Technical Report 71-108 May 1971

The GENREL Teletype Package

by

Brian K. Reid

UNIVERSITY OF MARYLAND
DEPARTMENT OF PHYSICS AND ASTRONOMY
COLLEGE PARK, MARYLAND

Technical Report 71-108 May 1971

The GENREL Teletype Package

by

Brian K. Reid

The development of the GENREL teletype package was supported in part by the National Aeronautics and Space Administration Contract #NAS 9-5886 and in part by the National Aeronautics and Space Administration grant number NsG-398 to the Computer Science Center of the University of Maryland.

Although the UNIVAC 1108 system under EXEC 8 is capable of processing both batch and teletype jobs, most of the software and application programs available are oriented more towards batch processing. These programs will execute from a teletype, but their use is generally less convenient in the demand mode.

The GENREL Teletype Package is a collection of programs designed specifically for use on a teletype (or similar conversational time-sharing terminal). They are not restricted to conversational usage, but will generally be less convenient to use in the batch mode, much in the same way that it would be less convenient to use the standard collection of batch programs on a teletype.

Some of the criteria which were kept in mind when designing these teletype-oriented programs were:

--They should be "conversational" in operation, i.e. they should not expect the user to enter unsolicited data images, but will ask for each piece of information as it is needed.

--The printed output generated by each program should be minimal. The wording of all output will be kept terse and to the point.

--Any bulk output to be generated by the program should be entered into a data file so that it may be selectively examined by the text editor or printed in toto onsite.

--Each program should print a sign-on line and a sign-off marker to better enable the user to follow the progress of his programming.

--Where applicable, the program functions are controlled by calling option letters on the program execution card, so that the user may use the programs to perform those functions and only those functions in which he is interested.

Access

All of the GENREL Teletype Package programs are designed to be run from the user's temporary program file (TPF$), except those which are relocatable library subroutines. A user must first load his TPF with a copy of the programs before he may use them; this is done with a COPY statement to copy the library file into his own file:

```
@COPY,P  GENREL*TP.
```

The programs are stored in a permanent file whose name is "TP" (for Teletype Package), and this file is catalogued under the qualifier GENREL. It is a public, read-only file, which means that its contents cannot be changed without destroying and re-building the file. This file should never be referenced in an assign (@ASG) statement, as the @COPY,P program will assign the file and free it as needed.

A number of the programs in the Teletype Package will not work unless executed from a temporary program file; for this reason it is not a good idea to copy the Teletype Package anywhere else except to TPF$.

The relocatable elements cannot be stored in GENREL*TP, because in general the collector will include all relocatable elements from TPF$ into every collection, regardless of whether they are needed or even wanted. The relocatable elements which form part of the GENREL Teletype Package are stored in a different file, the relocatable library, GENREL*RLIB. All that need be done to access these subroutines is to insert the card

```
LIB  GENREL*RLIB.
```

into the set of collector commands. The MAP processor will do the rest.

Feedback

If any difficulties should arise or errors be detected, please direct complaints to

Brian K. Reid
General Relativity Group
Physics Department
University of Maryland
College Park, Maryland 20742

Acknowledgements

Although the majority of the programs in this Teletype Package are from the General Relativity research group of the University of Maryland Department of Physics and Astronomy, some are from other places and have been included here out of convenience to the teletype user. All were produced at the University of Maryland.

For FILES, INSERT, AND RINGTEST, we thank Mr. Hans J. Breitenlohner of the Computer Science Center research staff. For EDIT, we thank Mr. Kern E. Sibbald of the Computer Science Center Systems Staff. For CSF, thanks to Mr. Jeffrey S. Jewett from the High Energy Physics group.

The Teletype Package project was conceived, edited, and acid-tested by Dr. Jerry Larson of the General Relativity group.

Summary

This chart gives a quick overview of the programs in the GENREL Teletype Package. All programs are described in detail on the pages to come: this section is intended as a reference or as a table of contents.

Program Name	Brief Description
CATALOG	List names of user cataloged files
COMM	Allows teletype-to-teletype communications.
COMMENT	Prints commentary from @ADD elements
CPMD	Conversational Post Mortem Dump
CSF	Edits read and write keys out of assign images
DRUMPLOT	CALCOMP package modified for teletype usage.
EDIT	Improved text editor
FILEDIT	Generalized conversational absolute editor
FILES	Prints a table of the user's assigned files
GARBLE	Scrambles a symbolic element for security
INSERT	Reads paper tape into a program file
ISTTY	FORTRAN subroutine to determine batch/demand mode
MIMIC	Conversational MIMIC processor.
OPT	Recovers options from @XQT card from FORTRAN
READY	Signals end of block of control images
RINGTEST	Determine presence of a ring in a tape.
SENTINEL	Allows teletype to monitor a batch run
TAPE	Assigns, switches, and labels tapes
TIME	Prints current and elapsed CPU and memory time
TOCGEN	Table-of-contents generator. Analyzes program files.
TTPLOT	High-efficiency auto-scaled teletype line plot
WHO	Identifies teletype site code and channel/unit

CATALOG -- List Table of Catalogued Files.

The CATALOG processor allows a teletype user to print a table of catalogued files catalogued under his project number and/or account number and/or qualifier. At the time of this writing, the EXEC 8 file security is minimal enough to permit any user to list a table of anyone elses' files, regardless of whether or not they are private. This situation will probably change in the not-too-distant future; any tightening of file security will reduce the capability of the CATALOG program in printing other peoples' files, but not ones' own.

The basic processor call to the CATALOG program is simply:

@CATALOG, options [keyword]

The [keyword] field is a sort key: all files which match the keyword will be listed. If, for example, the keyword is an account number, then all files which were created under that account number will be listed. It may be an account number, a project, or a qualifier.

From this point, the best documentation is probably by example.

Processor call	Action Taken
@CATALOG, P PROJ	List all files catalogued under project "PROJ"
@CATALOG, Q PROJ	List all files with qualifier "PROJ"
@CATALOG, N ACCTNO	List all files created by account no. "ACCTNO"
@CATALOG QUAL*FILE.	List all cycles of the requested file.
@CATALOG, P	List all files with the same project as this run.
@CATALOG	List all files with the same account number as this run.

Each time the CATALOG processor is called, it must obtain a copy of the master file directory from the system. This can be rather slow. In order to facilitate multiple calls to CATALOG, two options have been added to speed things up. They are:

D — Create a copy of the master file directory in a drum file for later use.

W — Use a previously stored copy of the master file directory instead of getting a new one from the system.

The sequence of using these options for multiple calls to CATALOG is:

```
@CATALOG,D
@CATALOG,W
@CATALOG,W
  . . .
@CATALOG,W
```

COMM -- Teletype-to-teletype communications.

