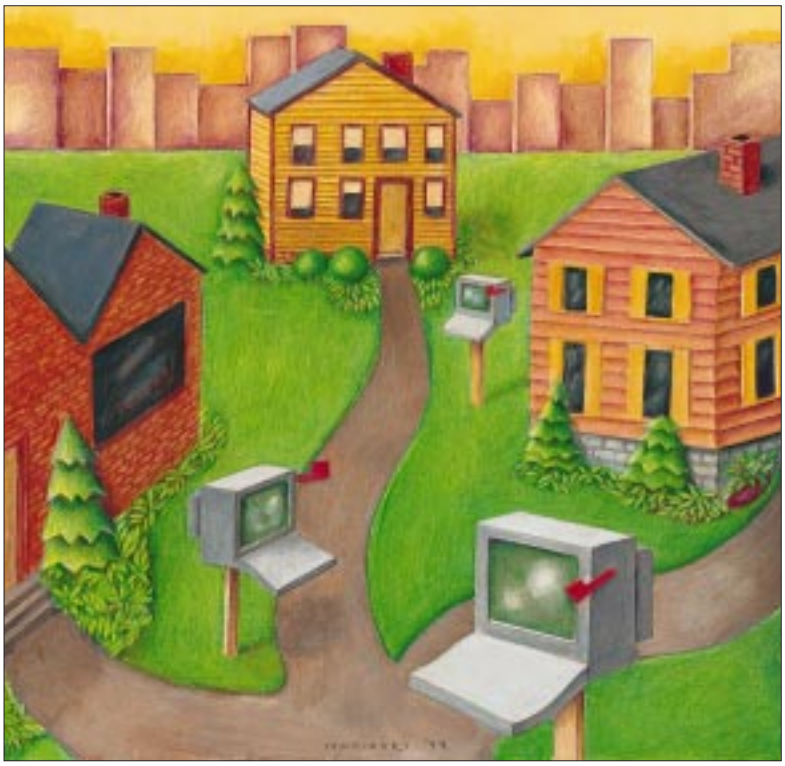
*by Peter Collinson,* **Hillside Systems**



JANE MARINSKY

# MIME

We think of the Internet as a recent phenomenon, but email interchange on the Internet uses a protocol that was defined in 1982 and hasn't changed substantially since. The email protocol was defined by RFC 822, written by David Crocker (then at the University of Delaware). An RFC is a "Request For Comments" and is the way Internet protocols are documented. RFC 822 revised RFC 733, which had been used on the ARPANET since 1977. The "one-step back, one-step on" correlation between 733 and 822 is simply an accident of history. Actually, there is now a series of "accepted" standards, and mail is documented by STD 11. However, I guess that the magic 822 number will stick around for some time to come.

RFC 822 has been very successful. It was very forward-looking, defining the domain-based address structure that we all use today. RFC 822 concentrates on the envelope around the message that is sent, it discusses the headers and their format, but says very little about the format of the message body.

However, the title of RFC 822 is significant: "Standard for the format of ARPA Internet text messages." It's the word "text" in this title that is relevant. The standard was created at a time when computing was largely text-based. Email use spread along with the rise of UNIX, whose basic file format is text and whose tools are designed to process text. Strictly, the word "text" in this case means the 7-bit ASCII character set.

The use of text is also defined by RFC 821, which specifies the Simple Mail Transport Protocol (SMTP) used to move the mail from machine to machine over the Internet. Strictly, SMTP is constrained to use 7-bit characters formed into lines of text that should be no more than 1,000 characters long. Each line is terminated by an end-of-line indicator, which is formed from a Carriage-Return character followed by a Line-Feed.

Actually, many UNIX implementations of email don't enforce the 7-bit restrictions and secretly support 8-bit characters. This has provided some comfort to friends of mine whose languages cannot be written properly in the ASCII character set. At least they can send mail in their own language, which, of course, is "text" to them, but that text needs to be represented by an 8-bit character set. Such mail would probably work between machines in Europe, but there is no guarantee that a gateway somewhere wouldn't strip the mail back to a 7-bit set, creating gobbledegook.

