NETED: A Common Editor for the ARPA Network

BACKGROUND

At the recent Resource Sharing Workshop, there was a somewhat
surprising degree of consensus on what I had anticipated would the
least popular aspect of the my "Unified User-Level Protocol" proposal:
A number of the attendees agreed without argument that it would be
a good thing to have "the same" context editor available on all
Servers -- where "the same" refers, of course, to the user interface.
We even agreed that "NETED" seemed to be a plausible common name. In
view of the fact that the rest of the proposal didn't seem to capture
anybody's imagination, though, it seemed to be a useful notion to
separate out the common editor and make it the subject of a
stand-alone proposal.

My resolve to come up with the following was further strengthened at
the the organizing meeting of the Network User Interest Group, which
followed the Workshop. Being primarily concerned with user issues,
this group was downright enthusiastic about the prospect of a common
editor. Indeed, this proposal has been reviewed by the group and is
endorsed by it.

REASONS

The need for a common editor might well be obvious to many readers.
They are encouraged to skip this section, which is for the benefit of
those who don't already see the light.

In the first place, it's almost axiomatic that to use a time-sharing
system, you have to be able to create files (/"datasets"/"segments").
Even if you're only using the Network to send "mail", you'd still like
to be able to create a file separately, so as to be able to edit it
before sending. And if you want to write a program -- or even make a
"runoff" source file -- you simply must be able to use some editor
command on the system at hand.

Unfortunately, there are even more editors than there are systems;
and each one has it own conventions and peculiarities. So "Network
users" (who use several Servers, as opposed to those who use the
Network only to access a particular system all the time) are faced
with the unpleasant chore of developing several sets of incompatible
reflexes if they want to get along. This can certainly be done. It
has been by a number of members of the Network Working Group.

The real kicker, however, comes when we consider the needs of those
users -- who are coming into the Network community in ever-increasing
numbers -- who are not professional programmers. They just want to
get some work done, "on the Net" (that is, irrespective of which
operating system they might be talking to). They are likely to be
appalled rather than amused by having to learn a dozen ways of getting
to first base. Therefore, it seems clear than not only do we need a
common editor, but we also need a simple common editor.

CHOICES

Simplicity is not the only criterion for rejecting the apparently
"obvious" choice of either TECO or QED. (That it is a strong factor
is indicated by the old test of "Consider explaining it to a naive
secretary -- now consider explaining it to a corporation president.")
Perhaps even worse is the problem of "dialects". That is, features
vary across implementations, and settling on a common set of features
(or dialect) is likely to be a very hard task, for programmers tend to
get very fond of their familiar goodies. Besides, both TECO and QED
have their own strong (/fanatic) advocates, who's probably never be
willing to settle for the other one. Further, not every system has
both, and implementing the other is a fairly large job even if the NWG
could agree on which (and how much).

At any rate, the difficulties seem overwhelming when it comes to
choosing a high-powered editor as the common editor. Therefore, I
tried to think of a nice low-powered editor, and it suddenly occurred
to me that I not only knew of one, but it was even fairly well
documented (!). The editor in question is known on Multics as "eds"
(the same member of the "ed" family of editors which started on
CTSS), a line-oriented context editor (no "regular expressions", but
also no line numbers). It is used as an extended example of
programming in the Multics environment in Chapter 4 of the Multics
Programmers' Manual, which gives an annotated PL/I listing of the
actual working program. It is simple to learn and should be quite
easy to implement, PL/I version serves as a detailed model with only
equivalent system calls and choice of language to worry about. I urge
its adoption as the common Network editor, to be known on all
participating Servers as "NETED" and/or "neted".

DOCUMENTATION

In view of the fact that if "eds"/NETED is adopted only perhaps a
dozen members of the NWG will actually have to implement one, it seems
wasteful to distributed some 30 pages of the MPM to everyone --
especially since most of the parties concerned have access to an MPM
already. (Another problem solved by not including it here is that of
whether I'd be violating copyright by doing so.) The exact reference
is pp. 24-54 of Chapter 4 of Part I of the Multics Programmer's
Manual.

Anybody who needs a copy can let me know. Although the emphasis in
the document is, naturally enough, on the Multics-specific aspects, I
believe that the listing is clear enough to serve as a model to
implementors without any great difficulty. If we do get to the
implementation stage, I'll be glad to try to explain any non-obvious
system calls, either individually or in a follow-up memo. But even
though we "initiate" where you "open", or we " call los_$read_ptr"
where you "IOT TTY" (or something), it shouldn't cause much trouble.
For that matter, some implementers might prefer to ignore the existing
program and simply work from the function specifications (below).

