To: MTB Distribution

From: Jim Siwila

Date: 06/29/82

Subject: Online Documentation for Multics

This MTB surveys online documentation facilities that seem to be applicable to Multics. It covers a number of possibilities--ideas that I've had, those of others on the Multics Project, and some discovered through research. Some of these ideas will be easier to implement than others, and some, perhaps, are more urgently needed. I offer my opinion about that in the Recommendations section at the end. To facilitate discussion of the MTB, I am convening a Forum meeting named online_doc (od) in my home directory (>udd>Pubs>Siwila). I encourage everyone to comment further on the feasibility of these projects, suggest procedures for implementing them, and recommend priorities.

Overview

There are several types of user assistance we might add to Multics as well as some enhancements we can make in existing facilities. First of all, Multics should probably offer more information about the system itself in the form of explanations of topics and concepts. We already offer some of this information in gi info segments, and of course we have a lot of information similar to this in our commands and subroutines infos, though those could be made easier to read. Ideally, concepts and commands would be integrated in a way that would permit ready cross-referencing. As a part of this, or perhaps separately, we could offer a tutorial on basic Multics information. It would also be good to offer a few more specialized tutorials, such as "teach_emacs."

Secondly, more help could be available to users while they are typing command lines. Increased interaction at this level of processing is discussed often in the literature on man-machine communications. For example, when prompted the system might tell the user what kind of argument comes next on a particular command line. It would also be good to offer more help with errors than our current error messages give. Then too, I would like to install the documentation commands where_doc and explain_doc so that users could find out about Multics manuals while online.

Finally, we must consider uses for INTELLECT. It seems like a natural tool for online documentation retrieval, but some problems have been encountered during experiments done thus far. Those will be discussed in detail later in the report.

Help With System Concepts and Commands

A large part of the literature dealing with online documentation concentrates on methods of displaying system information. Multics already has a considerable amount of information online in its info segments on commands and subroutines and its general information infos. But more can be done along this line. This winter I installed glossary items from the MPM Reference Guide in gi infos in response to a long standing request by Tom VanVleck to get more such information online (see TR4579). That is just a small, interim fix, though. This part of our help facility needs to be enhanced much more by adding significantly to the general information type of files and by making the module info segments easier to read.

A number of people have suggested that we put more information from our manuals online. There is, in fact, some literature on user assistance systems that are built on information that is in manual form. The simplest means of doing this would be to place portions of manuals into gi info segments, put the segments in >doc>info, and access them just as we do now.

That is not the direction suggested by the literature, however. The articles I've read are about systems that can read online versions of the manuals and extract info files from them. CP-6 has a help facility like this. Perhaps its most interesting feature is the single-sourcing of manuals and help files that enables users to read most manual information online and to print out the manuals as well. This is managed by compose-like controls in the online manual files that designate manual only, help only, and combined-purpose info text. Lee Baldwin has already created a version of such a tool for Multics manuals. She has written compose macros that enable her to create info segs directly from the Commands and Subroutines manuals.

Cross-referencing on CP-6 is done manually. That is, each info file itself contains explicit reference to related topics. This information is written into the files by writers before they become part of the help system. Some of the literature, however, suggests that cross-referencing can be done dynamically by using some kind of indexing system on the stored information. In an

article entitled "Using Offline Documentation Online," Lynne A. Price describes the system she designed as her doctoral thesis at the University of Wisconsin--Madison in 1978.(1) It would probably pay us to look at that thesis to see how she converted the data. With MRDS, it seems we have the perfect tool for providing dynamic cross-referencing on Multics. In fact, Lindsey Spratt has already written a tool for searching online information by topics. He searches all topics and then arranges the results of the search in a MRDS data base. From that data base we could construct menus for particular topics that would include subtopics and other related topics, including relevant commands and subroutines. To interrelate them all, we could make commands, subroutines, topics, and concepts all part of the same information source, perhaps the stored versions of our manuals.

