Network Working Group
Request for Comment: 666
NIC: 31396

M. Padlipsky

26 November 1974

Specification of the Unified User-Level Protocol

After many discussions of my RFC 451, I discovered that the "Unified User-Level Protocol" proposed therein had evolved into what had always been its underlying motivation, a common command language. There are several reasons why this latter approach satisfies the original goals of the UULP and goes beyond them into even more useful areas:

1. User convenience. As evidenced by the good response to the common editor "neted", the Network Working Group has come to acknowledge the fact that the convenience of non-system programmer users of the Network must be served. Allowing users to invoke the same generic functions -- including "batch" jobs -- irrespective of which Server Host they happen to be using is surely a compelling initial justification for a common command language. Note that the concern with generic functions -- which "all" Servers do, one way or another -- is intended to emphasize the common command subset aspects of the language, rather than the "linguistic" elegance of it all. The attempt is to specify an easy way of getting many things done, not a complicated way of getting "everything" done.

2. "Resource sharing". Another area which is receiving attention in the NWG of late is that of "automatic" or program-driven invocation of resources on foreign systems. A common intermediate representation of some sort is clearly necessary to perform such functions if we are to avoid the old "n by m problem" of the Telnet Protocol -- in this case, n Hosts would otherwise have to keep track of m command languages. For the common intermediate representation to be human-usable seems to kill two birds with one stone, as expanded upon in the next point.

3. Economy of mechanism. In RFC 451, I advanced the claim that a single user-level protocol which connected via socket 1 and Telnet would offer economy of mechanism in that new responders would not be required to service Initial Connection Protocols on socket after socket as protocol after protocol evolved. This consideration still applies, but an even greater economy is visible when we consider the context of resource sharing. For if the common command language is designed for direct employment by users, as the present proposal is, there is no need for users on terminal support "mini-Hosts" (e.g., ANTS and TIPs) to require an intermediary server when all they actually want is to work on a particular Server in the common

language. (This is especially true in light of the fact that many such users are not professional programmers -- and are familiar with no command language.) That is, if resource sharing is achieved by an intermediate language which is only suitable for programs, you would have to learn the native command language of Server B if you didn't want to incur the expense of using Server A only to get at generic functions on Server B. (And you might still have to learn the native language of Server A, even if the expense of using two Servers where one would do isn't a factor.)

4. Front-ending. Another benefit of the common command language proposed here is that it is by and large intended to lend itself to implementation by front-ending onto existing commands. Thus, the unpleasant necessity of throwing out existing implementations is minimized. Indeed, the approach taken is a conscious effort to come up with a common command language by addition to "native" command languages rather than by replacement, for the compelling reason that it would be unworkable as well as ill-advised to attempt to legislate the richness represented by existing command languages out of existence. Further, as it is a closed environment, no naming conflicts with native commands would arise.

5. Accounting and authentication. As evidenced by the spate of RFCs about the implications of the FTP in regard to both accounting for use of Network services and authenticating users' identifications (Bressler's RFC 487, Pogran's RFC 501, and my RFC 505 -- and even 491), this area is still up in the air. The generic login command proposed here should help matters, as it allows the Server to associate an appropriate process with the connection while actuating appropriate accounting and access control as well, if it chooses.

6. Process-process functions. By enabling the invocation of foreign object programs, the present proposal offers a rubric in which such process-to-process functions as "parallelism" can be performed. (See the discussion of the "call" command, below.) Note that the UULP is not being advanced as a panacea: It is assumed that the actual transactions carried out are most likely not going to be in the common command language (although some certainly could be); however, what is furnished is a known way of getting the presumably special-cased programs executing elsewhere. Also, it offers a convenient environment into which can be placed such new functions, which we would like to have become generic, as Day's File Access Protocol.

All of which seems to be a fair amount of mileage to get out of a distaste for remembering whether you find out who's logged in by saying "systat", "users", "s.who:c", "listf tty", or "who"....

Context

Although ultimately intended to become the general responder to the Initial Connection Protocol, the UULP is initially to be a Telnet Protocol "negotiated option". When the option is enabled, the Server Host will furnish a command environment which supports the common conventions and commands discussed herein.

