A User Agent Configuration Mechanism
For Multimedia Mail Format Information

Status of This Memo

Abstract

   This memo suggests a file format to be used to inform multiple mail
   reading user agent programs about the locally-installed facilities
   for handling mail in various formats. The mechanism is explicitly
   designed to work with mail systems based Internet mail as defined by
   RFC's 821 (STD 10), 822 (STD 11), 934, 1049 (STD 11), 1113, and the
   Multipurpose Internet Mail Extensions, known as MIME. However, with
   some extensions it could probably be made to work for X.400-based
   mail systems as well. The format and mechanism are proposed in a
   manner that is generally operating-system independent. However,
   certain implementation details will inevitably reflect operating
   system differences, some of which will have to be handled in a
   uniform manner for each operating system. This memo makes such
   situations explicit, and, in an appendix, suggests a standard
   behavior under the UNIX operating system.

Introduction

   The electronic mail world is in the midst of a transition from
   single-part text-only mail to multi-part, multi-media mail. In
   support of this transition, various extensions to RFC 821 and RFC 822
   have been proposed and/or adopted, notably including MIME [RFC-1521].
   Various parties have demonstrated extremely high-functionality
   multimedia mail, but the problem of mail interchange between
   different user agents has been severe. In general, only text
   messages have been shared between user agents that were not
   explicitly designed to work together. This limitation is not
   compatible with a smooth transition to a multi-media mail world.

   One approach to this transition is to modify diverse sets of mail
   reading user agents so that, when they need to display mail of an
   unfamiliar (non-text) type, they consult an external file for
   information on how to display that file. That file might say, for

example, that if the content-type of a message is "foo" it can be
displayed to the user via the "displayfoo" program.

This approach means that, with a one-time modification, a wide
variety of mail reading programs can be given the ability to display
a wide variety of types of message. Moreover, extending the set of
media types supported at a site becomes a simple matter of installing
a binary and adding a single line to a configuration file. Crucial
to this scheme, however, is that all of the user agents agree on a
common representation and source for the configuration file. This
memo proposes such a common representation.

Location of Configuration Information

Each user agent must clearly obtain the configuration information
from a common location, if the same information is to be used to
configure all user agents. However, individual users should be able
to override or augment a site's configuration. The configuration
information should therefore be obtained from a designated set of
locations. The overall configuration will be obtained through the
virtual concatenation of several individual configuration files known
as mailcap files. The configuration information will be obtained
from the FIRST matching entry in a mailcap file, where "matching"
depends on both a matching content-type specification, an entry
containing sufficient information for the purposes of the application
doing the searching, and the success of any test in the "test="
field, if present.

The precise location of the mailcap files is operating-system
dependent. A standard location for UNIX is specified in Appendix A.

Overall Format of a Mailcap File

Each mailcap file consists of a set of entries that describe the
proper handling of one media type at the local site.

For example, one line might tell how to display a message in Group
III fax format. A mailcap file consists of a sequence of such
individual entries, separated by newlines (according to the operating
system's newline conventions). Blank lines and lines that start with
the "#" character (ASCII 35) are considered comments, and are
ignored. Long entries may be continued on multiple lines if each
non-terminal line ends with a backslash character ('\', ASCII 92), in
which case the multiple lines are to be treated as a single mailcap
entry. Note that for such "continued" lines, the backslash must be
the last character on the line to be continued.

   Thus the overall format of a mailcap file is given, in the modified
   BNF of RFC 822, as:

         Mailcap-File = *Mailcap-Line

         Mailcap-Line = Comment / Mailcap-Entry

         Comment = NEWLINE / "#" *CHAR NEWLINE

         NEWLINE = <newline as defined by OS convention>

   Note that the above specification implies that comments must appear
   on lines all to themselves, with a "#" character as the first
   character on each comment line.

Format of a Mailcap Entry

   Each mailcap entry consists of a number of fields, separated by
   semi-colons. The first two fields are required, and must occur in
   the specified order. The remaining fields are optional, and may
   appear in any order.

   The first field is the content-type, which indicates the type of data
   this mailcap entry describes how to handle. It is to be matched
   against the type/subtype specification in the "Content-Type" header
   field of an Internet mail message. If the subtype is specified as
   "*", it is intended to match all subtypes of the named content-type.

   The second field, view-command, is a specification of how the message
   or body part can be viewed at the local site. Although the syntax of
   this field is fully specified, the semantics of program execution are
   necessarily somewhat operating system dependent. UNIX semantics are
   given in Appendix A.