One thing we wouldn't want to do with a new help system is force its elaborations on users, especially users who are accustomed to our current help system. Users should be able to go directly to the information they want when they know where it is. For instance, they should be able to get help with a command just as they do now, by typing "help COMMAND_NAME." Even in the case of new topics, users should be able to go directly to the subtopic desired if they know the name, without having to go through a series of menus. For example, the first menu for the topic "access" will include the subtopic "nondiscretionary access." When a user know she wants to read about nondiscretionary access, she should be able to get that explanation immediately. That doesn't mean she will not also get a menu of other options with her request. But that menu will include only the subtopics and other infos related directly to nondiscretionary access, including the parent topic "access." It will not be the same menu that would be displayed had the user asked for help with "access."

There is another viable model for constructing the system information component of our online documentation--the ITS system at MIT. That system has a hierarchical structure that directs the user to the next logical level of information and also lets the user select more randomly from a menu. The user enters this hierarchy by typing "INFO." That puts him in a menu of directories from which he selects the type of system info he wants to look at, including information about how to operate the help system. An advantage to this highly structured arrangement is that it provides a way of pointing to the next logical piece of

---

(1) Association of Computing Machines SIGSOC Bulletin, v. 13, Special Issue, pp. 15-20.

information. Some have pointed out that this type of presentation also encourages browsing and inadvertent discovery.(2)

There are drawbacks inherent in this system. A user has to go through at least one menu, and usually more, to get what she wants, which is annoying when when the user knows beforehand exactly what she wants. Also, in arranging the information in a set hierarchy, we are assuming that every user will want to learn according to our model. As Kehler and Barnes point out, such a system becomes cumbersome with large complex sources of information, requiring the user to retain a great deal of contextual information when searching for specific items. On the other hand, the ITS system usually gives users a pretty good idea of where they are in the hierarchy at any particular stage, something Honeywell's Human Factors Guide indicates it is important to do.(3) That is something an indexed system like Lindsey's can't do as well. Furthermore, in the latter we would often be unable to display text with menus, especially those menus displayed as the initial response to a query, because without a set hierarchy, we would have no basis for choosing which info seg, of all those retrieved by the particular request, to display first. The only time we could display an info seg with the initial menu is when the topic name used in the request matched the name of a particular info seg. However, by providing menus along with direct access to specified information, an indexed system like Lindsey's encourages users to browse while offering the advantages of goal-directed searching.

I think that no matter which type of system we choose, we should pay attention to the manner in which the explanations are written. CP-6 specifies that "definite information concerning specific techniques for using" the system is the most appropriate for online user assistance.(4) For the most part our manuals do not present information in this way; they describe the system instead. Furthermore, users have complained that our manuals don't exlain why things are done. I think we could address that

---

(2) Thomas P. Kehler and Mike Barnes, "Interfacing to Text Using HELPME," Association of Computing Machines SIGSOC Bulletin, v. 13, Special Issue, p. 117.

(3) Richard J. Frankosky, Human Factors Guide For Software Planners and Developers, 2nd ed., Honeywell Information Systems, p. 4-7.

(4) CP-6 FASTEXT Guide (CE59-00), p. 4-7.

concern at the same time we're focusing on techniques for doing things.

A similar concern was expressed by Fernando Corbato in a letter to Charlie Clingen written in October 1980. He said our info segments are not designed for rapid communication of ideas. He also noted that their prose is generally pretty turgid. He recommended that "every screen load should be polished like a precious gem." I think we have to look very carefully at what we actually display on the screen, and not be satisfied simply with getting it there in a very clever fashion.

The need for such rewriting of our documentation should not keep us from moving ahead now with an expansion of our general system information facility. That worthwhile improvements can be made in stages is demonstrated by Lee Baldwin's menu system for displaying our existing info segs. We could proceed immediately to single source all of our manuals, index them with Lindsey's tools, and create a menu interface to the resulting MRDS data base. Once we have all our documentation in a single source, we can make improvements to manuals and online information simultaneously, and thereby save time.

