                    The text/enriched MIME Content-type

Status of this Memo

Abstract

   MIME [RFC-1341, RFC-1521] defines a format and general framework for
   the representation of a wide variety of data types in Internet mail.
   This document defines one particular type of MIME data, the
   text/enriched type, a refinement of the "text/richtext" type defined
   in RFC 1341. The text/enriched MIME type is intended to facilitate
   the wider interoperation of simple enriched text across a wide
   variety of hardware and software platforms.

The Text/enriched MIME type

   In order to promote the wider interoperability of simple formatted
   text, this document defines an extremely simple subtype of the MIME
   content-type "text", the "text/enriched" subtype. This subtype was
   designed to meet the following criteria:

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

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

        3. If the character set in use is ASCII or an 8- bit ASCII
        superset, then the raw form of the data must be readable enough
        to be largely unobjectionable in the event that it is displayed
        on the screen of the user of a non-MIME-conformant mail reader.

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

This document defines a new MIME content-type, "text/enriched". The
content-type line for this type may have one optional parameter, the
"charset" parameter, with the same values permitted for the
"text/plain" MIME content-type.

The syntax of "text/enriched" is very simple. It represents text in
a single character set -- US-ASCII by default, although a different
character set can be specified by the use of the "charset" parameter.
(The semantics of text/enriched in non-ASCII character sets are
discussed later in this document.) All characters represent
themselves, with the exception of the "<" character (ASCII 60), which
is used to mark the beginning of a formatting command. Formatting
instructions consist of formatting commands surrounded by angle
brackets ("<>", ASCII 60 and 62). Each formatting command may be no
more than 60 characters in length, all in US-ASCII, restricted to the
alphanumeric and hyphen ("-") characters. Formatting commands may be
preceded by a solidus ("/", ASCII 47), making them negations, and
such negations must always exist to balance the initial opening
commands. Thus, if the formatting command "<bold>" appears at some
point, there must later be a "</bold>" to balance it. (NOTE: The 60
character limit on formatting commands does NOT include the "<", ">",
or "/" characters that might be attached to such commands.)

Formatting commands are always case-insensitive. That is, "bold" and
"BoLd" are equivalent in effect, if not in good taste.

Beyond tokens delimited by "<" and ">", there are two other special
processing rules. First, a literal less-than sign ("<") can be
represented by a sequence of two such characters, "<<". Second, line
breaks (CRLF pairs in standard network representation) are handled
specially. In particular, isolated CRLF pairs are translated into a
single SPACE character. Sequences of N consecutive CRLF pairs,
however, are translated into N-1 actual line breaks. This permits
long lines of data to be represented in a natural- looking manner
despite the frequency of line-wrapping in Internet mailers. When
preparing the data for mail transport, isolated line breaks should be
inserted wherever necessary to keep each line shorter than 80
characters. When preparing such data for presentation to the user,
isolated line breaks should be replaced by a single SPACE character,
and N consecutive CRLF pairs should be presented to the user as N-1
line breaks.

Thus text/enriched data that looks like this:

                    This is
                    a single
                    line

                    This is the
                    next line.

                    This is the
                    next paragraph.

should be displayed by a text/enriched interpreter as follows:

                    This is a single line
                    This is the next line.

                    This is the next paragraph.

The formatting commands, not all of which will be implemented by all
implementations, are described in the following sections.

Formatting Commands

The text/enriched formatting commands all begin with <commandname>
and end with </commandname>, affecting the formatting of the text
between those two tokens. The commands are described here, grouped
according to type.

Font-Alteration Commands

The following formatting commands are intended to alter the font in
which text is displayed, but not to alter the indentation or
justification state of the text:

     Bold -- causes the affected text to be in a bold font. Nested
          bold commands have the same effect as a single bold
          command.

     Italic -- causes the affected text to be in an italic font.
          Nested italic commands have the same effect as a single
          italic command.

     Fixed -- causes the affected text to be in a fixed width font.
          Nested fixed commands have the same effect as a single
          fixed command.