   The optional fields, which may be given in any order, are as follows:

   -- The "compose" field may be used to specify a program that can be
      used to compose a new body or body part in the given format. Its
      intended use is to support mail composing agents that support the
      composition of multiple types of mail using external composing
      agents. As with the view-command, the semantics of program
      execution are operating system dependent, with UNIX semantics
      specified in Appendix A. The result of the composing program may
      be data that is not yet suitable for mail transport -- that is, a
      Content-Transfer-Encoding may need to be applied to the data.

   -- The "composetyped" field is similar to the "compose" field, but is
      to be used when the composing program needs to specify the

Content-type header field to be applied to the composed data. The
"compose" field is simpler, and is preferred for use with existing
(non-mail-oriented) programs for composing data in a given format.
The "composetyped" field is necessary when the Content-type
information must include auxilliary parameters, and the
composition program must then know enough about mail formats to
produce output that includes the mail type information.

-- The "edit" field may be used to specify a program that can be used
   to edit a body or body part in the given format. In many cases,
   it may be identical in content to the "compose" field, and shares
   the operating-system dependent semantics for program execution.

-- The "print" field may be used to specify a program that can be
   used to print a message or body part in the given format. As with
   the view-command, the semantics of program execution are operating
   system dependent, with UNIX semantics specified in Appendix A.

-- The "test" field may be used to test some external condition
   (e.g., the machine architecture, or the window system in use) to
   determine whether or not the mailcap line applies. It specifies a
   program to be run to test some condition. The semantics of
   execution and of the value returned by the test program are
   operating system dependent, with UNIX semantics specified in
   Appendix A. If the test fails, a subsequent mailcap entry should
   be sought. Multiple test fields are not permitted -- since a test
   can call a program, it can already be arbitrarily complex.

-- The "needsterminal" field indicates that the view-command must be
   run on an interactive terminal. This is needed to inform window-
   oriented user agents that an interactive terminal is needed. (The
   decision is not left exclusively to the view-command because in
   some circumstances it may not be possible for such programs to
   tell whether or not they are on interactive terminals.) The
   needsterminal command should be assumed to apply to the compose
   and edit commands, too, if they exist. Note that this is NOT a
   test -- it is a requirement for the environment in which the
   program will be executed, and should typically cause the creation
   of a terminal window when not executed on either a real terminal
   or a terminal window.

-- The "copiousoutput" field indicates that the output from the
   view-command will be an extended stream of output, and is to be
   interpreted as advice to the UA (User Agent mail-reading program)
   that the output should be either paged or made scrollable. Note
   that it is probably a mistake if needsterminal and copiousoutput
   are both specified.

-- The "description" field simply provides a textual description,
   optionally quoted, that describes the type of data, to be used
   optionally by mail readers that wish to describe the data before
   offering to display it.

-- The "textualnewlines" field, if set to any non-zero value,
   indicates that this type of data is line-oriented and that, if
   encoded in base64, all newlines should be converted to canonical
   form (CRLF) before encoding, and will be in that form after
   decoding. In general, this field is needed only if there is
   line-oriented data of some type other than text/* or non-line-
   oriented data that is a subtype of text.

-- The "x11-bitmap" field names a file, in X11 bitmap (xbm) format,
   which points to an appropriate icon to be used to visually denote
   the presence of this kind of data.

-- The "nametemplate" field gives a file name format, in which %s
   will be replaced by a short unique string to give the name of the
   temporary file to be passed to the viewing command. This is only
   expected to be relevant in environments where filename extensions
   are meaningful, e.g., one coulld specify that a GIF file being
   passed to a gif viewer should have a name eding in ".gif" by using
   "nametemplate=%s.gif".

Any other fields beginning with "x-" may be included for local or
mailer-specific extensions of this format. Implementations should
simply ignore all such unrecognized fields to permit such extensions,
some of which might be standardized in a future version of this
document.

Some of the fields above, such as "needsterminal", apply to the
actions of the view-command, edit-command, and compose-command,
alike. In some unusual cases, this may not be desirable, but
differentiation can be accomplished via separate mailcap entries,
taking advantage of the fact that subsequent mailcap entries are
searched if an earlier mailcap entry does not provide enough
information:

    application/postscript; ps-to-terminal %s;\ needsterminal
    application/postscript; ps-to-terminal %s; \compose=idraw %s

In RFC 822 modified BNF, the following grammar describes a mailcap
entry:

```
        Mailcap-Entry = typefield ; view-command
                            [";" 1#field]

        typefield = propertype / implicit-wild

        propertype = type "/" wildsubtype

        implicitwild = type

        wildsubtype = subtype / "*"

        view-command = mtext

        mtext = *mchar

        mchar = schar / qchar

        schar = * <any CHAR except ";","\", and CTLS>

        qchar = "\" CHAR ; may quote any char

        field = flag / namedfield

        namedfield = fieldname "=" mtext

        flag = "needsterminal"   ; All these literals are to
              / "copiousoutput"   ; be interpreted as
              / x-token           ; case-insensitive

        fieldname =    / "compose"      ;Also all of these
                       / "composetyped" ;are case-insensitive.
                       / "print"
                       / "edit"
                       / "test"
                       / "x11-bitmap"
                       / "textualnewlines"
                       / "description"
                       / x-token
```

   Note that "type", "subtype", and "x-token" are defined in MIME. Note
   also that while the definition of "schar" includes the percent sign,
   "%", this character has a special meaning in at least the UNIX
   semantics, and will therefore need to be quoted as a qchar to be used
   literally.