There are still problems with inserting some of my friends' names in the headers of email that strictly use a 7-bit coding. Many people are actually unaware of the problem and just put their names into the headers using the correct character set for their language. Because I run my system in the Latin-1 locale (or more properly, ISO-8859-1), I can see every e-*acute* or c-*cedilla*. However, if you are in the United States, you probably cannot. Of course, access to Latin-1 is of no use to the Japanese or

the Russians, whose languages require completely different character sets.

Using text as a transport medium causes problems when you want to send a binary file to a friend. I suspect that we are now living in a world where we need to send binary files more and more.

Initially, binary files on UNIX were mostly compiled programs, but you could never make assumptions about file formats. The original `tar` program could be used to encapsulate several files into one archive file, which turned out to be a binary file even if the files that were contained in the archive were simply text. The header that was used between each file was designed to be read easily by the program and contained binary values. The folks at Berkeley changed this to the text header that was later enshrined in the POSIX standard.

If you want to send a binary file to someone in safety, then you need to encapsulate the data into a text form. Probably the most widespread method of turning a binary file into text was to use `uuencode` to create the text stream and `uudecode` to unscramble it. These programs were widely available on all UNIX systems as part of the UUCP suite that was used to form the Usenet network. If you are going to use some kind of encapsulation method, then you need to know that your recipient can decode the information.

The first line of a `uuencoded` file gives the target file name and its associated permissions. The remaining data contains lines of characters. Each line starts with a byte count, encoded as a printable character starting from "A" in the ASCII coding sequence. The value "1" is encoded as "A", "2" as "B," "3" as "C" and so on. Using the ASCII sequence with "A" translating to "1," the space character maps onto zero. So when decoding the data and obtaining the byte count for the line, you'll read the first character, subtract the value of the space character from it and the result is a numeric byte-count value. The text that forms the remainder of the line follows the byte count. Three 8-bit bytes from the source are mapped into four printable characters, only six bits of each input byte is used. Each character is mapped into a visible value using the ASCII coding sequence described above.

In general, using `uuencode` to encapsulate a binary file requires human intervention. Typically, the `uuencoded` part of the mail message would be surrounded by "cut here" text asking the receiver to save the message to a file and then run `uudecode` on it.

## Multimedia Mail

At the start of the '90s, Nathaniel S. Borenstein, then at Bellcore, started to look into the problems caused by the limitations of using text as the basic mail transportation mechanism. He wanted to allow people to interchange pictures, video and audio in various formats but still work within the confines of RFC 822. After all, email systems were a way of transporting files from one person to another, the fact that these files were mostly text was an artificial restriction. If the RFC 822 standard could be adapted to carry multimedia mail, then it would have zero impact on the mechanisms for transporting mail that were in widespread use.

However, attempting to introduce multimedia mail capabilities was still a daunting task because, even at that time, there was a huge number of different mail reading programs (the email world calls these *user agents*). Early on, he realized that attempting to disseminate a brand-new, wonderful multimedia mail reading and composing program would not work because people would stick with their existing systems. He needed to generate a system that would bolt onto existing programs in a painless manner. The pain for the user must be negligible. The pain for the implementor would be greater and any system needed to present easy ways for the implementors of user agents to pick up on the new facilities. Borenstein's system, known as `metamail`, became the basis for alterations to several user agents permitting them to handle multimedia mail cleanly. A cornerstone of the approach was the `mailcap` file, which allowed programs to have some centralized way on a specific system of finding what Netscape Communications Corp. later termed a "helper application"–a program on the local machine that could handle image display or play an audio clip. This meant that programmers of user agents didn't need to become audio or graphics specialists, they could leverage other people's work.

However, for the capabilities to become widely used, something was needed in the RFC 822 protocol that was aimed at this new breed of enhanced user agent. Something that would tell the user agent what type of formatted file the mail contained so that it could be decoded and passed to the relevant helper application. Borenstein and Ned Freed (Freed was then at Innosoft International Inc.) wrote RFC 1341, which defined three new header lines that were to be added to the 822 mail format. Incidentally, RFC 1341 has been superseded by RFC 2045, and you should go there if you are interested in delving into this subject in greater depth.

Being prudent people, the first new header line was designed for future expansion and supplies a protocol version number. The `Mime-Version:` field should be set to 1.0 to indicate that the mail follows the conventions described in the RFC.