Smaller -- causes the affected text to be in a smaller font.
    It is recommended that the font size be changed by two
    points, but other amounts may be more appropriate in some
    environments. Nested smaller commands produce ever-
    smaller fonts, to the limits of the implementation's
    capacity to reasonably display them, after which further
    smaller commands have no incremental effect.

Bigger -- causes the affected text to be in a bigger font. It
    is recommended that the font size be changed by two
    points, but other amounts may be more appropriate in some
    environments. Nested bigger commands produce ever-bigger
    fonts, to the limits of the implementation's capacity to
    reasonably display them, after which further bigger
    commands have no incremental effect.

Underline -- causes the affected text to be underlined. Nested
    underline commands have the same effect as a single
    underline command.

While the "bigger" and "smaller" operators are effectively inverses,
it is not recommended, for example, that "<smaller>" be used to end
the effect of "<bigger>". This is properly done with "</bigger>".

Justification Commands

Initially, text/enriched text is intended to be displayed fully-
justified with appropriate fill, kerning, and letter-tracking as
suits the capabilities of the receiving user agent software. Actual
line width is left to the discretion of the receiver, which is
expected to fold lines intelligently (preferring soft line breaks) to
the best of its ability.

The following commands alter that state. Each of these commands
force a line break before and after the formatting command if there
is not otherwise a line break. For example, if one of these commands
occurs anywhere other than the beginning of a line of text as
presented, a new line is begun.

Center -- causes the affected text to be centered.

FlushLeft -- causes the affected text to be left-justified with a
    ragged right margin.

FlushRight -- causes the affected text to be right-justified with
    a ragged left margin.

The center, flushleft, and flushright commands are mutually
exclusive, and, when nested, the inner command takes precedence.

Note that for some non-ASCII character sets, full justification may
be inappropriate. In these cases, a user agent may choose not to
justify such data.

Indentation Commands

Initially, text/enriched text is displayed using the maximum
available margins. Two formatting commands may be used to affect the
margins.

     Indent -- causes the running left margin to be moved to the
          right. The recommended indentation change is the width of
          four characters, but this may differ among
          implementations.

     IndentRight -- causes the running right margin to be moved to
          the left. The recommended indentation change is the width
          of four characters, but this may differ among
          implementations.

A line break is NOT forced by a change of the margin, to permit the
description of "hanging" text. Thus for example the following text:

Now <indent> is the time for all good horses to come to the aid of
their stable, assuming that </indent> any stable is really stable.

would be displayed in a 40-character-wide window as follows:

Now is the time for all good horses to
     come to the aid of their stable,
     assuming that any stable is
really stable.

Miscellaneous Commands

     Excerpt -- causes the affected text to be interpreted as a
          textual excerpt from another source, probably a message
          being responded to. Typically this will be displayed
          using indentation and an alternate font, or by indenting
          lines and preceding them with "> ", but such decisions are
          up to the implementation. (Note that this is the only
          truly declarative markup construct in text/enriched, and
          as such doesn't fit very well with the other facilities,
          but it describes a type of markup that is very commonly
          used in email and has no procedural analogue.) Note that

as with the justification commands, the excerpt command
implicitly begins and ends with a line break if one is not
already there.

Verbatim -- causes the affected text to be displayed without
filling, justification, any interpretation of embedded
formatting commands, or the usual special rules for CRLF
handling. Note, however, that the end token </verbatim>
must still be recognized.

Nofill -- causes the affected text to be displayed without
filling or justification, and hence without any special
handling of CRLFs, but with all remaining text/enriched
features continuing to apply.

Param -- Marks the affected text as command parameters, to be
interpreted or ignored by the text/enriched interpreter,
but NOT to be shown to the reader.

Note that while the absence of a quoting mechanism makes it slightly
challenging to include the literal string "<verbatim>" inside of a
verbatim environment, it can be done by breaking up the verbatim
segment into two verbatim segments as follows:

```
<verbatim>
...slightly challenging to include the literal string
"</</verbatim><verbatim>verbatim>" inside of a verbatim
environment...
</verbatim>
```

Note that the above example demonstrates that it is not desirable for
an implementation to break lines between tokens. In particular,
there should not be a line break inserted between the "</verbatim>"
and the "<verbatim>" that follows it.