LIMITATIONS

It became abundantly clear during the course of the review of this
document by the User Interest Group that the limitations of NETED must
be acknowledged (even insisted upon) and explained here. In the first
place, it must be emphasized that it is not being proposed as "THE"
Network editor. Rather, it is an insistently simple-minded editor for
two major reasons: 1) it is meant for use mainly by non-professional
programmers, and 2) more important still, it is meant to be extremely
easy to implement. Therefore, it seems far more important to go with
the published program, with all its warts, than to specify the
addition of new, undebugged features. The idea is to make it
implementable in man-days by an average to average-plus programmer
instead of in man-weeks by a superstar programmer.

In the second place, the very act of adding new features is fraught
with peril. To take some examples from the comments I received during
the review phase: In the first draft, I inadvertently failed to
document the mechanism by which the ability to "go backwards" (i.e.,
to reverse the direction of the current-line pointer described below)
is actuated. Several reviewers argued strongly for the inclusion of
such a mechanism; but, not knowing it was already "in" the code I
argued -- successfully -- for leaving it out, on the grounds that we
should stick to what's in the existing code, which is known to work as
published. Even what to call such a new request would have been
debatable -- should it be "-" and become the only non-alphabetic name?
should it be "b" for "bottom"? should "n" (for "next") become "+"?
And so on. Although this particular issue turned out be a false
alarm, I've left it in to emphasize the sort of pitfalls we can get
into by haggling over particular "features". Another familiar feature
is some sort of "read" request so that the file name need not be
specified as an argument to the command. Then, of course, one would
also want a "create" or "make" request. And perhaps a file delete
request? It keeps going on and on. The point, I think, is that if we
allow ourselves to go into "tinker mode" we could spend as many years
striving to achieve consensus on what features to add as we've spent
on Telnet or FTP ... and still not please everyone. Therefore, I urge
the NWG to accept the contention that having a working model to use as

a pattern is more important than any particular additional features
(even though I myself find "=" for "what's the current line's
number?" annoying to live without).

RESPONSES

For the benefit of those who don't want to plow through the functional
spec, this seems to be a good spot to indicate what appropriate
responses to this proposal would be. Ideally, I'd like to hear from a
responsible system programmer at, say, TENEX, CCN, UCSD, UCSB,
AMES-67, one of the DEC 10/50 Hosts, and from any of the experimental
Servers who happen to be interested, that they think it's a fine idea
and why don't I log in next week to try their NETEDs. Next most
desirable would be agreement in principle followed by specific
inquiries about "eds". I would hope that haggling over specific
features wouldn't occur (as we're not trying to do a definitive
editor, just an easy, commonly implemented one based on an existing
implementation), but unfortunately I can't legislate such haggles out
of existence. At the very least, I'd hope to either hear or see
reasoned arguments why it's not worth doing. As usual, phone, mail
"mail" ("map.cn" in sndmsg, or "map cn" in "mail" on Multics) or RFC's
are the assumed media for responding.

USAGE

(The following is intended to serve double-duty, as both a functional
spec now and -- with the addition of some examples -- a "users'
manual" later. So if it seems to "tutorial", I'm sorry. And if it
doesn't seem tutorial enough -- assuming the addition of examples --
please let me know.)

As is typical of "context editors," the NETED command is used both for
creating new files and for altering already existing files -- where
"files" are named collections of character encoded data in the storage
hierarchy of a time-sharing system. Consequently, NETED operates in
two distinct "modes" -- called "input mode" and "edit mode".

When NETED is used to create a file (that is, when it is invoked from
command level with an argument which specifies the name of a file
which does not already exist in the user's "working directory"), it is
automatically in input mode. It will announce this fact by outputting
a message along the lines of "File soandso not found. Input." Until
you take explicit action to leave input mode, everything you type will
go into the specified file. (Actually, it goes into a "working copy"
of the file, and into the real file only when you indicate a desire to
have that happen.) To leave input mode, type a line consisting of only
a period and the appropriate new-line character: ".<NL>", where <NL>
is whatever it takes to cause a Telnet New-Line to be generated from
your terminal

After leaving input mode, you are in edit mode. Here, you may issue
various "requests" which will allow you to alter the contents of the
(working) file, re-enter input mode if you wish, and eventually cause
the file to be stored. Note that edit mode is entered automatically
if the argument you supplied to NETED specified an existing file.
Regardless of how it was entered, being in edit mode is confirmed by
NETED's outputting a message of the form "Edit". Editing is performed
relative to (conceptual) pointer which specifies the current line, and
many requests pertain to either moving the pointer or changing the
contents of the current line. (When edit mode is entered from input
mode, the pointer is at the last line input; when entered from command
level, the pointer is at the "top" of the file.)