Help With the Command Line

In his letter to Charlie Clingen, Fernando Corbato also recommended two procedures that would facilitate typing of command lines on Multics. The first he called semi-automatic command completion. This would enable a user to send an incomplete character response to the command processor, which would process it if it were identifiable as a specific command or signal the user that the sequence was not unique. His second recommendation was that we provide a means for users to get a list of options available at any point in the command line. For instance, the user could stop after typing a pathname argument and enter a question mark (without a carriage return or line feed), and the system would then provide a list of things, say control arguments, that could be put in that position. After displaying the list, the system should redisplay the original command line with the cursor positioned for further input. Such command line processing is state-of-the-art right now. DEC has something very much like this on its TOPS 20 system, and literature I've read indicates that using contextual information in order to minimize what the

user must explicitly request is now seen as an important part of processing capability.(5)

Bill York has taken up the first of Corbato's suggestions in his Bachelor's thesis at MIT. He has implemented a system which uses the ESC key to tell iox_$get_line to find commands based on the character sequence provided and uses the question mark to list the full names of commands that begin with the characters already typed.

In the first case, when the user presses the ESC key, one of several things happens:

1) If the character sequence already typed is unique, that is, it matches only one specific command, then the rest of the command name is displayed and the cursor moves one space to the right, in position to receive arguments to the command.

2) If the character sequence already typed is not unique, then the bell is sounded and the cursor remains where it is, in order for the user to add characters to the command name.

3) If the character sequence can be recognized as the leftmost part of a specific command name, or names, as many characters as possible are displayed automatically. Then either the cursor moves one space to the right, if, as indicated in 1, the characters provided match unambiguously the name of a single command, or the bell is sounded and the cursor stays next to the character sequence so that the command name can be further specified. For instance, if a user presses the ESC key after having typed "prin", a "t" would be added to make "print", the bell would sound, and the cursor would remain in the space following the "t" in case the user wished to type out a longer command name, such as "print_wdir."

---

(5) See N. Relles, N. K. Sondheimer, and G. P. Ingargiola, "Recent Advances in User Assistance," Association of Computing Machines SIGSOC Bulletin, v. 13, Special Issue pp. 1-5 and Robert S. Fenchel, "An Integral Approach to User Assistance," same source, pp. 98-104.

Bill's system also allows users to stop while typing a command name and type a question mark to get a list of the commands that begin with the character sequence already typed. The question mark is not displayed, and when the list is complete the cursor is placed after the last character, ready for the user to complete the command.

These lists of commands would be culled from a table that would be created for each user's process based on the user's search rules. Bill says, however, that the program used to compile this table takes a long time to run, so a system-wide data base, available to all processes, would be better.

As to Corbato's second recommendation, Bill estimates it would take one programmer 6 months to implement such a system. Putting together the data base that this procedure would draw on would take more time, however.

The system Bill has implemented requires no change to the command processor. The actions signalled by the ESC key and question mark take place independent of the command processor. Only when the user deems the command line finished by typing a carriage return does processing pass to the command processor. Likewise, Bill doubts that any significant change would have to be made in the command processor to implement a cue request that could be invoked from any position on the command line.

One problem that Bill's implementation poses, and the same one could be posed by a system used to implement Corbato's second recommendation, is that it requires that input be transmitted one character at a time, a procedure that can be quite expensive on a network. Users would have to be aware of that additional expense when using these help systems. An alternative to this could be an interactive system that processes larger units at one time. For instance, to provide the kind of help contained in the cue request recommendation, we might create a system that responded to a cue request by

prompting the user for the remaining arguments on the command line. If a user asked for help after typing "set_acl path.compout," three prompts would appear:

```
access modes:
User_id:
control arguments (optional):
```

In fact, we might return these prompts anytime the user sends a correct but incomplete command line to the command processor. That would eliminate the need for many error messages that are now returned because of insufficient arguments. To do this we would have to get the network's front end to read a key like ESC as a break key. Otherwise, we would have to use a carriage return both to request help and process the command line, and that could complicate processing from the user's point of view. For instance, if the carriage return is used as a cue, then typing it will sometimes brings back a request for optional control arguments the user has no intention of using, thereby slowing him down. Clearly it would be better to have separate keys for requesting help and sending the line to the command processor. Using such a procedure, we could still return the prompts, rather than an error message, when a user types a carriage return after a command line that requires more arguments.