Acknowledgements

   The author wishes to thank Malcolm Bjorn Gillies, Dan Heller, Olle
   Jaernefors, Keith Moore, Luc Rooijakkers, and the other members of
   the IETF task force on mail extensions for their comments on earlier
   versions of this draft. If other acknowledgements were neglected,
   please let me know, as it was surely accidental.

Security Considerations

   Security issues are not discussed in this memo. However, the use of
   the mechanisms described in this memo can make it easier for
   implementations to slip into the kind of security problems discussed
   in the MIME document. Implementors and mailcap administrators should
   be aware of these security considerations, and in particular should
   exercise caution in the choice of programs to be listed in a mailcap
   file for automatic execution.

Author's Address

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

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

Appendix A: Implementation Details for UNIX

   Although this memo fully specifies a syntax for "mailcap" files, the
   semantics of the mailcap file are of necessity operating-system
   dependent in four respects. In order to clarify the intent, and to
   promote a standard usage, this appendix proposes a UNIX semantics for
   these four cases. If a mailcap mechanism is implemented on non-UNIX
   systems, similar semantic decisions should be made and published.

Location of the Mailcap File(s)

   For UNIX, a path search of mailcap files is specified. The default
   path search is specified as including at least the following:

   $HOME/.mailcap:/etc/mailcap:/usr/etc/mailcap:/usr/local/etc/mailcap

   However, this path may itself be overridden by a path specified by
   the MAILCAPS environment variable.

Semantics of executable commands

   Several portions of a mailcap entry specify commands to be executed.
   In particular, the mandatory second fie ld, the view-command, takes a
   command to be executed, as do the optional print, edit, test, and
   compose fields.

   On a UNIX system, such commands will each be a full shell command
   line, including the path name for a program and its arguments.
   (Because of differences in shells and the implementation and behavior
   of the same shell from one system to another, it is specified that
   the command line be intended as input to the Bourne shell, i.e., that
   it is implicitly preceded by "/bin/sh -c " on the command line.)

   The two characters "%s", if used, will be replaced by the name of a
   file for the actual mail body data. In the case of the edit adn
   view-command, the body part will be passed to this command as
   standard input unless one or more instances of "%s" appear in the
   view-command, in which case %s will be replaced by the name of a file
   containing the body part, a file which may have to be created before
   the view-command program is executed. (Such files cannot be presumed
   to continue to exist after the view-command program exits. Thus a
   view-command that wishes to exit and continue processing in the
   background should take care to save the data first.) In the case of
   the compose and composetyped commands, %s should be replaced by the
   name of a file to which the composed data should be written by the
   programs named in the compose or composedtyped commands. Thus, the
   calling program will look in that file later in order to retrieve the
   composed data. If %s does not appear in the compose or composetyped

commands, then the composed data will be assumed to be written by the
composing programs to standard output.

Furthermore, any occurrence of "%t" will be replaced by the content-
type and subtype specification. (That is, if the content-type is
"text/plain", then %t will be replaced by "text/plain".) A literal %
character may be quoted as \%. Finally, named parameters from the
Content-type field may be placed in the command execution line using
"%{" followed by the parameter name and a closing "}" character. The
entire parameter should appear as a single command line argument,
regardless of embedded spaces. Thus, if the message has a Content-
type line of:

        Content-type: multipart/mixed; boundary=42

and the mailcap file has a line of:

        multipart/*; /usr/local/bin/showmulti \
          %t %{boundary}

then the equivalent of the following command should be
executed:

        /usr/local/bin/showmulti multipart/mixed 42

If the content-type is "multipart" (any subtype), then the two
characters "%n" will be replaced by an integer giving the number of
sub-parts within the multipart entity. Also, the two characters "%F"
will be replaced by a set of arguments, twice as many arguments as
the number of sub-parts, consisting of alternating content-types and
file names for each part in turn. Thus if multipart entity has three
parts, "%F" will be replaced by the equivalent of "content-type1
file-name1 content-type2 file-name2 content-type3 file-name3".