It is occasionally useful to be able to leave a message for another teletype user, or to have one left for you. Ideally, this would be performed by the executive system, with provisions for message storage, duplication, broadcasting, recording, etc. However, this not being the case at this time, a less sophisticated technique may be used.

The COMM program maintains a catalogued file (the same one, in fact, which is used by the SENTINEL processor) to store a message. When a user executes the COMM program, it will print out the contents of the file, and then during execution it will print out the new contents each time the file is changed.

To execute:

```
@XQT COMM
```

No options. A sign-on line will be printed, then the current contents (if any) of the communications file. To enter something into the file, just type. Anytime anybody else types something, it will be printed at your teletype. No more, no less.

COMMENT -- Print comments from an ADD file.

It is convenient to be able to assemble a package of control statements and data images into elements which will be added to the run stream via the @ADD statement. It is equally convenient to be able to follow the progress of the statements in the added element; the executive currently provides no means of doing this. The COMMENT program provides a satisfactory, although not ideal, solution by allowing the added element to print a running commentary of what it is doing. The program itself is absurdly simple, it is in fact included more as a suggestion of teletype technique than as a valuable piece of software. To use, simply insert, as frequently as desired, the following sequence of images into the @ADD element:

```
@XQT COMMENT
line image
line image
line image  (etc.)
@EOF
```

Each image will be printed exactly as it stands, with no adornments or trappings.

As a utilitarian example, consider the following un-narrated sequence:

```
@ELT,DI F.MAP/APEX
@PREP F.
@RALPH,RN PROG
@MAP,NI  ,PROG
 IN .PROG
 LIB F.
 LIB LIBR.
 CLASS APEX
 END
@XQT COMMENT
APEX VERSION READY AS TPF$.PROG
@EOF
@END
```

CPMD -- Conversational Post Mortem Dump

Since the dawn of computer technology, the major weapon in the arsenal of debuggery has been the Post Mortem Dump: a complete labelled printout of the entire contents of the computer's memory, taken just after the termination (normal or otherwise) of the program being debugged. The very thought of a full core dump sends the average teletype programmer screaming for aspirin, yet no reasonable alternative has ever really been devised.

The GENREL Conversational PMD is a program designed to allow the teletype user all of the conveniences of a full core dump (and then some), with none of the obvious drawbacks. For example, most core dumps are taken in octal (or hexadecimal, or whatever), but most of the real information to be gleaned from a dump can be gotten only be converting the octal numbers into another format. The CPMD program will automatically convert to any one of a number of useful formats, thus saving the tired programmer the work of conversion.

When a program terminates on the 1108 system, the final contents of its memory are written to a drum file whose name is 'DIAG$'. The CPMD program allows the teletype user to selectively examine the contents of this file (which is equivalent to selectively examining a core dump).

To initiate the CPMD program, one need only type:

```
@CPMD
```

with no options or fields. The program will respond with:

```
GENREL CPMD LEVEL x
IBANK: xxxxxx TO yyyyyy
DBANK: xxxxxx TO yyyyyy
```

The 'IBANK' and 'DBANK' numbers are the lower and upper limits respectively of the program's instruction and data banks. The word '(NONE)' will be printed instead of the limits if one bank does not exist. CPMD will next ask the question

```
FUNCTION?
```

and wait for an answer. The rest of this description details the various possible answers to the question.

Notes:

In the VARIABLE dump mode, CPMD operates in conjunction with the RALPH compiler to allow the teletype user to dump program variables by name rather than by address. Any element which was compiled by RALPH using the 'D' (diagnostic) option will contain the necessary symbol table, and may be dumped in VARIABLE mode.

Short Table of Conversational Post Mortem Dump Commands

Command Name	Purpose and short description
OCTAL) ALPHABETIC) FLOATING) INTEGER) PROGRAM)	These five commands are the 'output' commands of the CPMD. They will print the designated core locations in the desired format. The calling sequence of each output command is the same: command start, count command start/lcc, count The first form dumps 'count' words starting at absolute address 'start' in the format specified by 'command'. The second form dumps 'count' words starting at relative location 'start' control counter 'lcc' in the deck last specified in a 'DECK' command.
DECK (name)	This command specifies the deck to be used as the base of relative addressing, and also as the deck for variable-name dumping (see below).
MAP LMAP MAP (name)	These commands list the location-counter allocation information that would have been printed by the collector as a storage map. The 'MAP' command prints a map of all user subroutines. The 'LMAP' command prints a map of all library subroutines. If a name (deckname) is specified on the MAP command, then it will print out the allocation information for that deck only.
VARIABLE	Causes CPMD to enter VARIABLE-name print mode. Dumps by symbol name rather than by location.
LOCATION	Leave VARIABLE mode and enter normal (LOCATION) dump mode.
DIMENSION a,b,c,...	Specify (for use with the SUBSCRIPT command) a dimension vector.
SUBSCRIPT a,b,c,...	Calculate a multiple subscript from dimension information.
IDENTIFY n	Translate the absolute location 'n' into a relative location with deck name.
FIND x	Search core for the next occurrence of 'x'.
LIMITS m,n	Specifies search limits for FIND command as "m" to "n".

Command	Description
OCTAL	The OCTAL command causes CPMD to print out the contents of selected cells of memory in an octal format. Four words of twelve digits each are printed on each line. The format is 'OCTAL m,n', where 'm' is the starting address and 'n' is the count of words to be dumped. If the field 'n' is omitted, it will be taken as 1. The field 'm' is always treated as octal, regardless of the presence or absence of a leading zero. The address 'm' may be relative or absolute; see the section entitled 'RELATIVE ADDRESS SPECIFICATIONS' for a description of relative addresses.
ALPHA	The ALPHA command causes CPMD to print out the contents of selected cells of memory in an alphabetic format. Eight words of six characters each are printed on each line. The format is 'ALPHA m,n', where 'm' is the starting address and 'n' is the count of words to be dumped. If the field 'n' is omitted, it will be taken as 1. The field 'm' is always treated as octal, regardless of the presence or absence of a leading zero. The address 'm' may be relative or absolute; see the section entitled 'RELATIVE ADDRESS SPECIFICATIONS' for a description of relative addresses.
FLOATING	The FLOATING command causes CPMD to print out the contents of selected cells of memory in an edited floating-point format, five numbers per line. The calling format and restrictions are exactly as described above for OCTAL and ALPHA.
INTEGER	The INTEGER command causes CPMD to print out the contents of selected cells of memory in a base-10 integer format. Five numbers per line are printed. The format and restrictions are as described above for OCTAL and ALPHA.
PROGRAM	The PROGRAM command causes CPMD to print out the contents of selected cells of memory in a reconstructed assembly language format. Operation mnemonics and register names are printed. The format is one instruction per line. When relative addressing mode is used, all u-field addresses printed which reference the same relocatable element that contains the instruction are un-relocated and printed as "address/lcctr" rather than "address".