In a sense, the UULP is a "selector". That is, the common command subset includes commands to exit from the common command environment and enter various other environments, along the lines of CCN's current Telnet Server. To exit from the UULP environment to the "native" command processor, the UULP command is "local" (see also the discussion of Case, below). Note that all commands terminate in Telnet "Newline" (currently cr-lf), unless altered by the "eol" command (below); internal separator is space (blank). (Entrance into other environments -- such as the FTP Server -- is discussed below.) There are two reasons for introducing a mechanism other than the apparently natural one of simply de-negotiating the option: First, it is bound to be more convenient for the user to type a command than to escape to his User Telnet program to cause the option disabling. Second, it is hoped that eventually the UULP will be legislated to be the default environment encountered by any Network login, in which case the natural way to enter the Server's "native" command environment would be by UULP command.

> Note: all UULP commands discussed herein are listed in Appendix 1, categorized as to optionality, with brief descriptions given. The appendix may be taken as a first-pass UULP Users' Manual.

Responses

Any optional commands which are not supported by a particular Server are to be responded to by a message of the form "Not implemented: commandname.", where the variable is the name of the command which was requested. Note that throughout this document, all literals must be sent exactly as specified, so as to allow for the possibility of Servers' being driven by programs (including "automata" or "command macros") in addition to "live" users.

In general, the view has been taken here that a small number of literal, constrained responses is superior to a vast variety of numerically coded responses in which text may vary. Again, the motivation is to achieve an economy of mechanism. For on the coded model, there must be a coordinator of code assignments, which is just as well avoided. Further, as has been experienced in the use of the FTP, when there are many codes there are many ambiguities. (The sender may have a perfectly valid case for choosing, say, 452, while

the receiver may have an equally good interpretation of the codes' definitions for expecting, say, 453.) Experience with a related "error table" mechanism on Multics also bears out the assertion that coded responses create both managerial and technical problems. A final objection to numeric codes might be considered irrelevant by live some, but I think that the aesthetics of the situation do merit some attention. And when the common command language is being employed by live users, it seems to me that they would only be distracted by all those numbers flying around. (Nor can we assume that the numbers could be stripped by their "User UULP", for one of the basic goals here is to make it straightforward enough for a user at a TIP to deal with.)

Arguments

During the review process, it became evident that some global comments on arguments were in order. Two areas in particular appear to have led to some confusion: the strategy of specification of arguments on the command line, and the question of "control arguments". On the first score, the goal of "front-endability" must be recalled. Consider two native implementations of a particular command, one of which (A) expects to collect its arguments by interrogation of the user, and the other of which (B) expects to receive them on invocation (being invoked as a closed subroutine). Now, it is easy to imagine that a "Server UULP" could feed the arguments to A as needed without requiring A to be rewritten, but it is quite difficult to see how B could be made to interrogate for arguments without extensive rewriting. Therefore, a "least common denominator" approach of specifying arguments in advance incurs the minimum cost in terms of reworking existing implementations.

On the second score, I have borrowed a notion from the Multics command language's convention called "control arguments" because it seems to be quite convenient in actual practice. The key is that some arguments are meant as literals, usually specifying a mode or control function to the command, while others are variables, specifying something like a particular file name or user identifier. A common example is a "mail" command, where the variables are the user identifiers and the Host identifiers, and the "control argument" is the designator that user identifiers have ceased and Host identifiers have begun. The convention used here is to begin the control argument with a hyphen, as this character never seems to be used to begin variable arguments. Thus, we use "-at" in the mail example. Although it is not a deep philosophical point, this approach does relieve argument lists of order-dependency, and feels right to me.

