

For total robustness, it is better to quote *every* character except for the 73-character set known to be invariant across all gateways, that is the letters and digits (A-Z, a-z and 0-9) and the following 11 characters:

```
' ( ) + , - . / : = ?
```

Note: This invariant list does not even include the SPACE character. For practical purposes, when encoding text files, only a SPACE at the end of a line should be quoted. Otherwise readability is severely impacted.

Base64 Encoding

This encoding is intended for data that does not consist mainly of text characters. Quoted-Printable replaces each non-text character with a 3-byte sequence which is grossly inefficient for binary data. Base64 encoding works by treating the input stream as a bit stream, regrouping the bits into shorter bytes, padding these short bytes to 8 bits and then translating these bytes to characters that are known to be mail-safe. As noted in the previous section, there are only 73 safe characters, so the maximum byte length usable is 6 bits which can be represented by 64 unique characters (hence the name Base64). Since the input and output are both byte streams, the encoding has to be done in groups of 24 bits (that is 3 input bytes and 4 output bytes). The process can be seen as follows:

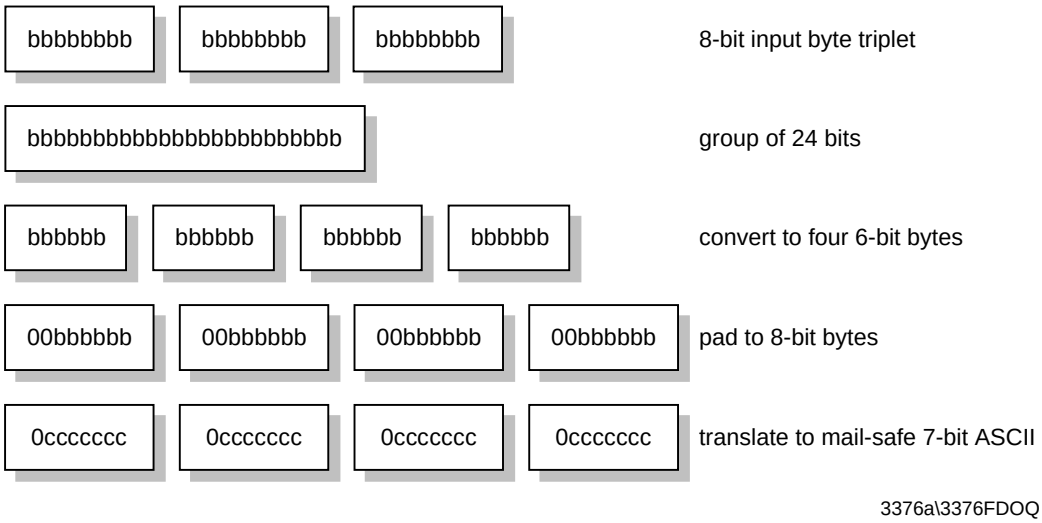

Figure 131. MIME - Base64 Encoding. How 3 input bytes are converted to 4 output bytes in the Base64 encoding scheme.

The translate table used is called the *Base64 Alphabet*.

Base64 value	ASCII char.	Base64 value	ASCII char.	Base64 value	ASCII char.	Base64 value	ASCII char.
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Figure 132. MIME - The Base64 Alphabet

One additional character (the = character) is needed for padding. Because the input is a byte stream that is encoded in 24-bit groups it will be short by zero, 8 or 16 bits, as will the output. If the output is of the correct length, no padding is needed. If the output is 8 bits short, this corresponds to an output quartet of two complete bytes, a short byte and a missing byte. The short byte is padded with two low-order zero bits. The missing byte is replaced with an = character. If the output is 16 bits short, this corresponds to an output quartet of one complete byte, a short byte and two missing bytes. The short byte is padded with 6 low-order zero bits. The two missing bytes are replaced with an = character. If zero characters (that is As) were used, the receiving agent would not be able to tell when decoding the input stream if trailing X'00' characters in the last or last two positions of the output stream were data or padding. With pad characters, the number of “=”s (0, 1 or 2) gives the length of the input stream modulo 3 (0, 2 or 1 respectively).