Command	Description
MAP	The MAP command causes CPMD to access the diagnostic tables in the absolute element from which the program was executed and from these tables to reconstruct the storage allocation map which resulted from the program collection. Only user-generated subroutines and the main program will be listed: no information about library and system subroutines will be printed. If a deck name is specified on a MAP command, then only the allocation information for that deck will be printed.
LMAP	The LMAP command is identical to the MAP command except that it prints all subroutines, both user-generated and system library-provided.
DECK	The DECK command is used to specify a deck name for use as the base of relative addressing. (See the section, following, which describes relative addressing). All relative addresses are taken as relative to the specified deck name. The call is simply "DECK deckname", where "deckname" is the 1-to-12 character name.
IDENTIFY	The IDENTIFY command causes CPMD to search its diagnostic tables in an attempt to determine the deck, location counter, and relative address corresponding to the specified absolute address. The call is "IDENTIFY nnn", where "nnn" is an address (octal). If the address "nnn" is in the program being dumped, then CPMD will print out the corresponding deck name, location counter, and relative address, otherwise an error message.
LIMITS	This command is used to specify the area to be searched by the FIND command. The call is "LIMITS aaa,bbb", where "aaa" is the first address to be examined and "bbb" is the last address to be examined. Both addresses are octal, and must be absolute addresses.
FIND	The FIND command causes CPMD to search the specified region of core for a given word or character string. The call is "FIND xxx", where "xxx" is one of the following: octal integer, decimal integer, floating point number, alphabetic string of up to six characters. Alphabetic strings must be delimited with quotes ('). All searching is done on a fullword basis; no partialword or sliding-character searches will be made.

Command	Description
ADD	The ADD command causes CPMD to add a list of numbers and print the sum in both octal and decimal. It may be used for subscript calculations, link tracing, etc. If only one number is provided to be added, it will be printed out in both octal and decimal, thus serving as a converter.
DIMENSION	The DIMENSION command is used to specify dimension information for use with the SUBSCRIPT command. The call is "DIMENSION aa,bb,cc, . . .", where "aa" etc. are decimal integers.
SUBSCRIPT	The SUBSCRIPT command computes multiple subscripts for FORTRAN variables. Supposing, for example, that one wanted to dump VAR(4,2,1), where VAR had been dimensioned at (5,6,7). The statement DIMENSION 5,6,7 would enter the appropriate dimension information. Then, typing SUBSCRIPT 4,2,1 will cause CPMD to print the equivalent linear subscript. Using the ADD command to add the base address to the linear subscript will give the address to dump; or this linear subscript may be used directly with the VARIABLE dump mode.
VARIABLE	The VARIABLE dump mode allows the CPMD user to dump RALPH-compiled programs by variable name instead of by address. The only requirement is that the program being dumped must have a symbol table. This symbol table will be generated automatically by RALPH if the 'D' (diagnostic) option is present during compilation. To enter 'VARIABLE' dumping mode, type "VARIABLE". To leave 'VARIABLE' dumping mode, type "LOCATION". After entering VARIABLE dumping mode, a deck name to be dumped must be specified. If the deck does not contain a symbol table, it will not be accepted. Once a deck has been specified, then the dumping commands will expect variable names (with possible subscripts) instead of addresses. The calling format is (command) NAME,count (command) NAME(subscript),count This will cause CPMD to dump, in the format dictated by (command), "count" words starting with NAME or NAME(subscript). For example, to dump the variable "I" as an integer and then X(I) as floating point, type: INTEGER I FLOATING X(n) Where "n" is the number which was printed as the contents of "I".

RELATIVE dumping mode.

If the address portion of a dump command (described as "m" in the descriptions of the dump commands) is coded as "address/counter" instead of just "address", then the address will be interpreted as being a relative address relative to the specified location counter in the deck declared in the last DECK command. For example, supposing that there existed a deck 'UUU' which contained data in location counter 3 from addresses 013044 to 023266, then the two following commands would print identical values:

```
OCTAL 13050,2
OCTAL 4/3,2
```

Thus location 13050 is relative location 000004 on control counter 3 in the deck under study, "UUU".

When dumping in PROGRAM format, any addresses printed as U-fields will be printed as address/counter with relative address value if they refer to the deck which is being dumped.

CSF -- Control Statement Processor

The EXEC 8 system as defined provides for security keys of up to six characters each for use with catalogued files. The 'read key' must be specified on an assign statement before a file may be read, and the 'write key' must be specified before it may be written or deleted. Unfortunately, any batch run which contains assign cards specifying keys will list out all of the cards, including the assign card. Thus, each time you run a batch job which assigns a file with keys, you are displaying your keys for all the world to see. A lot of good keys are!

The CSF program accepts modified assign/catalog statements as input, and produces a batch listing with the read/write keys replaced by a string of asterisks.

For example, suppose that one were to assign the file 'GLURK' with its read and write keys. The assign image as printed on a batch listing would be

```
@ASG,A GLURK/READ/WRITE
```

Anyone passing by would be able to look at the listing and see that the keys were "READ" and "WRITE". To prevent this sort of thing from happening, substitute the sequence

```
@XQT CSF
ASG,A GLURK/READ/WRITE
```

The onsite listing will be:

```
@XQT CSF
ASG,A GLURK/*****/*****
```

thereby leaving everyone in the dark as to what the keys are. Notice that column 1 of the assign card image has been left blank. If the letter 'N' is punched in column 1, then nothing will be printed. The card will sneak through undetected. This effect may also be had by placing the 'N' option on the XQT card: @XQT,N CSF.

DRUMPLOT -- Teletype-oriented CALCOMP plot system.

Normally, the use of the CALCOMP plotter requires that the program write directly to a plot tape, which may not be rewound or otherwise edited, and which must stay mounted, thus tying up a valuable tape unit. The GENREL package for the CALCOMP plotter consists of two parts:

1) A version of the CALCOMP subroutines which records its output in a program file instead of on tape.

2) A program (DRUMPLOT) which will transfer the plot from the program file to the tape.

Plots are stored in program files as symbolic elements. They may be copied, deleted, and so on: they may be handled in the same way as any other symbolic element. It is not recommended that one try to edit a plot element, as its contents look like complete gibberish when interpreted as print images.