Case

Although it appears to have been legislated out of existence by the specification of the Network Virtual Terminal's keyboard in the Telnet Protocol, the question of what to do about users at upper-case-only terminals remains a thorny one in practice. There are two aspects to consider: the alphabetic case of commands, and the ability to cause "case-mapping" in order to allow lower-case input. Some Servers have no local problems with the first aspect, as they operate internally in all upper-case or all lower-case and merely map all input appropriately. (Problems do arise, though, when one is using the User FTP on such a system to deal with a mixed-case system, for example.) Other Servers, however, attach the normal linguistic significance to case. (E.g., Smith's name is "Smith" -- not "SMITH", and not "smith".) To minimise superfluous processing for those Servers which are indifferent to case, all UULP commands are to be recognized as such whether they arrive as all upper-case or all lower-case. (They will be shown here as all lower merely for typing convenience.) Note that arbitrarily mixed case is not recognized, as it is an unwarranted assumption about local implementation to suppose that input will necessarily be case-mapped.

On the second aspect, any Server which does distinguish between upper- and lower-case in commands' arguments (a.k.a. parameters) must furnish a UULP "map" command as specified in Appendix 2 in order to support logins from upper-case-only terminals attached to User Hosts which either do not support the Telnet Protocol's dictum that all 128 ASCII codes must be generable, or support it awkwardly. This seems a simpler and preferable solution than the alternative of legislating that upper-case Network-wide personal identifiers (and perhaps even Network Virtual Path Names) be pre-conditions to a usable common command subset. (As noted below, these latter concepts will fit in smoothly when they are agreed upon. The point here, though, is that we need not deprive ourselves of the benefits of a UULP until they are agreed upon.)

User Names

As implied above, the various Servers have their various ways of expressing users' names. Clearly, the principle of economy of memory dictates that there should be a common intermediate representation of names in and for the Network. It is probably also clear that this representation will be based upon the Network Information Center's "NIC ID's". However, it is unfortunately amply clear than an acceptable mechanism for securing up-to-date information cannot be legislated here - much less a mechanism for securely updating the implied data base. Therefore, at this stage it seems to be the

sensible thing to specify only the UULP syntax for conveying to the Server the fact that it is to treat a user name as a Network-wide name rather than as a local name, and let the supporting mechanisms evolve as they may.

The prefacing of a name with an asterisk ("*") denotes a Network-wide name. (Such names may be either all upper-case or all lower-case, as with UULP commands' names.) The name "*free" is explicitly reserved to mean that (in the context of logging in) a login is desired on a supported or sampling account, if such an account is available. The response if no such account is available is to be "Invalid ident: *free." When Network-wide names are generally available Servers will either map them into local names or cause them to be registered as local names as they prefer. The point is that a Network-wide name will be "made to work" by the Server in the context of the UULP.

Special Characters and Signals

Another area in which the facts of life must outweigh the letter of the Telnet Protocol if the user's convenience is to be served is that of "erase" and "kill" characters. It is possible that User Telnets will uniformly facilitate the transmission of the Telnet control codes for generic character erase and generic line kill. It is certain, however, that User Telnets will differ -- and users will, if they use more than one User Telnet, be again placed in the uncomfortable position of having to develop too many sets of reflexes. Therefore, the UULP will optionally support the following commands: "erase char" and "kill char", where char is a printable ASCII character (to avoid possible conflicts with "control characters" which are recognized in the innermost areas of particular operating systems). Presumably, unwary users can be instructed not to choose an alphabetic, so as to avoid being placed in a position where they cannot invoke certain commands (erase and kill themselves, for example, in which case they couldn't be changed).

These commands are supplements to the related Telnet control codes, and have the same meanings. The point here is that it may be far more convenient for a user to be able to say "erase #" and get the "#" to be recognized as the erase character by the Server than for the user to get his User Telnet to send the Telnet equivalent. The commands are designated as optional because they may lead to severe implementation problems on some Servers, and because the equivalent functions do, after all, exist in Telnet.

> Note: the erasing is assumed to be performed "as early as possible". That is, the sequence "erase x" "erase x" should come out equivalent to "erase x" "erase" -- the second appearance of

"x" resulting in the erasing of the space in the command line. Presumably, this is a sufficiently uncommon path that anomalous results would be tolerated by the user community, but the intent ought to be clear.

The Telnet "synch" and "break" mechanisms are, by their very nature, best left to Telnet. End of line, however, might well be a different story. Therefore, as a potential convenience, the UULP optionally supports "eol char" to ask the Server to treat char as the end of line character thenceforth. To revert to Telnet Newline, "eol" (i.e., no argument, current terminator).