Balancing and Nesting of Formatting Commands

Pairs of formatting commands must be properly balanced and nested.
Thus, a proper way to describe text in bold italics is:

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

or, alternately,

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

but, in particular, the following is illegal
text/enriched:

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

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

Unrecognized formatting commands

Implementations must regard any unrecognized formatting command as
"no-op" commands, that is, as commands having no effect, thus
facilitating future extensions to "text/enriched". Private
extensions may be defined using formatting commands that begin with
"X-", by analogy to Internet mail header field names.

In order to formally define extended commands, a new Internet
document should be published.

"White Space" in text/enriched Data

No special behavior is required for the SPACE or TAB (HT) character.
It is recommended, however, that, at least when fixed-width fonts are
in use, the common semantics of the TAB (HT) character should be
observed, namely that it moves to the next column position that is a
multiple of 8. (In other words, if a TAB (HT) occurs in column n,
where the leftmost column is column 0, then that TAB (HT) should be
replaced by 8-(n mod 8) SPACE characters.) It should also be noted
that some mail gateways are notorious for losing (or, less commonly,
adding) white space at the end of lines, so reliance on SPACE or TAB
characters at the end of a line is not recommended.

Initial State of a text/enriched interpreter

Text/enriched is assumed to begin with filled, fully justified text
in a variable-width font in a normal typeface and a size that is
average for the current display and user. The left and right margins
are assumed to be maximal, that is, at the leftmost and rightmost
acceptable positions.

Non-ASCII character sets

If the character set specified by the charset parameter on the
Content-type line is anything other than "US-ASCII", this means that
the text being described by text/enriched formatting commands is in a
non-ASCII character set. However, the commands themselves are still
the same ASCII commands that are defined in this document. This
creates an ambiguity only with reference to the "<" character, the
octet with numeric value 60. In single byte character sets, such as
the ISO-8859 family, this is not a problem; the octet 60 can be
quoted by including it twice, just as for ASCII. The problem is more
complicated, however, in the case of multi-byte character sets, where
the octet 60 might appear at any point in the byte sequence for any
of several characters.

In practice, however, most multibyte character sets address this
problem internally. For example, the ISO-2022 family of character
sets can switch back into ASCII at any moment. Therefore it is
specified that, before text/enriched formatting commands, the
prevailing character set should be "switched back" into ASCII, and
that only those characters which would be interpreted as "<" in plain
text should be interpreted as token delimiters in text/enriched.

The question of what to do for hypothetical future character sets
that do NOT subsume ASCII is not addressed in this memo.

Minimal text/enriched conformance

A minimal text/enriched implementation is one that simply recognizes
the beginning and ending of "verbatim" environments and, outside of
them, converts "<<" to "<", removes everything between a <param>
command and the next balancing </param> command, removes all other
formatting commands (all text enclosed in angle brackets), converts
any series of n CRLFs to n-1 CRLFs, and converts any lone CRLF pairs
to SPACE.

Notes for Implementors

It is recognized that implementors of future mail systems will want
rich text functionality far beyond that currently defined for
text/enriched. The intent of text/enriched is to provide a common
format for expressing that functionality in a form in which much of
it, at least, will be understood by interoperating software. Thus,
in particular, software with a richer notion of formatted text than
text/enriched can still use text/enriched as its basic
representation, but can extend it with new formatting commands and by
hiding information specific to that software system in text/enriched
<param> constructs. As such systems evolve, it is expected that the

definition of text/enriched will be further refined by future
published specifications, but text/enriched as defined here provides
a platform on which evolutionary refinements can be based.