The modified CALCOMP package to transfer plots to a drum file is contained as a relocatable element in the file GENREL*RLIB. This program will behave in the same way as the standard CALCOMP package when run as a batch program, but when run from teletype, the first call to any CALCOMP subroutine will produce the question

ENTER PLOT FILE.ELEMENT/VERSION-----

to which you should respond with a FILE.ELEMENT or FILE.ELEMENT/VERSION name. It is not recommended that plot elements be stored in the same file as other programs, because they tend to be rather large and can easily exceed the maximum allowed file size of a catalogued file. It is best to store the plot elements in a temporary file.

If, at any point during the execution of the program, it is desired to terminate a plot element and begin another, simply code:

CALL NEWPLT

which will re-ask the question to enter the plot file and element name. If the program terminates in error, the plot element will be closed and a 999 block written automatically at the end.

To transfer a plot to a tape named PLOT-TAPE (which will be assigned by the program if none is present), one need only type

@DRUMPLOT file.element/version

The DRUMPLOT program will copy the requested plot to tape. If another call to DRUMPLOT is made, the second plot will be adjoined to the first one, etc.

EDIT -- New Improved Text Editor

There exist several versions of an "improved text editor" for the 1108 EXEC. No two of these seem to be completely compatible. This editor is a re-worked compendium of all of the juicy extras which anyone has seen fit to put into an 1108 text editor, along with some tidbits stolen from the GE-600 TSS II editor, along with some good old-fashioned innovations.

If you are familiar with the operation of a text editor, please skip the remainder of this page.

Theory of Operation of a text editor.

A text editor operates in three distinct phases:

1) Copy the text to be edited into a scratch file.
2) Edit the text.
3) Copy the edited text back into the original program area.

Because all editing is performed on a scratch file, the user may at any time abort the editing and start over again without harming his original program. This is accomplished by bypassing step 3, and is caused by typing the command 'END'.

During editing, the text may be thought of as a long vertical list of statements, with a cursor which moves up and down the "stack". The "top" of the stack refers to the first statement in the text, and the "bottom" of the stack refers to the last. The line to which the cursor points at any given time is referred to as the "current line". The process of editing consists of moving the cursor up and down the stack and altering, deleting, or inserting images into the stack. All changes to the stack are made at the cursor, so that it is relatively easy to keep track of what is going on.

There are two basic "modes" of editing with the editor. The first, "EDIT" mode, is the normal mode. The second, "INPUT" mode, is used for typing in straight text; i.e., for using the editor as a sort of keypunch. In "INPUT" mode, the editor takes each line exactly as typed and inserts it into the stack after the current line. Any image save one which has a master space in column 1 (@) may be entered in "INPUT" mode. To leave or enter INPUT mode, type a null line (carriage return only).

The editor is called via the statement

```
@EDIT,options  spec1, spec2
```

where "options" is the string of option letters, "spec1" is the input specifier, and "spec2" is the output specifier. If either specification is of the form (name) followed by a period, it will be taken as a data-format file rather than as a program file. These may be mixed--i.e., the input may be from a program file and the output to a data file, or vice versa. A name with no punctuation is considered to be an element in TPF$.

The option letters available and their meaning are listed below.

Option	Meaning
U	"Update"--Specification 2 is not specified, and the output of the editor overwrites the input, producing the next cycle. For example, @EDIT,U GP.LIN will edit the element "LIN" in the file "GP", and then at editor termination will write the edited text back into file "GP", calling it "LIN" with a cycle level one higher. If the specification field is a data-format file, the editor will, in the presence of a "U" option, first try to assign the next higher file cycle of the specified file. If it does not exist, then it will overwrite the existing file with the edited contents.
I	"Input"--The editor needs only a Spec1 field, and creates it from scratch. The editor will start off in "input" mode, but may be toggled back and forth to EDIT mode in the standard manner.
R	"Read-only"--The editor bypasses step (3) as listed on the previous page, which means that it will produce no output. The "R" option is used for examining files or elements without the danger of altering them.
O	This option causes the editor to use permanent files for its internal scratch files. Thus, if there is a system failure while in the midst of a long edit run, you may pick up exactly where you left off after the system has been rebooted. The mechanism is as follows: When the "O" option is specified, the editor looks to see whether or not there exist catalogued files with the correct name ("EDITA" and "EDITB"). If these files exist, then step (1) of the editing process as described previously is bypassed, and editing resumes on these permanent files. If however, these filed do not exist, then they will be created by the editor. As editing is terminated, they will be decatalogued, but only after the output has been safely filed away. CAUTION: Permanent files are SLOW.
E	Designed for batch program use. Causes the editor to print an echo of each command before it is processed.

Where no options have been specified, the editor will produce Spec2 from Spec1. The output element Spec2 will have cycle level 0 regardless of the cycle level of Spec1.

The next two pages contain a quick summary and table of contents of the editing commands available. For descriptive purposes only the editing commands have been grouped into four groups:

1) Normal editing commands
2) Block editing commands
3) Editor mode-control commands
4) Miscellaneous commands

Normal Editing Commands

TOP	Move the cursor to line zero.
HEAD	Move the cursor to line one.
BOTTOM	Move the cursor to the last line and enter INPUT mode.
NEXT n	Move cursor by 'n' lines, forward or backwards depending on the sign of 'n'.
BACK n	Move the cursor backwards 'n' lines.
UP n	Move the cursor backwards 'n' lines.
PRINT n	Print 'n' lines including the current line.
OUTPUT n	Print 'n' lines not including the current line.
NPRINT n	Print 'n' lines including the current line, with line no.
NOUTPUT n	Print 'n' lines excluding current line, w/line number.
DELETE n	Remove 'n' lines starting with current line.
INSERT string	Insert specified character string as an image in the text.
IB string	Insert 2 line before the current image.
RETYPE string	Replace the current image with the specified string.
LOCATE string	Locate the image which contains the specified string.
FIND string	Locate the next image with the specified string in the first columns.
NLOCATE string	Locate command with line number prefix on print.
NFIND string	Find command with line number prefix on print.
GOTO n	Move the cursor to line 'n'.
CHANGE	Perform character editing on text images.
HOLD	Make a copy of the current image for use with the DUP command.
DUP n	Insert 'n' copies of the held image after the current line.
DHOLD	Same as "HOLD" except it deletes after holding.

Block Editing Commands

BEGIN	Declare beginning of block to be edited.
SPLIT f.e	Move a block of code to specified file.element and delete.
COPY f.e	Move a block of code as specified, without delete.
ADD f.e	Begin editing from specified external block of code.
STOP	End editing from specified external block of code.
CLOSE	Copy all of ADD block into edit file and leave ADD mode.

Miscellaneous Commands

TIME	Print header line with signon time.
SEQUENCE	Insert card sequence date in columns 73-80.