Prompts

Another aspect in which Servers vary while being the same is how they indicate "being at command level". Some output "ready messages"; others, "prompt characters". For the UULP, where some functions will be performed by means of a command's logging in to another system, the ability to specify a known prompt character is extremely desirable. The UULP command is "prompt char" where char is the character which is to be sent when the user's process (on the Server) is at command level. It is explicitly permitted to prefix char to a line consisting of a "native" prompt or ready message. Also, this command is explicitly acknowledged to be permissible prior to login. (Again, warning must be made of the bad results which can ensue if an alphabetic character is chosen.)

Note: "prompt", "eol", "erase", and "kill" may all be re-invoked with a new value of char in order to change the relevant setting; all may be turned off by invocation with no argument.

Login

Perhaps the stickiest wicket of them all is the attempt to specify a generic login, but here we go. The UULP login command is "login userident", where userident is either a locally-acceptable user identifier or a Network-wide identifier as discussed above. Note that for utility in contexts to be discussed later, the locally-acceptable form must not contain spaces. Servers may respond to the login attempt with arbitrary text (such as a "message of the day"), but some line of the response must be one of the following: a prompt (as discussed above; indicating, in the present context, successful login); "Password:"; or "Invalid ident: userident." When passwords are required, it is the Server's responsibility either to send a mask or to successfully negotiate the Hide Your Input option.

Note that "login *free" is specifically defined to require no password. (If a "freeloader" has access to a User Telnet and has learned of the "*free" syntax, it is fruitless to assume that he couldn't have also read the common password.) If a password must be given, acceptable responses are arbitrary text containing a line beginning either with a prompt or with "Login unsuccessful." or with "Account:". If an account is requested, the responses must be either the "Login unsuccessful" message or the text containing a prompt already described. If any errors occur during the login sequence, users are to re-try by starting from the login command. (I.e., it is not required that the Server "remember" idents or passwords.)

It is explicitly acknowledged that an acceptable response to "login *free" is "Limited access only." (followed by a prompt). This is intended to warn (human) users that the free account on the Server in question exists only to allow such functions as accepting mail and telling if a particular user happens to be logged in. (For objections to "loginless" performance of such tasks, see RFC 491. Note also that nothing here says that a Server must do anything other than return a prompt in response to "login *free" in the event that loginless operation is natural to it.) Given the UULP login discipline and the "prompt" command, it is reasonably straightforward for a program to login on a free account and perform one of these functions, for if the login command succeeded, the program will "see" a guaranteed prompt character.

To make life simpler for those Hosts which normally have some sort of "daemon" process service mail and the like, a further expansion to login is in order. The point here is that some Hosts may not know what sort of process to pass an unqualified "login *free" to, whereas they'd be sure what to do with an explicit request to process mail, do a who command, or set up console to console communications. Therefore, UULP "login" will allow a "control argument" (as discussed above) of either "-mail", "-who", or "-concom", and the respective UULP commands involved must use the respective strings in any login line they transmit. Again, nothing is being said about what a Server has to do with the information, but some Servers need/want it.

Usage Information

Most Servers offer some sort of on-line documentation, from calling sequences of commands to entire users' manuals. There are two sorts of information of interest in the UULP environment: "normal" system information, and information about the particular Server's UULP implementation. To learn how to get descriptions of "native" commands, the UULP command is "help -sys" (abbreviation: "?"). Note that "-sys" is viewed as a "control argument" and as such prefaced by

a hyphen ("-") to facilitate distinction from other sorts of name (e.g., command names). To get a description of the Server's UULP implementation, "help -uulp". To get a description of a particular UULP command's implementation, "help comname". To be reminded of how to use the help command, "help".

Note: as with command names and Network-wide user names, control arguments may be either all upper-case or all lower-case.

It is specifically acknowledged that "No peculiarities." is an appropriate response to "help comname" if nothing of interest need be said about the Server's implementation of the UULP command in question. (After all, we're sparing users the necessity of studying a dozen or so users' manuals; the least they can do is to read the UULP command list.) Appropriate information for less taciturn Hosts to furnish would be such data as local command invoked (if such be the case), argument syntax (e.g., pathname description, or name of help file about pathnames), "To be implemented.", or even "Not to be implemented."