Conversion between Encodings

The Base64 encoding can be freely translated to and from the binary encoding without ambiguity since both treat the data as an octet-stream. This is also true for the conversion from Quoted-Printable to either of the other two (in the case of the Quoted-Printable to Binary conversion the process can be viewed as involving an intermediate binary encoding) by converting the quoted character sequences to their 8-bit form, deleting the soft line breaks and replacing hard line breaks with <CRLF> sequences. This is not strictly true of the reverse process since Quoted-Printable is actually a record-based system. There is a semantic difference between a hard line break and an imbedded =0D=0A sequence. (For example when decoding Quoted-Printable on a EBCDIC record-based system such as VM, hard line breaks map to record boundaries but =0D=0A sequences map to X'0D25' sequences.)

Multiple Encodings

MIME does *not* allow nested encodings. Any Content-Type that recursively includes other Content-Type fields (notably the multipart and message types) cannot use a Content-Transfer-Encoding other than 7-bit, 8-bit or binary. All encodings must be done at the innermost level. The purpose of this restriction is to simplify the operation of user mail agents. If nested encodings are not permitted, the structure of the entire message is always visible to the mail agent without the need to decode the outer layer(s) of the message.

This simplification for user mail agents has a price: complexity for gateways. Because a user agent can specify an encoding of 8-bit or binary, a gateway to a network where these encodings are not safe must encode the message before passing it to the second network. The obvious solution, to simply encode the message body and to change the Content-Transfer-Encoding: field, is not allowed for the multipart or message types since it would violate the restriction described above. The gateway must therefore correctly parse the message into its components and re-encode the innermost parts as necessary.

There is one further restriction: messages of type message/partial must *always* have 7-bit encoding. (8-bit and binary are also disallowed.) The reason for this is that if a gateway needs to re-encode a message, it requires the entire message to do so, but the parts of the message may not all be available together. (Parts may be transmitted serially because the gateway is incapable of storing the entire message at once or they may even be routed independently via different gateways.) Therefore message/partial body parts must be mail safe across lowest common denominator networks; that is, they must be 7-bit encoded.

Using Non-ASCII Characters in Message Headers

All of the mechanisms above refer exclusively to bodies and not to headers. The contents of message headers must still be coded in US-ASCII. For header fields that include human-readable text, this is not adequate for languages other than English. A mechanism to include national language characters is defined by the second part of MIME (RFC 2047). This mechanism differs from the Quoted-Printable encoding, which would be used in a message body for the following reasons:

- The format of message headers is strictly codified by RFC 822, so the encoding used by MIME for header fields must work within a narrower set of constraints than that used for bodies.

- Message relaying programs frequently change message headers, for example re-ordering header fields, deleting some fields but not others, re-ordering mailboxes within lists or spilling fields at different positions than the original message.
- Some message handling programs do not correctly handle some of the more arcane features of RFC 822 (such as the use of the \ character to quote special characters such as < and >).

The approach used by MIME is to reserve improbable sequences of legal ASCII characters that are not syntactically important in RFC 822 for use with this protocol. Words in header fields that need national characters are replaced by *encoded words* which have the form:

```
=?charset?encoding?word?=
```

where:

charset The value allowed for the charset parameter used with text/plain MIME type, that is: “us-ascii” or “iso-8859-1” through “iso-8859-9.”

encoding B or Q. B is identical to the Base64 encoding used in message bodies. Q is similar to the Quoted-Printable encoding but uses _ to represent X'20' (ASCII SPACE).[10] Q encoding requires the encoding of _ characters and does not allow line breaks. Any printable ASCII character other than _, = and SPACE may be left unquoted within an encoded word unless it would be syntactically meaningful when the header field is parsed according to RFC 822.

charset and encoding are both case-insensitive.

word A string of ASCII text characters other than SPACE, which conforms to the rules of the encoding given.