Something else we might do to improve the ease with which a user enters commands is make parsing of the command line more flexible. That way some errors could be understood by the command processor, and it could proceed to process the command without sending the user an error message. Likewise, when a command's arguments are incorrect, the command could make certain reasonable assumptions about what the user intends and process accordingly. It would, of course, be necessary to tell the user what assumptions are being made and give him a chance to modify them. But that would be superior to sending an error message and thereby forcing the user to figure out his mistake and retype the request. In a way, this flexible parsing acts like a natural language interface. When people talk to one another, they gloss over many imperfections in communication because they assume they know what the other means in spite of the mistake. If they come across a misspelled word while reading, they can often figure out what is meant anyway, whereas the command processor does not understand anything that is misspelled. Flexible parsing would make the command interface act much more like communication between people rather than the current mode of communication between a person and

a machine. There is a very interesting article on this subject by three people from Carnegie-Mellon.(6)

Help With Programming Languages

Originally, I had the idea that we could provide help with programming languages in a manner similar to the cue request on the command line discussed above, something like what we do now for pl1 subroutines in emacs pl1 mode (ESC ^D). This, CISL programmers believe, isn't feasible because the syntax of languages is too varied to provide reasonably accurate and short lists of options.

One simple enhancement would be to provide info segments on programming errors. This would be easy to implement in that we would simply be adding to our info segments. But it would take writers a long time to write infos for all the error messages generated by Multics compilers. Fortran, the compiler I'm most familiar with, has close to 400 error messages when you include runtime io errors. PL1 has at least as many, followed by COBOL, BASIC, and Apl. There will always be times when such information is useful, but I think its value would decrease if we implemented debugging tools like those described below--a program manipulation system or an interactive compiler.

If we were to write info segments that explain programming errors, we could use them to enhance emacs error scan mode. That way, while in error scan mode, a programmer could ask for a more detailed explanation of what a particular error message might indicate. There is, however, a problem with emacs error scan mode that this enhancement will not change. The error scan mode works off error messages generated by the compilers, and these error messages are often inaccurate because of the error recovery procedures that Multics compilers use. So, under these conditions, error scan mode will only be as helpful as the compiler's error messages.

A structured language editor might actually be the logical next step for Multics. Such an editor has been built by IRIA in France, and in fact, it was demonstrated at CISL last year. This system uses a parser to create a syntax tree that can then be

---

(6) Phil Hayes, Eugene Ball, Raj Reddy, "Computers With Natural Communication Skills," Computer Science Research Review, 1979-80, pp. 39-51.

scanned logically. For instance, a programmer can ask to see two logical levels of a program starting from a certain point, and what he will get is an abstracted representation of just that logical portion he's asked for. This system can also check the syntax of lines as they are typed.

Most of the hard work required to implement such a system on Multics has already been done. We have a very suitable editor in emacs; all that would need be done is to create a new emacs mode with a new type of buffer, one that would contain the syntax trees. This would not be difficult and shouldn't take much time. As for the structured language editor itself, LALR has done the hard work of constructing a parser for such a system, plus we might be able to use the IRIA model to work from.

The structured language editor could be an intermediate step toward the state-of-the-art in programming assistance--interactive compilers. IBM has built one of these "check-out compilers," and what it does is compile each line of code as it is typed. Because of the size of its memory, Multics is an ideal system on which to implement such a compiler. In addition, an interactive compiler could replace probe and debug, at least for pl1. It would, however, be a big job, requiring at least seven to eight man-years of work according to Peter Krupp's estimate.