Editor Mode-Control Commands

ALLOW n	Allow 'n' columns of editing area, where 0 n 133.
MARGIN n	Print only 'n' columns, regardless of the image size.
SAVE n	Do not alter text past column 'n' with the CHANGE command.
TAB x	Declare the character 'x' to be a tab character.
SET a,b,c,...	Set tab stops in columns a, b, c, etc.
BRIEF	Do not print the new current image whenever the cursor moves.
VERIFY	Print the current image each time the cursor moves.
SQUEEZE	Enter squeeze-print mode. Compress out multiple blanks.
EXPAND	Leave squeeze-print mode. Print all blanks as they occur.
LINE	Print the current line number.
END	Abandon text editing, release all edit files. Do not save results of editing.
TYPE typ	Declare the element type of the output element (ASM, FOR, etc.)

Notes on using a text editor:

It is useful to note that the text editor provides several methods to recover from some stupid blunder in editing. For example, suppose that you are editing along and delete a block of 8 lines, but accidentally type 'DELETE 88' instead. Chances are, your program will not be working up to snuff at that point, unless you are very very prolific with comment cards, you have probably deleted a large part of the program. If you realize your mistake right away, you may merely type 'END', and the editor will go away, leaving your original element unharmed. If you do not notice the mistake until later, when you try to compile the program and get hundreds of error messages, you may recover by going back to edit from the previous cycle. If you did not do a U-option update, you are in trouble. Moral: save your skin by always doing a U-option edit.

The 'UP' and 'BACK' commands are relatively inefficient; they work by recording the current cursor position and then performing the sequence 'TOP', then 'NEXT n'. It has to be this way because of the structure of the standard-data-format intermediate files. When the system is heavily loaded down, it might be judicious to use as few upwards-moving commands as possible.

If the editor is busily printing away and you want to stop it, you may use the 'BREAK' key in the following way: Hit the 'BREAK' key on the teletype. The computer will answer with 'INTRPT LAST LINE', which means that you have interrupted it in the middle of printing the last line. It pauses. Type a carriage return. Nothing else. The computer will re-print the previous line, print three or four more, then print EDIT. You are now safely out of the print-loop and still editing. Be more careful next time.

Where applicable, the 'FIND' command should be used, as it is much faster than the 'LOCATE' command. For example, if you are going to move the editor to a statement label halfway down the text, use 'FIND' instead of 'LOCATE'. It will use much less computer time and also make your wait time less. (You rub my back and I'll rub yours.)

TOP T	This command moves the cursor to line zero. This line is a dummy line kept by the editor which is always above all text lines. In this way, one may insert before the first program image by inserting after the zero image.
HEAD H	This command moves the cursor to the first line of the text. It is equivalent to the pair of commands "TOP" then "NEXT".
BOTTOM B	This command moves the cursor to the last line of the text and then enters INPUT mode. In this way, one may easily append lines to the end of an existing program.
NEXT n N n	This command will move the cursor over "n" lines of text. If n is positive, the cursor will be moved forward. If n is negative, the cursor will be moved backward. If n is zero or not given, it will be taken as one and the cursor will be moved to the next image. If a move backwards attempts to move the cursor past the beginning of the text, it will be moved to line one instead. If a move forwards attempts to move the cursor past the end of the text, it will stop on the last image in the text and the message "n LINES TALLIED AT EOF" will be printed, where n is the number of lines which were actually skipped before the end of text was encountered.
BACK n UP n	This is equivalent to NEXT -n, and moves the cursor back by n lines.
PRINT n P n	This command prints "n" lines of text starting with the current line.
OUTPUT n O n	This command prints "n" lines of text starting with the image after the current line.
NPRINT n PN n	This command prints "n" lines of text starting with the current line, but prefixes each line with its line number.
NOUTPUT n ON n	This command prints "n" lines of text starting with the line after the current line, but prefixes each line with its number.
SQUEEZE	This command causes all printed lines to have multiple blanks compressed out of them before they are printed.
EXPAND	This command restores printing to normal un-squeezed mode, with all blanks printed as they occur.
DELETE n D n	This command deletes "n" images starting with the current image. After execution of the DELETE command, there is no current image, a "NEXT" or "UP" command must be executed to define a current image.

INSERT string I string	The specified string of characters, starting with the first character after the first blank in the typed line, is inserted into the text as a line, its position being just after the current line. The cursor is moded to the new line. If "string" is not specified, this command will cause the editor to enter INPUT mode.
INSERT,m string I,m string	The string is inserted into the text as described above, except that it is inserted starting in column "m".
IB string	This command is the same as the INSERT command, except it inserts before the current image.
RETYPE string R string	The current line is replaced with the specified string. This command is equivalent to the sequence "DELETE" then "INSERT string".
RETYPE,m string R,m string	The specified string is inserted into the current image starting in column "m". The remainder of the current image is left intact.
LOCATE string L string	This command causes the editor to search for the next occurrence of the character string "string" in the text after the current image. Blanks are ignored, so that a command "LOCATE X Y Z" would treat "X Y Z" as a valid occurrence. If there is a blank in the string to be located, then it must be matched by at least one blank in the text string, but any number of blanks in the text string will not affect the find. The cursor will be left at the located line. If the end of the text is reached before the required string is located, the message "END OF FILE ENCOUNTERED." will be printed and the cursor left at the bottom of the text. If, however, another LOCATE or FIND command is given immediately, an automatic TOP command is performed first, so that one need not keep restoring the cursor to the top for multiple searches.
LOCATE,m string L,m string	This command is identical to the LOCATE command, save that it will locate the next "m" occurrences of the string and position the cursor on the last one. Lines printed will be prefixed with a line number.
LOCATE,* string L,* string	This command will locate all occurrences of the specified string and leave the cursor at the end of file. Lines printed will be prefixed with a line number.
FIND string F string	This command causes the editor to search for the next occurrence in columns 1 through n, where n is the length of the specified string. Unlike the LOCATE command, the FIND command treats blanks as characters.
FIND,m string FIND,* string F,m string F,* string	These forms of the FIND command are identical to the same forms of the LOCATE command as far as the meaning of the "m" and the asterisk. See above for this description.

NLOCATE string
LN string

This command is identical to the normal LOCATE command save that it prefixes the line with a line number before it prints it.

NFIND string
FN string

This command is identical to the normal FIND command save that it prefixes the line with a line number before it prints it.

GOTO n
GO n

This command moves the cursor to line number "n". If "n" is larger than the number of images, then the cursor will be left at the bottom of the stack and there will be no current image.