The second header line defines the type of content that is to be found in the mail. An earlier RFC (RFC 1049 by M. Sirbu of Carnegie Mellon University) had proposed that mail should contain a `Content-Type:` field, but chose to use argument keywords that were aimed at specific applications or devices. RFC 1341 made the specification considerably more general, allowing the argument field to give a broad type of application, followed by a subtype. There are intentionally only a small list of broad application types: `application`, designed to permit binary data to be sent; `audio` and `image`, which are self-evident; `message`, defining a mail message; `text`, for text of some form or other; and `multipart`, which allows one mail message to encapsulate several other types of message.

The idea of having a broad type followed by a subtype allows mail readers to easily reject message parts with which they cannot deal. So if a mail reader cannot deal with images of a specific type, it still knows that a message flagged as `image/something` contains binary information that cannot be displayed as text. However, the user agent may elect to present other options, perhaps allowing the user to store the

image as a file somewhere on their local disk.

The third new header line tackles the problem that RFC 822 mail is constrained to printable text. The `Content-Transfer-Encoding:` header gives a standard method of encoding the embedded information. The sender of the mail can automatically turn their data into length-limited lines of text, and have it be reconstituted by the recipient's mail reader without any special knowledge of the encoding.

Of course, transformation of the original data may not be needed. If you've typed in an ASCII message, which contains lines of less than 998 bytes (two bytes are needed for the end-of-line indicator), then there is no need to alter your message. The sender can use `Content-Transfer-Encoding:` values of `7-bit`, `8-bit` or `binary` to indicate that no transformation has been made to the original data.

Should encoding be needed, the RFC provides two styles of encoding. The first, more lightweight, encoding is called `quoted-printable` and is designed to be applied to data that is "nearly ASCII text." The text perhaps contains a few characters like the acute accent in "Café," or perhaps it has very long lines, which need to be broken to ensure that they are not truncated by any restricted gateway.

The `quoted-printable` encoding method attempts to maintain message readability for naive mail readers while allowing correct translation of characters for any mail reader that can decode the message. To this end, all the 7-bit ASCII characters are passed through unchanged. However, any character can be encoded as an equals sign followed by two characters that represent the character form of a hexadecimal value (so the equals sign is encoded as `=3D`). The equals sign can also be used at the end of a line to introduce an end-of-line indicator that can be removed by the reading program. There are some other conversions; see the RFC for the full gory details.

When there are a few characters that need to be hidden from the transport mechanism `quoted-printable` encoding works well, imposing very little size overhead. But it would prove to be very heavyweight if used for pure binary files containing, say, audio or images. The RFC defines `base64` encoding to cope with binary files. The mechanism is very similar to `uuencode`. However, some care has been taken with the encoding of binary values into characters to make that character subset portable into both ASCII and EBCDIC. The encoding/decoding mechanism uses a table rather than the pure ASCII sequence, ensuring that the final representation has good portability across machines.

## Content Types

As I said above, there is a very small number of basic broad categories used in the `Content-Type:` field in the mail header. The standards define a very small number of subtypes, and there is now a registration system that allows the registration of new ones. The registration database is quite large now. The types you will mostly see in your mail are:

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

This is the default setting for what might be termed "a normal piece" of mail. The `Content-Type:` is `text` and the subtype is `plain`, indicating that the text contains no inherent formatting information. Parameters follow the main argument, separated by a semi-colon. The `charset` parameter defines the character set that should be used to render the text. It's not just `ascii` because the term ASCII has come to represent a multitude of different character sets (RFC 2047 goes on at great length about this). I should have used US-ASCII throughout this article to emphasize that I was talking about a specific standard, but I thought it would confuse you. The `us-ascii` character set points at a specific ANSI standard–the "7-bit American Standard Code for Information Interchange," ANSI X3.4-1986–and so is precise. Alternatives to US-ASCII point to the various ISO standards that define character sets, so mail from my machine says

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

which is the character set that I call Latin-1.

Recently, browser manufacturers have been pushing HTML as a basic mail interchange standard, and you will see

```
Content-Type: text/html; charset="iso-8859-1"
```

encapsulating an HTML document in the mail message. In December, I interviewed Eric Allman (creator of `sendmail` and founder of Sendmail Inc.) for the U.K. magazine, *EXE*, for which I write regularly, and he said that HTML mail is liked by businesses "because they can find out when you read the mail since their Web server can be hit when you pull a graphic. Also, they can tailor their mail message so that you see what they want you to see when you read the mail, rather defining its contents when they send it."