Help With Errors

The explain_error program that is available at MIT could, in an expanded form, provide very helpful user assistance. This program permits a user to follow up an error with a request for further explanation of the error. It also lets the user ask, independently of committed errors, what specific conditions signify. For example, the user could type:

```
explain_error record_quota_overflow
```

to find out about that error. If he had committed that error, he could simply type "explain error," and the program would look up the stack for a frame with an error and print an explanation for it.

At present, explain_error works on system faults only. It does not explain program errors such as:

print: Entry not found. >user_dir_dir>Pubs>Siwila>foo

The explain_error command cannot explain errors like that now because program execution does not stop for such errors. However, because the com_err_ and sub_err_ subroutines raise conditions, the same procedure followed by system faults, it is conceivable that explain_error could be made to explain program errors. It won't do to simply save error code because often another error can slip in before you get a chance to query the first one. We will probably have to suspend execution of the program, the way system faults do, in order to keep track of the condition.

Benson Margulies has proposed a mechanism whereby users would (optionally) be thrown into a little interactive error handler routine when they get errors, where they could ask for more information about the error. This would have the advantages of an improved version of explain_error plus it would allow the user to select the type (new user, programmer, etc.) and amount of information given. This system would require a supporting system that would enable users to set defaults for their environments (e.g., use of parentheses on the command line). But Benson and others have already done some work on the latter, and Benson says that neither is a hard job.

Tutorial Help

The need for tutorial help to teach new users the basics of Multics as well as special subsystems like emacs, qedx, LINUS, compose, and others is obvious. That's why we've written the New User's Introductions and user guides for editors, etc. in recent years. It would be a nice enhancement of online documentation if we could offer such help online, especially if that help could let the user interact with the instruction, that is, practice as he goes what he is being preached. In fact, interactive learning is probably the most effective learning, more so even than a well-written tutorial manual.

The "teach_emacs" tutorial at MIT does just this. This tutorial is unique, however. It is simply a segment which is placed in emacs for the user. The user then performs the various emacs operations that the segment describes. One thing the user

can't do with the segment is write out changes to it. That way the segment is preserved for future use. We may be able to write tutorials like this for the other Multics editors (though programmers at the preliminary design review were skeptical), but we can't write them for things like LINUS, compose, or for the basics of Multics.

There may be several ways to deal with this. We could make the general system help described above tutorial to some extent. This could be done by putting the simpler descriptions of topics ahead of the more technical descriptions. This would be necessary anyway for topics that both new users and non-programmers and experienced programmers might ask about. For instance, the topic "segment" would have to have an explanation like the one in the New User's Introduction as well as a more technical description. A drawback to this approach is that users who don't want the more basic explanation may be forced to see it anyway. The only way to handle that inconvenience is to keep track of the user's experience level and give him the appropriate explanation. In any event, this method of providing help with the basics of Multics would not allow the user to practise in any structured way what is being taught.

Another approach, one that can work with any of the subsystems or subsets of information, is to provide a segment that goes through the basic concepts step by step, without any interactive learning by the user. Mike Auerbach of HIS UK has written such a segment for basic Multics concepts needed by new users. Trouble with this approach is that it remains abstract. The user must remember a great deal of information before getting a chance to apply any of it. The system used for this type of tutorial could be more structured than Auerbach's. It could be constructed hierarchically and enable the user to select the succession of topics from menus. But the fundamental problem of its abstractness would remain.

Probably the best way to write tutorials is to use a subsystem specifically designed for interactive tutorials. Several programmers mentioned Control Data's Plato system. Such a system would cost quite a lot to buy, but no more, probably, than it would cost to build one ourselves. The question then is how much use we'd make of such a system. At this point we can at least estimate how many tutorials we'd want to write. How much customers would use them is something Marketing Education should be better able to judge. We'd probably want tutorials for our commonly used editors--emacs, ted, and qedx--for LINUS, WORDPRO, and read_mail and send_mail if they're not made obsolete by

Executive Mail. We'd also want tutorials for an introduction to Multics and a programming introduction for applications programmers and maybe even one for system administrators. Then too, there is the graphics system and SORT/MERGE.