"Mail"

Even though a separate mail protocol is being evolved for general purposes, the UULP needs to address this topic as, by virtue of being login based, it allows systems which do access control and sender authentication on mail to make these abilities available to users within its framework of generic functions. Therefore, to read one's mailbox, the UULP command is "readmail". To have "live" input collected and sent to a local user, "mail userident"; to a remote user, "mail userident -at hostname", where the arguments have the "obvious" meanings. To send a previously-created file, "mail -f filename userident -at hostname". Several useridents may be furnished; the delimiter is space (blank). Similar considerations apply to hostnames. If both are lists, they sould be treated pairwise. (A more elaborate syntax could be invented to deal with the desire to send to several users at a given host and then to other users at other hosts, but it seems unnecessary to do so at this point, for multiple invocations would get the job done.)

The mail command prefaces the message with a line identifying the sender (Host and time desirable, but not mandatory). For "live" collection, the end of message is indicated by a line consisting of only a period (".") followed by the regnant line terminator (usually the Telnet Newline, but see also the discussion of the eol command). If remote mail is not successfully transmitted, it is to be saved in a local file and that file's name is to be output as part of the failure message. ("Queueing" for later transmission is admired, but

not required.) The transmission mechanism will follow the general mail protocol. Note that when invoked with a "-at" clause, the mail command will send "login *free -mail" to the remote Host(s), followed by a mail command with no "-at" clause.

A desirable, but not required, embellishment to "readmail" would be the accepting of a Host name ("-at hostname") to cause the local Host to go off to the named Host (via "login *free -mail") and check for mail there. Several hostnames could, of course, be specified. A further embellishment, which would probably be quite expensive, would be to accept "-all" as a request to check all Hosts (or, perhaps, all Hosts known to have a free account for the purpose) for mail.

Direct Communication

The ability to exchange messages directly with other logged in users is apparently greatly prized by many users. Therefore, despite the fact that there is a sense in which this function is not within the purview of the UULP, we will address it, after a digression.

> Digression: The UULP assumes that there can be straightforward "front ends" at the various Servers which translate generic function calls in a common spelling to calls for specific, pre-existing "native" functions. In the area of console to console communications, however, this premise does not really hold. The problem is that both major "native" implementations known to the author are seriously flawed. The TENEX "link" mechanism is both insecure (you've got no business seeing everything I type even if I'm careless enough to let you) and inconvenient (why should I be forced to remember that pesky semi-colon? how do I get back into phase after I've forgotten one?). It is also likely to be extremely difficult to simulate on systems which do not force Network I/O through local TTY buffers, even if the user interface were not subject to criticism. The Multics "send_message" mechanism, on the other hand, has a more sophisticated design, but is absurdly expensive. Therefore, the UULP mechanism to be described assumes that, for this function, new local implementations will be developed to support it.

To permit console to console communications: "concom -on"; to refuse, "concom -off". Default is off. To enter message-sending mode: "concom userident -at hostname" ("-at" clause is optional). To exit from message-sending mode, type a line consisting of only a period (cf. Mail, above). While in message-sending mode, each line will be transmitted as a unit. The first message sent by concom must be prefaced by an identifying line, beginning "From:" and containing an appropriate address to which to reply. The closing period-only line

should be transmitted, so as to allow the other concom to close as well. Acceptable error response is "Not available: userident." (which neither confirms nor denies the existence of the particular user -- a matter of concern on the security front). The command must, of course, do whatever is necessary to transmit the messages; i.e., if locally invoked, access the local mechanism, and if invoked for remote communications, access the remote Host's concom command (via "login *free -concom"). Thus, a user at a TIP would use the local form of concom on the Host of the other party if this is convenient, or would use the remote form on his "usual" Server if the direct use is inconvenient for some reason (such as having no account there, say).