An expected common way that sophisticated mail programs will generate
text/enriched data is as part of a multipart/alternative construct.
For example, a mail agent that can generate enriched mail in ODA
format can generate that mail in a more widely interoperable form by
generating both text/enriched and ODA versions of the same data,
e.g.:

        Content-type: multipart/alternative; boundary=foo

        --foo
        Content-type: text/enriched

        [text/enriched version of data]
        --foo
        Content-type: application/oda

        [ODA version of data]
        --foo--

If such a message is read using a MIME-conformant mail reader that
understands ODA, the ODA version will be displayed; otherwise, the
text/enriched version will be shown.

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

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

Extensions to text/enriched

It is expected that various mail system authors will desire
extensions to text/enriched. The simple syntax of text/enriched, and
the specification that unrecognized formatting commands should simply
be ignored, are intend to promote such extensions.

Beyond simply defining new formatting commands, however, it may
sometimes be necessary to define formatting commands that can take
arguments. This is the intended use of the <param> construct. In
particular, software that wished to extend text/enriched to include
colored text might define an "x-color" environment which always began
with a color name parameter, to indicate the desired color for the
affected text.

An Example

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

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Content-type: text/enriched

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

to the aid of their

<x-color><param>red</param>beloved</x-color>country.
<verbatim>
By the way, I think that <smaller>
should
REALLY be called
<tinier>
and that I am always right.
-- the end
</verbatim>
```

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

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

beloved country.
By the way, I think that <smaller>
should
REALLY be called
<tinier>
and that I am always right.
-- the end
```

   where the word "beloved" would be in red on a color display if the
   receiving software implemented the "x-color" extension.

Security Considerations

   Security issues are not discussed in this memo, as the mechanism
   raises no security issues.

Author's Address

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

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

   Phone: +1 201 829 4270
   Fax: +1 201 829 5963
   EMail: nsb@bellcore.com

Acknowledgements

   This document reflects the input of many contributors, readers, and
   implementors of the original MIME specification, RFC 1341. This memo
   also reflects particular contributions and comments from Terry
   Crowley and Rhys Weatherley.

Appendix A -- A Simple enriched-to-plain Translator in C

   One of the major goals in the design of the text/enriched subtype of
   the text Content-Type is to make formatted text so simple that even
   text-only mailers will implement enriched-to-plain-text translators,
   thus increasing the likelihood that multifont text will become "safe"
   to use very widely. To demonstrate this simplicity, what follows is
   a simple C program that converts text/enriched input into plain text
   output. Note that the local newline convention (the single character
   represented by "\n") is assumed by this program, but that special
   CRLF handling might be necessary on some systems.

```
                 #include <stdio.h>
                 #include <ctype.h>

                 main() {
                     int c, i, paramct=0, newlinect=0, verbatim=0,
                 nofill=0;
                     char token[62], *p;

                     while ((c=getc(stdin)) != EOF) {
                         if (c == '<') {
                             if (verbatim != 0) {
                                 for (i=0, p=token; (*p++ = getc(stdin))
                 != EOF
                                     && !lc2strncmp(token, "/verbatim>",
                 i+1) && i<9; i++) {}
                                 if (i==9) {
                                     verbatim = 0;
                                 } else {
                                     *p = '\0';
                                     putc('<', stdout);
                                     fputs(token, stdout);
                                 }
                                 continue;
                             } else {
                                 newlinect=0;
                                 c = getc(stdin);
                                 if (c == '<') {
                                     if (paramct <= 0) putc(c, stdout);
                                 } else {
                                     ungetc(c, stdin);
                                     for (i=0, p=token; (c=getc(stdin))
                 != EOF && c != '>'; i++) {
                                         if (i < sizeof(token)-1) *p++ =
                 isupper(c) ? tolower(c) : c;
                                     }
```

```
                                    *p = '\0';
                                    if (c == EOF) break;
                                    if (strcmp(token, "param") == 0)
                                         paramct++;
                                    else if (strcmp(token, "verbatim")
== 0)
                                         verbatim = 1;
                                    else if (strcmp(token, "nofill") ==
0)
                                         nofill++;
                                    else if (strcmp(token, "/param") ==
0)
                                         paramct--;
                                    else if (strcmp(token, "/nofill")
== 0)

                                         nofill--;
                           }
                      }
          } else {
              if (paramct > 0)
                ; /* ignore params */
                 else if (c == '\n' && verbatim == 0 &&
nofill <= 0)
                      if (++newlinect > 1) {
                          putc(c, stdout);
                      } else {
                          putc(' ', stdout);
                      }
                 else {
                      newlinect = 0;
                      putc(c, stdout);
                 }
          }
         }
         /* The following line is only needed with line-
buffering */
         putc('\n', stdout);
         exit(0);
}

lc2strncmp(s1, s2, len)
char *s1, *s2;
int len;
{
     if (!s1 || !s2) return (-1);
     while (*s1 && *s2 && len > 0) {
      if (*s1 != *s2 && (tolower(*s1) != *s2)) return(-
```

```
                1);
                    ++s1; ++s2; --len;
                }
                if (len <= 0) return(0);
                return((*s1 == *s2) ? 0 : -1);
            }
```