NETED's edit mode requests follow, in order intended to be helpful.
Two important reminders: the requests may only be issued from edit
mode, and each one "is a line" (i.e., terminates in a new line /
carriage return / linefeed is appropriate to the User Telnet being
employed). SYNTAX NOTE: If the request takes an argument, there must
be at least one space (blank) between request's name and the argument.

1. n m

For unsigned m, the n(ext) request causes the pointer to be moved
"down" m lines. If m is negative, the pointer is moved "up" m lines.
If m is not specified, the pointer is moved one line. If the end of
the file is reached, an "End of file reached by n m" message is output
by NETED; the pointer is left "after" the last line.

2. l string

The l(ocate) request causes the pointer to be moved to the net line
containing the character string "string" (which may contain blanks);
the line is output. If no match is found, a message of the form "End
of file reached by l string" will be output (and the pointer will
have returned to the top of the file). The search will not wrap
around the end of the file; however, if the string was above the
starting position of the pointer, a repetition of the locate request
will find it, in view of the fact that the pointer would have been
moved to the top of the file. To find any occurrence of the string --
rather than the next occurrence -- it is necessary to move the pointer
to the top of the file before doing the locate (see following
request).

3. t

Move the pointer to the top of the file.

4. b

Move the pointer to the bottom of the file and enter input mode.

5. "."

Leave the pointer where it is and enter input mode. (First new line
goes after current old line.)

6. i string

The i(nsert) request cause a line consisting of string (which will
probably contain blanks) to be inserted after the current line. The
pointer is moved to the new line. Edit mode is not left.

7. r string

The r(eplace) request causes a line consisting of string (probably
containing blanks) to replace the current line.

8. p m

The p(rint) request causes the current line and the succeeding m - i
lines to be output. If m is not specified, only the current line will
be output. End of file considerations are the same as with "n".

9. c /s1/s2/ m g

The c(hange) request is quite powerful, although perhaps a bit complex
to new users. In the line being pointed at, the string of characters
s1 is replaced by the string of characters s2. If s1 is void, s2 will
be inserted at the beginning of the line; if s2 is void, s1 will be
deleted from the line. Any character not appearing within either
character string may be used in place of the slash (/) as a delimiter.
If a number, m, is present, the request will affect m lines, starting
with the one being pointed at. All lines in which a change was made
are printed. The pointer is left at the last line scanned. If the
letter "g" is absent (after the final delimiter) only the first
occurrence of s1 within a line will be changed. If "g" (for "global")
is present, all occurrences of s1 within a line will be changed. (If
s1 is void, "g" has no effect.) NOTE WELL: blanks in both strings
are significant and must be counted exactly. End of file
considerations are the same as with "n".

10. d m

The d(elete) request causes m lines, including the current one, to be
deleted from the working copy of the file. If m is not specified, only
the current line is deleted. The pointer is left at a null line above
the first undeleted line. End of file considerations are the same as
with "n".

11. w

Write out the working copy into the storage hierarchy but remain in
NETED. (Useful for those who fear crashes and don't want to lose all
the work performed.)

12. save

Write out the working copy into the storage hierarchy and exit from
NETED.

Additional specs:

a. On Multics, type-ahead is permitted. This approach is recommended
for all versions of NETED, but is of course not required as various
Servers' NCP Implementations may prohibit it; however:

b. If an error is detected, the offending line is output, and pending
typeahead (if any) must be discarded (to guard against the possibility
of the pending request's being predicated on the success of erroneous
request).

c. The command is not reinvokable, in the sense that work is lost if
you "quit" out of it via the Telnet Interrupt Process command or its
equivalent; indeed, quitting out is the general method of negating
large amounts of incorrect work and retaining the original file
intact.

(When the time comes, I'll be glad to furnish examples for the users'
manual version; but for now, that's all there is.)

NOTE

It really does work, unsophisticated though it may be. I think that
it's sufficient to get new users going, and necessary to give them a
fighting chance. It would even be of utility within the NWG, for
those of us who don't like having to learn new editors all the time.
If anybody wants to try it, I'll make a version available to
"anonymous users" (see the Multics section in the Resource Notebook if
you don't already know how to get in our sampling account), under the
name "neted". (*) (If you do try it, please delete files when done
with them.)

_____

(*) Knowledgeable Multics users with their own accounts can instead
link to >udd>cn>map>neted. It is also there under the names "eds" if
you want to save typing a couple of characters.