The prerequisites for establishing communications are to find out if the user is logged in, and what "address" to use if so. The mechanism for gathering this information is an expanded "who" command. (Note that "who" is the UULP command to invoke the generic who's logged in function, with no constraints on format of reply.) The syntax is "who userident -at hostname", where both arguments may be multiple. If no "-at" clause, then check local Host only. Response must begin "From hostname: userident:" followed by either an appropriate address (e.g., "ll" if local "concom" uses TTY numbers and userident is logged in on TTY ll), or "Not available."

As with mail, a "-all" embellishment might be pleasant. Note that the search for the specified user(s) -- whether or not "-all" is used -- still assumes that a "login *free -who" login will be used on the appropriate remote Host(s), followed by "who userident". This is why responses to the expanded who command must be so rigidly specified. Note also that regardless of whether the inquiry is made in terms of Network-wide or local user name, the response must be appropriate for use in "concom".

"Good" concom implementations will presumably do an expanded who command automatically, so as to spare the user the necessity of having to do it separately. Indeed, the -concom control argument to login is defined to imply the ability to do a who as well as a concom to cater to this possibility. It is tempting to legislate that such an approach be the rule, but the implementation implications are not quite clear enough to do so. The implicit who should be viewed as a strong hint to implementers, though.

File Creation and Manipulation

The common command subset must furnish the ability to create and manipulate files. Creation is necessary in order to send mail on the one hand, and to produce source files for subsequent compilation on the other hand. Manipulation (such as copying, renaming, typing out,

and the like) is necessary both as a convenience aspect for users who seek to operate only in the common command language and as a means of performing desired batch functions (see below). For file manipulation commands, the user could enter the File Transfer Protocol environment. However, the FTP user interface is constrained by a very high degree of program-drivability. It is also lacks abbreviations and suffers from the lack of mnemonicity dictated by limiting command names to four characters. Further, some valuable functions (such as causing a file to be typed out) are not dealt with. Therefore, various UULP file manipulation commands are given in Appendix 1. They need not be addressed in detail here. However, some context would be useful:

The file manipulation commands assume that all Servers have some notion roughly corresponding to "the user's working directory". All file names, whether the yet to be invented Network Virtual Pathname or the "local" variety, are taken to refer to files in this directory unless otherwise indicated. That is, the user should not have to furnish "dsk:" or the like; it is taken as given that when he refers to file "x" he means "the file named 'x' in my current working directory" and the Server "knows" what that means.

At the present stage of development of the UULP, it does not seem fruitful to go into a reasoned explication of the following statement. For now, suffice it to say that those file manipulation commands (a copy of a foreign file, for example) which need to employ the FTP do employ the FTP and let it go at that. As the context and implications of the protocol become more widely understood, the detailed implementation notes will be added to the file commands -- and refined for the other commands, doubtless. In a way, the common file commands may be viewed as a kind of "User FTP" of known human interface when they deal with foreign files. (And, of course, until there's a Network virtual pathname, the issue doesn't really arise.) I expect that an "identify" command might be desirable, so that UULP commands which have to access other Servers in turn on behalf of the specific current user can have the necessary login information available to them. Such a command is included in Appendix 1, but should rank as speculation for now.

On the topic of file creation, matters are rather complicated. It is clear that the ability to create files in the UULP environment is extremely desirable. It is also clear that using mail to a fake address to get the file created, then renaming the "unsent mail" file is too byzantine to expect users to do. Unfortunately, it is not clear exactly what the alternative is. That is, it's fairly clear that we need a common editor, but it's not at all clear which editor it should be.

Two widely-known editors come to mind: TECO and QED. However, not everybody has them. Even if everybody did, the "dialects" problem is bound to be a large one. Even if all the relevant system programmers could agree, there remains the question of whether the intended user population would be willing to bother learning a language as complex as TECO or QED. Therefore an optional UULP command to be called "neted" is proposed. (See also RFC 569.) This editor is a line-oriented context editor (no "regular expressions", but also no line numbers). It is copiously documented in Chapter 4 or the Multics Programmers' Manual, including an annotated listing of the (PL/I) source code. A simple user's guide has been prepared (see Appendix 3). Several implementations already exist, and commitments have been made for more. It may also be repugnant to some of the system programmers who would be called upon to implement it -- which is why it is optional, until and unless higher authority makes it mandatory.