   It should be noted that one can do considerably better than this in
   displaying text/enriched data on a dumb terminal. In particular, one
   can replace font information such as "bold" with textual emphasis
   (like *this* or _T_H_I_S_). One can also properly handle the
   text/enriched formatting commands regarding indentation,
   justification, and others. However, the above program is all that is
   necessary in order to present text/enriched on a dumb terminal
   without showing the user any formatting artifacts.

Appendix B -- Differences from RFC 1341 text/richtext

   Text/enriched is a clarification, simplification, and refinement of
   the type defined as text/richtext in RFC 1341. For the benefit of
   those who are already familiar with text/richtext, or for those who
   want to exploit the similarities to be able to display text/richtext
   data with their text/enriched software, the differences between the
   two are summarized here. Note, however, that text/enriched is
   intended to make text/richtext obsolete, so it is not recommended
   that new software generate text/richtext.

   0. The name "richtext" was changed to "enriched", both to
   differentiate the two versions and because "richtext" created
   widespread confusion with Microsoft's Rich Text Format (RTF).

   1. Clarifications. Many things were ambiguous or unspecified in the
   text/richtext definition, particularly the initial state and the
   semantics of richtext with multibyte character sets. However, such
   differences are OPERATIONALLY irrelevant, since the clarifications
   offered in this document are at least reasonable interpretations of
   the text/richtext specification.

   2. Newline semantics have changed. In text/richtext, all CRLFs were
   mapped to spaces, and line breaks were indicated by "<nl>". This has
   been replaced by the "n-1" rule for CRLFs.

   3. The representation of a literal "<" character was "<lt>" in
   text/richtext, but is "<<" in text/enriched.

   4. The "verbatim" and "nofill" commands did not exist in
   text/richtext.

5. The "param" command did not exist in text/richtext.

6. The following commands from text/richtext have been REMOVED from
text/enriched: <COMMENT>, <OUTDENT>, <OUTDENTRIGHT>, <SAMEPAGE>,
<SUBSCRIPT>, <SUPERSCRIPT>, <HEADING>, <FOOTING>, <ISO-8859-[1-9]>,
<US-ASCII>, <PARAGRAPH>, <SIGNATURE>, <NO-OP>, <LT>, <NL>, and <NP>.

7. All claims of SGML compatibility have been dropped. However,
with the possible exceptions of the new semantics for CRLF and "<<"
can be implemented, text/enriched should be no less SGML-friendly
than text/richtext was.

8. In text/richtext, there were three commands (<NL>, <NP>, and
<LT>) that did not use balanced closing delimiters. Since all of
these have been eliminated, there are NO exceptions to the
nesting/balancing rules in text/enriched.

9. The limit on the size of formatting tokens has been increased
from 40 to 60 characters.

References

[RFC-1341] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet
Mail Extensions): Mechanisms for Specifying and Describing the Format
of Internet Message Bodies", RFC 1341, Bellcore, Innosoft, June 1992.

[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, September 1993.