However, these browsers don't just send HTML hoping that the reader will be able to deal with it; they use the multipart capabilities of Multipurpose Internet Mail Extensions (MIME) to send alternative views of the same mail message. The main header to the mail will contain something like the following:

```
MIME-Version: 1.0
Content-Type: multipart/alternative;
      boundary="++++Boundary_between_messages++++"
```

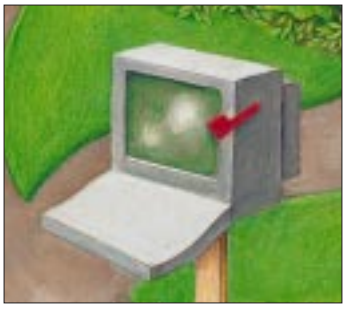
This tells the mail reader that the message actually contains several alternate views of the same message. Each part will be separated by

```
--++++Boundary_between_messages++++
```

and will start with a new `Content-Type:` and, possibly, a new `Content-Type-Encoding:`. It is assumed that the `boundary` text will not occur "naturally" in the body of the message. Notice that the `boundary` field gains two hyphen characters marking the start of the line. Similarly, the last `boundary` field in a file has two hyphens appended to the end of its text.

The RFC requests that the least complex type of message should appear first, followed by messages of increasing formatting richness. The idea here is that the message is somewhat more viewer-friendly. If the reader is using a naive mail agent, then they should be able to easily find and see the message as text.

Also, any text "outside the boundary" is ignored. So text before the first `boundary` marker, or after the last, can be used to contain comments. You'll typically see a message like, "This is a multipart MIME message," appearing before the first `boundary`. The comment is aimed at users of naive mail readers, telling them what this odd-looking message actually is.



Another possible multipart type allows for attachments of different types to be added to the message:

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

I tend to use this when sending Microsoft Corp. Word documents to people. The first part of the message will be some explanatory text and the second will be the Word document, which is basically a binary format, and will generally be encoded using the `base64` technique.

## MIME Mail on Your Sun

In general, the PC world has embraced MIME because it needed to find some way of sending binary email, because many PC applications deal with what is essentially a binary file format. Sadly, Sun Microsystems Inc. has been lagging woefully behind these developments. To be fair, I haven't had a chance to look at the Solaris 7 release that has found its way to my office, so things may have changed. The alternatives provided on earlier Solaris releases are pretty poor.

I've been using email for eons. At one point, some years back, I migrated to the Rand MH mail system, and I've been using it ever since. These days, I use Brent Welch's excellent `exmh` interface that sits in front of the command-line programs that are normally used to drive MH. MIME email is handled by `exmh`, and it will also cope with some simple HTML email messages with its own built-in HTML browser. One of the benefits of using MH is that you can still use the basic MH commands when you are far from home and can only log in with `telnet` to read email from a GUI-free zone. I did this once from a public library in Fort Bragg, CA. I wasn't desperate to read my email, but it seemed fun and sounded like a challenge to be able to do it from a small town on the Californian coast. Typing `telnet:` and my machine name into Netscape gave me an interactive login to my computer in the United Kingdom, and MH allowed me to read my messages and reply to them.

However, I suspect that getting and installing all the pieces for `exmh` from scratch is not for the fainthearted. First, you have to install MH and then Tcl/Tk to support `exmh`. Installing `exmh` is actually the least of your worries, it just slots in. Again, to be fair, installing MH and Tcl/Tk is simply a matter of getting the code and compiling it, and I would expect there to be no problems.

Another alternative is to install Netscape Communicator and use its inbuilt email facilities. Again, I haven't tried doing this. I was put off by the size of the file that was required.

## Reading and Software

The sources for this article are all RFCs, actually. I haven't come across a book that explains MIME and its influence on email or the Web. MIME was picked up and used by the designers of HTML to provide data typing for the protocol, which, to me, seems a highly commended case of not reinventing that wheel. RFCs are available using anonymous FTP to `ftp.isi.edi`, although I used a local mirror site to access the information.

You can get `exmh` from `http://www.scriptics.com`, and you will also find the most recent Tcl/Tk sources there as well.

---

***Peter Collinson*** *runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever... He writes, teaches, consults and programs using Solaris running on a SPARCstation 2. Email:* `pc@cpg.com`.