Other Protocols

The nominal initial impetus for proposing a UULP was to allow new Network user protocols to be invokable through a common mechanism, rather than requiring a new responding mechanism to be built for a new contact socket for each new protocol. Although this goal has been shunted into the background by the admission of the true goal of the UULP, it has not been dropped completely. Therefore, to enter the FTP Server environment, the UULP command is "ftp"; to enter the RJE Server environment, the UULP command is "rje". Exit is as per the respective protocols. (Where possible, exit should be back to the UULP environment.)

Invoking Foreign Programs

There are two broad contexts in which it is desirable to cause a specific local program to be invoked from the common command environment: The User side of the connection may itself be a program, and the desired Server side program a specifically cooperating one; this is the more sophisticated context, of course. The less sophisticated context assumes that the User side is a "live" user, and the desire is to invoke a compiler or an object program the user has already compiled in the common language -- again as a convenience to the user so that he may operate in a sort of "Server-transparent" mode. (The latter case also covers "batch" use of the Server; see below.) In both contexts, the important role of the UULP is to specify the mechanisms through which the particular programs may be invoked, irrespective of the idiosyncrasies of the Servers' command languages.

Programming languages are much too big a problem to tackle here. However, assuming that a user somehow manages to create a source program, he still wants some commonality of spelling in invoking the appropriate compiler, or even the object program. As an optional but strongly recommended UULP command, then, "call name" should invoke object program name (where the named program may be a "native" command with arguments specified as appropriate). The values "pl1", "-basic", "-fortran", "-lisp", etc., should be recognized as requesting the invocation of the appropriate language processor (to operate on a named source file or interpretively/interactively if no source file was named), with "reasonable" defaults in effect. Note that this all is meant to imply that "native" commands are not directly invokable from the UULP environment (other than by "call"), to avoid potential naming conflicts between system commands and new UULP commands.

Note that the "call" command in the UULP environment constitutes a rubric for "parallel" computation, given any ad hoc convention for the return of completion information. (Writing on the Telnet write socket plus 2 would seem appropriate, provided the initiator has the ability to "listen" for the rfc; but even a response in the data stream as a special-cased program is assumed on the "user"side anyway.)

Other Matters

The topic of "batch" mode merits some attention. As with the file manipulation commands, more consultation is necessary for a firm spec. However, I suspect that a "-batch" control argument to login should initiate batch mode processing by the Server, and given the call and identify commands all we might then require is a convention for designating the output file in order to return it via a copy command in the "job" itself (if output is to be returned rather than stored at the Server). Of course, -batch will probably need some substructure as to password and timing matters. More details will emerge in this area in future iterations.

An admittedly fictionalized scenario might look like this:

```
login Me -batch -pw xxx -shift 3
copy *452<me>source.text source.pl2
call -pl2 source
call source input output
identify Me2 yyy
copy output *555>root>Me>output452
logout
```

where user "Me" wants the Server receiving the commands (either directly from him at a TIP or perhaps from some other Server on which he has created a file containing them) to set up a batch job for him, with password "xxx", to be run on Shift 3 (whenever that is). The job first copies file "source.text" from directory "<me>" on Host 452 into local file "source.pl2", then compiles it with the local PL2 compiler, executes it (assuming a "Not found" response would go into a known file if compilation had failed) with specified arguments (presumably the names of files for input and output), then copies the "output" file to Host 555's file hierarchy at the indicated place, using the user identifier "Me2" and the password "yyy". It's not elegant, but it ought to work.

Finally, on the topic of logging out, the UULP command is "logout". The Server must close the Telnet connection after doing whatever is appropriate to effect a logout. To retain the Telnet connection, "logout -save". Having the Server close is viewed as a convenience for the user, in that it spares him the necessity of causing his User Telnet to close. It is also desirable for program-driven applications, so as not to leave the connections "dangling" and not to require possibly complex negotiations with the User side to break the connection.