Help With Documentation

We presently have several info segs that contain information about Multics manuals (manuals.gi, order_manuals.gi, and documentation.gi), but we don't yet provide the user with a way to find out where certain things are documented in manuals or with online descriptions of manuals. Last year Betsy Kerr and I put together a data base that can provide users with this information and two commands for searching that data base--where_doc and explain_doc. These commands were not installed because they used a MRDS data base, and it was felt that we should not offer such a product to MRDS customers only. Now, however, we can package a portion of MRDS separately so that sites that don't have MRDS can buy a particular data base without buying the entire MRDS package. This should allow us to implement where_doc and explain_doc in good faith, and it opens the door for us to put other parts of our online documentation package in a MRDS data base. I have kept the data base up-to-date, so installation could take place in a very short time.

An Outline for an Online Documentation System

To draw a clearer picture of how a help system might appear to users, especially the system documentation component, I'm presenting here an outline I've devised for such a system. It includes cross-referencing, and users can enter it at any point.

Were a user to type "help" without any arguments, the following menu and explanation would be displayed:

```
                        Press F4 key to quit

(1) Help with System Concepts          (5) A Multics Tutorial
(2) Help with Commands                 (6) An Emacs Tutorial
(3) Help with Subroutines              (7) Documentation for Multics
(4) Help with Errors
```

Multics provides online help through "info segments" such as this one. Usually, an info segment comes with a "menu" like the one above. The menu enables you to choose another info segment by typing the number associated with it in the menu. To get the help you are seeking now, type the number of the subject listed above that you would like more help with. Whenever you want to get back to this menu, press function key 2 (usually labelled F2).

The first option in this menu would provide a list of system categories something like the one Lee Baldwin has constructed for commands. The bottom window would contain a revised version of the topics.info segment currently on the system. That would explain how to get topic descriptions by using the help command. It would also have to explain the menu and tell the user how to get to the menu of system categories directly, which would probably be with an argument to the help command (e.g., "help concepts").

The second option in the menu above would provide Lee's categories of commands menu. The bottom window wuld contain a revised version of the modules.info segment, which would explain basic use of the help comand and how to get the command categories menu (e.g., "help commands").

By selecting an option in the concepts menu, the user would get a menu of subtopics for that category along with an explanation of the concept selected.. System concepts would be cross-referenced with commands, so when the user selected one of these subtopics, the ensuing menu would include relevant commands. Similarly, by selecting an option in the commands menu, the user would get a menu of commands classified in that category, along with options for topics of related interest. A request for a command description would then yield another menu, this one containing the major divisions of the description. Lee has already devised such a menu:

(1) All	(5) Arguments
(2) Brief	(6) Control arguments
(3) Syntax	(7) Access required
(4) Function	(8) Notes

The menu displayed here is for the copy command; the menu varies according to the info seg of the command in question.

The Help with Subroutines option in the first menu would produce a menu of subroutine categories similar to the ones for commands and concepts. The bottom window would contain an explanation of how to get subroutine infos from command level. Selecting a category would yield a menu of individual subroutines, and selecting one would give the user a menu composed of entry points.

The Help with Errors option would describe the error reporting system on Multics. Similarly, the other options in the first menu would explain how to get the help indicated. The Multics Tutorial explanation may be accompanied by a menu, depending on just what kind of a tutorial we construct. In any event, users should be able to get to at least the basic tutorial from within this hierarchy of menus. They should not have to return to command level. The Documentation for Multics option would very likely have a menu containing the info segs related to manuals and online user assistance (e.g., where_doc, explain_doc, manuals.gi, help). This selection would also include in the bottom window a general explanation of Multics documentation so that the user could make an informed choice among the options in the menu.

How to Use INTELLECT

From the time Honeywell purchased INTELLECT, online documentation was considered a likely application for the product. In fact, Jim Paradise wanted to use the where_doc/explain_doc data base as the display model for INTELLECT. In one way it makes sense to let users ask for help in the way they know best--natural language. With INTELLECT specifically, however, there are a number of problems. First of all, it is cumbersome to open and thus is most efficient when the user stays in it for awhile. When asking for help, though, the user is usually making only one query and then exiting. If we could figure out a shorthand way of entering INTELLECT and then keeping it open for later queries, while distinguishing between INTELLECT queries and other input, I think it could be practical, at least in that regard.