Semantics of the "test" field

The "test" field specifies a program to be used to test whether or
not the current mailcap line applies. This can be used, for example,
to have a mailcap line that only applies if the X window system is
running, or if the user is running on a SPARCstation with a
/dev/audio. The value of the "test" field is a program to run to
test such a condition. The precise program to run and arguments to
give it are determined as specified in the previous section. The
test program should return an exit code of zero if the condition is
true, and a non-zero code otherwise.

Semantics of the "compose" field

   On UNIX, the composing program is expected to produce a data stream
   for such a body part as its standard output. The program will be
   executed with the command line arguments determined as specified
   above. The data returned via its standard output will be given a
   Content-Type field that has no supplementary parameters. For
   example, the following mailcap entry:

         audio/basic; /usr/local/bin/showaudio %t
           compose = /usr/local/bin/recordaudio

   would result in tagging the data composed by the "recordaudio"
   program as:

         Content-Type: audio/basic

   If this is unacceptable -- for example, in the case of multipart mail
   a "boundary" parameter is required -- then the "compose" field cannot
   be used. Instead, the "composetyped" field should be used in the
   mailcap file.

Semantics of the "composetyped" field

   The "composetyped" filed is much like the "compose" field, except
   that it names a composition program that produces, not raw data, but
   data that includes a MIME-conformant type specification. The program
   will be executed with the command line arguments determined as
   specified above. The data returned via its standard output must
   begin with a Content-Type header, followed optionally by other
   Content-* headers, and then by a blank line and the data. For
   example, the following mailcap entry:

         multipart/mixed; /usr/local/bin/showmulti %t \
           %{boundary}; \
           composetyped = /usr/local/bin/makemulti

   would result in executing the "makemulti" program, which would be
   expected to begin its output with a line of the form:

         Content-Type: multipart/mixed; boundary=foobar

   Note that a composition program need not encode binary data in base64
   or quoted-printable. It remains the responsibility of the software
   calling the composition program to encode such data as necessary.
   However, if a composing program does encode data, which is not
   encouraged, it should announce that fact using a Content-Transfer-
   Encoding header in the standard manner defined by MIME. Because such

encodings must be announced by such a header, they are an option only
for composetyped programs, not for compose programs.

Appendix B: Sample Mailcap File

   The following is an example of a mailcap file for UNIX that
   demonstrates most of the syntax above. It contains explanatory
   comments where necessary.

```
# Mailcap file for Bellcore lab 214.
#
# The next line sends "richtext" to the richtext
program
text/richtext; richtext %s; copiousoutput
#
# Next, basic u-law audio
audio/*; showaudio; test=/usr/local/bin/hasaudio
#
# Next, use the xview program to handle several image
formats
image/*; xview %s; test=/usr/local/bin/RunningX
#
# The ATOMICMAIL interpreter uses curses, so needs a
terminal
application/atomicmail; /usr/local/bin/atomicmail %s; \
    needsterminal
#
# The next line handles Andrew format,
#   if ez and ezview are installed
x-be2; /usr/andrew/bin/ezview %s; \
   print=/usr/andrew/bin/ezprint %s ; \
   compose=/usr/andrew/bin/ez -d %s \;
   edit=/usr/andrew/bin/ez -d %s; \;
   copiousoutput
#
# The next silly example demonstrates the use of
quoting
application/*; echo "This is \"%t\" but \
    is 50 \% Greek to me" \; cat %s; copiousoutput
```

Appendix C: A Note on Format Translation

   It has been suggested that another function of a mailcap-like
   mechanism might be to specify the locally available tools for
   document format translation. For example, the file could designate a
   program for translating from format A to format B, another for
   translating from format B to format C, and finally a mechanism for
   displaying format C. Although this mechanism would be somewhat
   richer than the current mailcap file, and might conceivably also have
   utility at the message transport layer, it significantly complicates
   the processing effort necessary for a user agent that simply wants to
   display a message in format A. Using the current, simpler, mailcap
   scheme, a single line could tell such a user agent to display A-
   format mail using a pipeline of translators and the C-format viewer.
   This memo resists the temptation to complicate the necessary
   processing for a user agent to accomplish this task. Using the
   mailcap format defined here, it is only necessary to find the correct
   single line in a mailcap file, and to execute the command given in
   that line.

References

   [RFC-822] Crocker, D., "Standard for the format of ARPA Internet
   text messages", STD 11, RFC 822, UDEL, August 1982.

   [RFC-1521] Borenstein, N., and N. Freed, "MIME (Multipurpose
   Internet Mail Extensions) Part One: Mechanisms for Specifying and
   Describing the Format of Internet Message Bodies", RFC 1521,
   Bellcore, Innosoft, September 1993.