CHANGE /S1/S2/
C / S1/S2/

The CHANGE command allows the edit user to replace a portion of the current image and/or following images with a substituted character string. In its simplest form, it simply replaces the first occurrence of the character string "S1" with the character string "S2". In the sample commands at the left, the virgule (/) has been used as a string delimiter. Any character may be used for this purpose as long as neither of the strings S1 or S2 contains that character.

As an example of the use of the CHANGE command, suppose that the current image is the line:

HORSES HAVEN'T ELEVEN FEET.

We shall enter the change command:

CHANGE /HAVEN'T/HAVE/

And the line will now read:

HORSES HAVE ELEVEN FEET.

Once more, the line

CHANGE *ELEVEN*FOUR*

will cause the current image to read:

HORSES HAVE FOUR FEET.

Notice that the characters to the right of the changed portion of the string were moved in as needed.

If the string S2 is null, then the string S1 will be deleted from the image. If the string S1 is null, then the string S2 will be inserted before the first character of the line.

CHANGE/S1/S2/G C /S1/S2/G	This form of the CHANGE command causes all occurrences of the string S1 to be replaced with the string S2 in the current image. The big "G" does not stand for Goodness (as the Cheerios folks would have us believe), but for Global.
CHANGE /S1/S2/n C /S1/S2/n	This form of the CHANGE command operates on more than one image--on "n" of them, to be precise. The editor will search the next "N" images starting with the current line for "S1", and will replace the first occurrence (if any) of "S1" in each line with "S2". The final position of the cursor will be the same as would have been caused by a "NEXT n-1" command. If the letter "G" is appended to the change command in this form, then all occurrences instead of the first occurrence of S1 will be changed in each of the "n" images.
LINE	This command will tell you the line number on which the cursor is resting--i.e. the line number of the current line.
TAB k	This command declares a tab character for use with the INSERT and RETYPE commands and the INPUT mode. The tab character may be any character except a blank; the tab stops are set by the SET command.
SET n1, n2,n3...	This command sets the tab stops for the specified tab character. If not set, there will be three stops set in columns 11, 21, and 39 assumed by the editor.
BRIEF VERIFY	These commands control the print/no-print option for cursor control. In VERIFY mode (entered via the VERIFY command), the current line is printed whenever it is changed or the cursor is moved. In BRIEF mode, only the commands which specifically print (PRINT and OUTPUT) will cause text lines to be printed. Until directed, the editor is in VERIFY mode.
HOLD	This command causes the editor to make a copy of the current image and hold it for use with the DUP command. Each occurrence of the HOLD command overrides the previously held line, if any.
DHOLD	This command is like HOLD, but deletes the current image after holding.
DUP DUP n	The DUP command causes the editor to insert a copy or "n" copies of the held line (held by the HOLD command, above) into the text after the current image. If there is no held line, a warning message will be printed.
ALLOW n	This command declares the width of the editing area in columns. The value "n" must be between 2 and 132 inclusive.

MARGIN n PL n	This command declares the width of the print area. The number "n" is the margin, or print limit. No characters past column number "n" will be printed under any circumstances until the print limit is changed.
SAVE n	The Save command declares the width of the editing area which is succeptible to the CHANGE and RETYPE, m commands, and to being searched by the LOCATE and FIND commands. When altering the current image, no characters past the save margin will be altered. For example, of one were editing an assembly program and wished to keep the comments in columns 39 to 80 unchanged, then a "SAVE 39" command would do the trick. When using the LOCATE and FIND commands, a character string will not be found if it occurs past this margin.
SPLIT file.element SPLIT file.	The SPLIT command moves a block of the program to another file or element. The execution of a SPLIT command causes all images from and including the line specified by a BEGIN statement (below) to but not including the current line to be removed from the edit file and transferred to the specified file and element. If no element is specified, the split will be taken in data-file format. The first image transferred is the image specified by the BEGIN command; the last image transferred is the image before the current image at the time of execution of the SPLIT command.
BEGIN	The BEGIN command declares the first image to be transferred via a SPLIT or COPY command. If none has been specified, line 1 is assumed.
COPY file.element COPY file.	The COPY command is identical to the SPLIT command except that it does not delete the original from the edit file.
SPLIT,n f.e or f. COPY,n f.e or f.	This form of the SPLIT and COPY commands redefines the BEGIN pointer at line number "n" and then procedes as usual with the execution of the SPLIT or COPY.
ADD file.element ADD file.	The ADD command causes the editor to temporarily use an external block of images as part of the edit stack. The execution of an ADD command followed by a NEXT command will cause the cursor to rest on the first image in the element or file which is being added. As the cursor passes over each image in the added text, that image is made a part of the edit file. Any image which the cursor passes over will be automatically included in the text unless it is deleted via a DELETE command.

STOP

The STOP command causes the editor to stop reading images from an ADD block, close the ADD file, and return to the original text file. All images which the cursor has passed over and which have not been deleted will be a part of the text file now; any images after the current image at the time of execution of the STOP command will not be included.

CLOSE

This command causes the editor to move to the end of an ADD block and include all remaining images in the file or element as a part of the text file, then close the ADD file and return to the text file.

TYPE sym

The TYPE command is used optionally to declare the language code of the output element. If no TYPE command is entered, the output element will have the same type as the input element, or remain untyped if the input element was untyped. The field "sym" is the mnemonic for the type code, and may take any of the following values:

ASM	Assembler
FOR	Fortran
COB	Cobol
ALG	Algol
ELT	Data element
MAP	Collector symbolics
MAD	Mad symbolic
SEC	Secure symbolic
SSG	Skeleton
MIM	Mimic source program

END

The END command causes the editor to stop dead in its tracks, to abandon editing. All editing which has been performed up to this point is discarded. The END command is an emergency exit for people who have made big mistakes and who want a way out.

FILEDIT -- Conversational File Editor

This program allows the user to examine, and if necessary to modify, the contents of any drum file in any format. It deals strictly with sectors and tracks, and makes no attempt to conform to any particular data format. Observations may be taken in octal or alphabetic format, while corrections may be made in any combination of octal, alphabetic, integer, or floating point. Further documentation is best made by example.

To initiate the program, one need only type

```
@FILEDIT
```

to which the program will respond with the signon line and question

```
GENREL FILEDIT LEVEL x
FILENAME?
```

Answer the 'FILENAME?' question with the complete file name, including, if necessary, qualifier, read key, write key, etc. FILEDIT will assign the file if it is not already assigned, print the assign status if non-zero, and then query

FUNCTION?

This question can be answered in any number of ways. Possible answers, along with their meanings, are listed in tabular form below.