This spring Steve Herbst and I tried to build a lexicon for the where_doc/explain_doc data base. This data base presented a problem for INTELLECT that all documentation data bases would present--it has field values longer than 80 characters, the maximum length INTELLECT can handle. To deal with this, Steve arranged each field as a separate file. Thus the long fields could be split into separate records arranged in one file. When INTELLECT retrieved one of the long files, such as a manual's description, all the records would come out in the order they were loaded and produce a paragraph of information. The trouble with this approach was that the data base paraphrase of users' requests did not appear logical to the user. Because the file names and field names were redundant, paraphrases could come out in such forms as "full_name full_name" or full_name_rel full_name." Furthermore, each line of of text came out with the manual's order number attached to it because each line was a different record in the file.

Another problem that this application pointed out is that user queries of a documentation data base are too rich semantically for INTELLECT to handle. INTELLECT is keyword-driven, but often, the same word or phrase was needed for several different files. For instance, "tell me about the help command" and "tell me about AG91" would cause ambiguity or worse for the lexicon because in one case "tell me about" is needed to ask about a topic and in the other to ask about a manual. The where_doc/explain_doc data base is relatively limited as documentation data bases go, so another application is likely to be even harder to work out. At this point, I have to say that INTELLECT is not applicable to online documentation.

Recommendations

Based on the literature and what other systems are implementing, I think the top priorities for Multics online documention should be to significantly enhance the system information available and to increase the ease of typing command lines. The first of these will require quite a bit of work on the part of one programmer and two or more writers, but it would satisfy what users probably expect first from documentation--explanations of how to do things with the system. We need not implement single sourcing immediately; that can be done in a second stage. The first stage should include putting text from our manuals into info segments, creating a data base for topics that would cross-reference all of our info segs, and building a menu interface to display subtopics and related commands and facilitate the reading of the info segments.

The second of the top two priorities can be accomplished simply by installing the programs Bill York has already written. This latter is a very important area, perhaps the cutting edge for ease-of-use, so I think we should go beyond what Bill has already done. We should immediately take up Fernando Corbato's second suggestion to provide a means of listing options available at any point in the command line, and we should immediately begin investigating flexible command line parsing. These improvements in command line processing will go the furthest towards facilitating communication with Multics.

As this report indicates, there is much that can be done to improve Multics online documentation with relatively little expense. Much work has already been done; we need only polish it a bit. I recommend that we buy the teach_emacs tutorial from MIT and install it. It's ready to use. Likewise, the where_doc and explain_doc commands are ready to use, and we should take advantage of them.

As for a basic Multics tutorial, I think we should construct a hierarchical system with menus that will lead the user progressively through the information. This would be very easy to program. The biggest job would be for a couple of writers, who might need four or five months to put together and polish all the pieces. It might actually take less time, though, if large portions of the tutorial-like New User's Introduction to Multics (CH24 and CH25) could be used online.

We could also buy the explain_error program from MIT. It needs some work, though, before it can handle the full range of

errors, and we could probably build an interactive error handler with same amount of work. Benson is anxious to work on this, and in this case, I think it would be best to build our own system rather than take up a partially completed one because the end product will be better.

Perhaps the biggest project touched upon in this report is that dealing with programming help. Because so much application programming is done on Multics, I think we should provide more online help for this task. An interactive compiler would be ideal, but it would require an immense amount of work. Multics could, however, be state-of-the-art with a structured language editor, and we may be able to build such an editor in a fairly short time.

Currently, Multics offers quite a lot of information online. What the proposals outlined in this report would do is increase that information and make it much easier to get. Since ease-of-use is a crucial concept in computing now, I think we could make Multics an even more attractive system by implementing as many as possible of the online documentation facilities discussed in this MTB.