An encoded word must have no imbedded white space characters (SPACE or TAB), can be up to 75 characters long, and cannot be on a line that is greater than 76 characters long (excluding the <CRLF>). These rules ensure that gateways will not fold encoded words in the middle of the word. Encoded words can generally be used in the human-readable parts of header fields. For example, if a mailbox is specified in the following form:

```
The Octopus <octopus@garden.under.the.sea>
```

an encoded word could be used in the The Octopus section but not in the address part between the < and the>). RFC 2047 specifies precisely where encoded words can be used with reference to the syntax of RFC 822.

References

A detailed description of MIME can be found in the following RFCs:

- *RFC 2045 – MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*
- *RFC 2046 – MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types*

---

[10] The underscore character is not strictly mail-safe, but it is used because the use of any other character to indicate a SPACE would seriously hamper readability.

- *RFC 2047 – MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*
- *RFC 2048 – MIME (Multipurpose Internet Mail Extensions) Part Four: Registration Procedures*
- *RFC 2049 – MIME (Multipurpose Internet Mail Extensions) Part Five: Conformance Criteria and Examples*
- *RFC 2156 — MIXER (MIME Internet X.400 Enhanced Relay): Mapping between X.400 and RFC 822/MIME*

Post Office Protocol (POP)

The Post Office Protocol, Version 3 is a *standard protocol* with STD number 53. Its status is *elective*. It is described in RFC 1939. The older Post Office Protocol Version 2 is an *historic protocol* with a status of *not recommended*.

The Post Office Protocol is an electronic mail protocol with both client (sender/receiver) and server (storage) functions. POP3 supports basic functions (download and delete) for electronic mail retrieval. More advanced functions are supported by IMAP4 (see 4.10, “Internet Message Access Protocol Version 4 (IMAP4)” on page 209).

POP3 clients establish a TCP connection to the server using port 110. When the connection is established, the POP3 server sends a greeting message to the client. The session then enters the *authentication state*. The client must send its identification to the server while the session is in this state. If the server verifies the ID successfully, the session enters the *transaction state*. In this state, the client can access the mailbox. When the client sends the `QUIT` command, the session enters the *Update state* and the connection is then closed.

POP3 Commands and Responses

POP3 commands consist of a keyword and possibly one or more arguments following the keyword. Keywords are three or four characters long and separated by one space character from arguments. Each argument must be up to 40 characters long.

The server sends a response to the command that was issued by the client. This response must be up to 512 characters and begin with a status indicator which shows whether the reply is positive or negative. These indicators are (+OK) or (-ERR). The server must send these indicators in upper case.

Here are the three states for a POP3 session:

Authorization state

In this state, the client sends identification to the server. This is implemented in two ways. More information on authentication is described in RFC 1734.

1. Using USER and PASS commands.
2. Using APOP command.

Transaction state

In this state, the client can issue commands for listing, retrieving and deleting. Please note that the deleting action is not taken in this state. The client must send the QUIT command and then the server goes to the update state.

Update state

In this state, the server updates the mailbox according to the commands received from the client in the transaction state and the TCP connection ends. If the connection is broken for any reason before quit command is received from the client, the server does not enter the update state. Thus, the server will not update anything.

Important POP3 Commands

USER `name` User name for authentication.

PASS `password` Password for authentication.

STAT To get the number of messages and total size of the messages.

LIST [`msg`] If a message number is specified, the size of this mail is listed (if it exists). If not, all messages will be listed with the message sizes.

RETR `msg` This command sends the whole message to the client.

DELE `msg` This command deletes the specified message.

NOOP The server does not do anything, just sends a positive response.

RSET This command cancels previous delete requests if they exist.

QUIT If entered in the authorization state, it merely ends the TCP connection. If entered in the transaction state, it first updates the mailbox (deletes any messages requested previously) and then ends the TCP connection.

References

A detailed description of the Post Office Protocol can be found in the following RFCs:

- *RFC 937 — Post Office Protocol – Version 2*
- *RFC 1939 — Post Office Protocol – Version 3*