Function	Meaning
GET n	Sector 'n' (octal or decimal integer, octal denoted by a leading zero) is loaded into the sector buffer.
ALPHA m,n	Prints 'n' words beginning with word 'm' in alphabetic format. The words in a sector are numbered 1 to 28. Any attempt to print a word numbered less than one or greater than 28 will result in an error message.
OCTAL m,n	Same as the ALPHA command, except that it prints in an octal format.
CHANGE m,W1,W2,...	Changes the contents of the sector buffer beginning with word 'm'. The words W1, W2, etc. are written over the existing contents of words m,m+1, etc. in the sector buffer. The drum file itself is not changed until the execution of a "WRITE" command. The change words W1, W2, etc. may be any combination of octal (leading zero), floating point (decimal in the number), integer, or alphabetic (delimited by quotes). An alphabetic item which is longer than 6 characters will occupy more than one word.

WRITE	Writes the current contents of the sector buffer back to the drum. This command must be executed after any CHANGE commands if the changed values are to be saved.
NEXT NEXT n	Loads the sector buffer with the contents of the "n"'th sector after the current one. If "n" is left blank, it is assumed to be one.
TOP	Loads the sector buffer with the contents of sector zero. This command is primarily used in conjunction with the 'SEARCH' command described below.
SEARCH target	Searches the drum file for the next occurrence of the item 'target' in or after the current sector. The search target may be any number of words long, and may be any comma-separated mixture of octal, decimal, floating-point, and alphabetic items. See the 'CHANGE' function above for a description of format differentiation of octal from integer, etc. The search is performed up to and including the track specified in the LENGTH directive, or to the 'next write address' if it is a program file.
LENGTH n	Declare the length (in tracks) of the working file, for search purposes only.
TRACK n	Loads the sector buffer with the first sector of the specified track.
(blank line)	A blank line in response to the 'FUNCTION?' question causes FILEDIT to re-ask the question 'FILENAME?', after freeing the old file if necessary.
END	Typing 'END' will terminate the file editor and free the current file, if necessary.

FILES -- List user's assigned files.

The FILES program prints a compact tabular listing of all files and all @USE names currently assigned to the run.

The call is simply

@FILES

The output is in two parts:

1) Table of @USE names.
2) Table of assigned files.

The information printed about each file is as follows:

a) Device code: 'F2', '8CB', 'F4', or whatever. File medium.

b) Permissions: 'R', 'W', 'N', or blank. These stand for read-only, write-only, no permission, and all permissions.

c) Status. 'T' indicates a temporary file.
'A' indicates a catalogued file.
'C' indicates file being conditionally catalogued.
'U' indicates file being unconditionally catalogued.
'D' indicates file being conditionally deleted.
'K' indicates file being unconditionally deleted.

d) File name and cycle.

@FILES,A and @FILES,U will print only the table of assigned files or the table of @USE names respectively.

GARBLE -- Program Security Scrambler

If, for any reason, a programmer would like a level of security for a symbolic element beyond that which can be had through the use of read and write keys on files, then the GARBLE processor can come to his aid. GARBLE scrambles a symbolic element beyond any possible recognition or reconstruction except by ungarbling using the same combination. For example, if it is desired to protect a program named 'MONEY' in a file named 'J', then one would enter

@GARBLE J.MONEY

The GARBLE program will respond with a header line, and then the question

ENTER 3 6-DIGIT COMBINATION NUMBERS.

At this time, the GARBLE program will pause. You must enter three numbers of six digits each for it to use as the combination. After you have typed in the three numbers, it will scramble the element J.MONEY so that it cannot be examined, compiled, printed, punched, or copied.

There is only one way to access a symbolic element once it has been garbled: you must ungarble it using the GARBLE processor and the same three combination numbers. If you attempt to ungarble it using any other combination numbers, you will only compound the garbling. To ungarble, one types

@GARBLE,U J.MONEY

The GARBLE program will again ask for the combination numbers, and if they are correct, it will ungarble the program. If they are not correct, then the program will just be further garbled. If, for some reason, you accidentally type the wrong combination while ungarbling, the element has not been destroyed, for the garbling algorithm is symmetric: typing @GARBLE file.element will reverse the action of an erroneous @GARBLE,U in exactly the same way that normal use of @GARBLE,U will reverse the action of a @GARBLE.

There is absolutely no way that the combination numbers can be analytically reconstructed from a garbled element, even if the actual contents and the garbled contents are available printed side by side. As long as the combination numbers are kept a secret, then a garbled program is completely secure from unauthorized access.

INSERT -- Read a paper tape with minimum overhead.

One of the problems inherent in the design of the 1108 Teletype Handler is that it provided no easy way to read a paper tape with any degree of reliability. The paper-tape reader transmits characters at a fixed rate of 10 characters per second: if the Teletype Handler cannot process them at that rate, then characters are lost and chaos results as a ***WAIT*** message is transmitted to the teletype while the paper-tape reader is churning away.

This ***WAIT*** message is sent whenever the input buffers of the teletype handler are full. The buffers will fill whenever the input rate is higher than the output rate. Thus, optimally, the buffers should be emptied as fast as they are being filled. To guarantee one's ability to empty the buffers at a minimum speed even during periods of system overload, the reading program must tie itself in knots to avoid being swapped out or stalled: If the reading program is ever swapped out, the teletype will keep transmitting, and after 7 seconds the buffers will be full.

The INSERT program uses a sneaky trick involving interrupt I/O and multiple activities to guarantee that it will never be swapped out while a paper tape is being read. For this reason, the INSERT program is guaranteed to be able to keep up with a paper-tape reader regardless of system loading.

To Use:

```
@INSERT  file.element
```

This call will load the INSERT program; it will answer with

```
READY
```

when it is ready to read paper tape. At this point, you must enter an "X-ON" key (control/Q), and then start the paper-tape reader. On some teletypes the X-ON will start the reader automatically.

When the paper tape has been read in, you must type

```
@EOF
```

followed by two successive occurrences of the character "X-OFF" (control/S). The message "END OF TAPE" will appear, and you are done.

ISTTY -- Determine Program Environment.

The FORTRAN function ISTTY permits the FORTRAN programmer to code his programs to be aware of when they are being run from a teletype and to respond accordingly. The function returns a logical value of 'true' when the program is being run from a teletype and 'false' when it is not. To use, the declaration statement

```
LOGICAL ISTTY
```

must be entered at the beginning of the program, along with DIMENSION statements and type declarations. Then, whenever the program wishes to test its environment, it would call the function in an 'IF' statement:

```
IF(ISTTY(0)) GO TO 14
```

This example will cause the program to transfer to statement number 14 if the program is being run from a teletype and to continue to the next statement if it is not being run from a teletype.

The argument of zero must be provided so that the FORTRAN compiler can distinguish between the function 'ISTTY' and a simple program variable.