APPENDIX 1. THE COMMON COMMAND SUBSET

Syntax	Opt
I. "Set-up" Commands	
login id arg The id may be Network-wide or Host-specific. "*free" is reserved. The arg may be "-mail", "-who", "-concom", "-batch", or may be absent. Result is to be either logged in or passed off to appropriate daemon.	
prompt char Specifies that char is to become or precede the normal prompt message. Acceptable prior to login.	
erase char	X
Specifies that char is the erase character. Invocation with no argument reverts to default.	
kill char	X

Specifies that char is the kill character.
Invocation with no argument reverts to default.

eol char X
Specifies that char is the newline character.
Invocation with no argument reverts to default.

local
Enter the local command environment.

ftp
Enter the FTP environment.

rje
Enter the RJE environment.
logout
Logout and sever the Telnet connection.

logout -save
Logout but keep the Telnet connection.

map
Apply the case-mapping conventions of Appendix 2.
Required on Hosts to which case is significant.

identify id arg X
Specifies that id is to be used as the user
identifier in any "fanout" logins required.
If arg is specified, it is to be either the
password to be used in such logins or "-pw", in
which case the Server will furnish a mask or negotiate the Hide Your Input Telnet option; if no arg, then no password is to be furnished on fanout logins.
Default id is "*free".

II. Communications Commands

readmail
Type out "mailbox".

readmail (id) -at host X
Type out "mailbox" on remote Host host.
Multiple Hosts may be specified,
separated by spaces (blanks).

Implies ability to change working directory
at host to directory implied by known
user identifier, or (optionally) by id.

readmail -all XX

Search for mail.
Extremely optional.

mail id
Collect input until line consisting of
only a period (".") for mailing to local
user specified by id.

mail -f file id
Send contents of specified file to specified
local user.

mail id -at host
Collect input until line consisting of
only a period (".") for mailing to remote
user(s) at specified Host(s). Both id and
host may be multiple, separated by spaces.
(If multiple, they should be taken pairwise.)

mail -f file id -at host
Send contents of specified file to specified
remote user(s).

who
The generic who’s logged in command.

who id
Is id logged in? Constrained responses.

who id -at host
Is the specified user logged in at the
specified host. Constrained responses.

concom -on
Enable console to console communications.

concom -off
Disable console to console communications.

concom id
Send messages to specified local user
until line consisting of only a period (".").

concom id -at host
Send messages to specified remote user.

III. File Commands

type path
Type out the contents of the specified file.
Pathname may be local or Network-wide.
Default to current working directory.

listdir
List the contents of the current working directory. (Local format acceptable.)

listdir path
List the contents of the specified directory.

rename old new
Change the specified file's name as indicated.

addname old new X
Give the specified file the specified extra name.

delete path
Get rid of the specified file.
("Expunge" if necessary.)

copy from to
Make a copy of the file specified by the first pathname at the second pathname.

link from to X
If your file system has such a concept, make a "link" between the two pathnames. If no second argument,
use same entry name in working directory.

status path st X
If your file system has such a concept, give status information about the specified file or directory.

changewd path X
If no argument, return to the "home" directory.

typewd
Type out the pathname of the current working directory.

neted path

See Appendix 3.

IV. Invoking "Native" Programs

call name (args)
Invoke the specified program with the
specified arguments (if any).
The following names are reserved to indicate the
invocation of the corresponding language processor: "-pl1", "-basic",
"-fortran", "-lisp".
(If no source file indicated, invoke "interpretively" if possible.)

V. On-line Documentation

help name
Type out information about the specified UULP command. If name is
"-sys", type out information about how to use the local system's help
mechanism; if
"uulp", about the local system's UULP implementation. If no name
given, describe the command itself.

APPENDIX 2. MAP COMMAND CONVENTIONS

This appendix will eventually contain the case-mapping conventions detailed in RFC 411.

APPENDIX 3. EDIT COMMAND REQUESTS

This appendix will eventually contain descriptions of the neted command requests (a draft of which now exists), or a reference to the Resource Notebook version, if that gets published first. For now, it should be sufficient to point out that the requests are basically locate, next, top, change, save, and quit -- i.e., it's the "old-fashioned" flavor of context editor.

[Optical character recognition and initial proofreading performed 11/20-21/04 by The Author. A few original typos were corrected; some may remain.]