If the program makes a large number of references to ISTTY, a slight increase in efficiency of execution can be had by using a local logical variable in each 'IF' statement:

```
LOGICAL ISTTY,LTTY
LTTY=ISTTY(0)

. . .

IF(LTTY) GO TO 14
```

MIMIC -- Conversational MIMIC processor enhanced for Teletype usage.

MIMIC is a high-level programming language designed to allow digital processing of analog equations with a minimum of work. The MIMIC language itself is far too complex to describe here, rather refer to one of several publications. The most thorough, if not the best, treatment of MIMIC is contained in the first few chapters of the book Digital Simulation of Continuous Systems (1969, McGraw-Hill) by Yaohan Chu.

The GENREL version of MIMIC differs from the standard universal version of MIMIC in the following ways:

1) It is a processor rather than a program, and expects its input from a symbolic program element instead of from cards.

2) It has been slightly enhanced in its input capabilities to permit conversational entry of parameters during the execution of the program.

3) The plot feature has been extensively re-written to accomodate the teletype as a plotting device, and to include CALCOMP plotting capabilities.

4) A large amount of error-checking has been added to the compiler phase of the program to make it easier for the MIMIC programmer to locate an error in his source program.

5) The Runge-Kutta integrator portion of the program and the mechanism of the LIN function have been completely re-written to correct errors which arose under certain conditions and to improve the accuracy of the integrator. The new integrator is capable of achieving approximately half the error (one more bit of significance) of the previous version, with no appreciable change in speed.

   Previous implementations of the LIN function were apt to overflow when the derivative was very large with respect to the integrator resolution. The new LIN function implementation will not overflow under any circumstances.

MIMIC programs for use with the GENREL conversational MIMIC processor should be entered with the text editor. The TAB features of the editor may be used to make the 10/19 column convention quite painless. In any event, let us assume that there exists a MIMIC program in an element called TEST in a file called FILE. The program may be run by simply typing

```
@MIMIC  FILE.TEST
```

using no options on the processor call. The MIMIC program will be run, and any output will be placed into a file called MIMOUT, which may be examined with the text editor. Plotting will be performed directly on the teletype.

Certain options are available for use with the MIMIC processor call; they are as described below.

Option	Meaning
S	Print the source statements which comprise the MIMIC program.
L	Print the generated MIMIC function-language program.
O	Print all output of the program directly to the teletype instead of to the drum file MIMOUT.
Z	Suppress generation of plots.
C	Print out the values of all constants.
P	Print out the values of all parameters.
V	(for video) plot format is for CRT screen rather than teletype. (specifically designed for UNISCOPE and DATAPOINT 3000).

Two new input functions have been added to provide for interactive input of constants and parameters. They are TCN and TPR (for Teletype Constant and Teletype Parameter). They are used in exactly the same way as the CON and PAR functions, save that instead of reading from the MIMIC source program, they cause the variable name to be printed, followed by a question mark, allowing the user to type in the value of the variable.

The OPT function, designed to specify plot options, is used to declare the length (in rows) of a teletype plot: OPT(N) causes the plot to be scaled to fit into N rows. If not specified, it will be assumed to be 50. The remaining fields of the OPT function behave as described on page 35 of Chu, with the exception that logarithmic scaling has not been implemented. It is in the works, but just hasn't worked yet.

OPT -- Access calling options from XQT card.

One of the primary design principles of a teletype-oriented program is that it be as flexible as possible in its use. The 1108 Executive allows a user to specify twenty-six 'option' letters on the XQT card when he executes his program. The OPT function allows the user to probe these option letters to modify program behavior correspondingly.

To use the OPT function, it must first be delcared 'LOGICAL' in the declaration section of the program:

```
LOGICAL OPT
```

This declaration will allow OPT to return a true/false value for the presence/absence of calling options. In use, one merely references the function OPT wherever he would normally reference a logical variable. For example, one might enter:

```
IF(OPT('W')) PRINT 900,X,Y,Z          testing for XQT,W

IF(OPT('U')) STOP                     testing for XQT,U

IF(OPT('A').AND.OPT('B')) GOTO 10     testing for XQT,AB
```

The OPT function returns a value of TRUE if the option corresponding to the argument has been specified; it returns a value of FALSE if it has not.

READY -- Record Program Execution Progress

The READY program causes the message "READY." to be printed on the teletype. It is intended to be used to signal the end of execution of a program which does not print its own signoff message.

To call the READY program:

```
@READY
```

As an example of its use, consider the following sequence of commands:

```
@COPY,P FILE1,FILE2
@PACK FILE2
@READY
```

When the COPY and the PACK have completed, a process which may take a few minutes, the message 'READY.' will be printed on the teletype.

RINGTEST -- Determine whether or not a tape has a ring.

The RINGTEST program detects the presence or absence of a ring without actually writing on the tape. To use, type

@RINGTEST [filename]

One of the following messages will be printed:

TAPE HAS RING.

NO RING (OR INTERLOCK)

DEVICE IS NOT TAPE.

DEVICE STATUS: XXXXXX

The program cannot differentiate between an interlocked tape (i.e. one which is not mounted or not readied) and a mounted tape without a ring. It is not possible to determine whether a tape is in ready status without actually moving the tape.

An unusual device status returned from the RINGTEST program is usually indicative of the tape being in the middle of a rewind operation.

SENTINEL -- Batch-to-teletype communication system.

When a batch run is submitted from a demand terminal via a START statement, there is no direct way of knowing whether or not said batch run has started, completed, or error-terminated. The SENTINEL processor provides a means by which the teletype user may monitor the progress of batch runs.

The SENTINEL processor maintains twenty-six "sentinels", or message cells, one for each letter of the alphabet. Each sentinel may be set with a date, time, and run id, as well as an optional one-card message. If a batch run sets these sentinels via @SENTINEL cards placed in its runstream, then a teletype run may examine the contents of the sentinels to see whether or not the batch run has passed the specified mark points. A separate set of sentinels is maintained for each project identity, so that a teletype run must have the same project field on the RUN card as the batch run being monitored in order that the same set of sentinels is referenced by both runs.

To set sentinel "Y", for example, one need only include the statement

@SENTINEL,Y

followed by an optional one-line message. The sentinel processor will record the date, time, and run id, and write this information into sentinel "Y". If there is a message card, it will also be written to the sentinel. Later on, a teletype run may probe the sentinel: A teletype run would enter

@SENTINEL

with no option letters. The SENTINEL processor will ask

SENT?

to which question the user would respond with the letter name of the sentinel which he wishes to examine. The two possible responses are:

SENTINEL Y IS NOT SET

or

Y -- SET 4 AUG 70 AT 13:45 BY JONES
(if a message is present, it will be printed here)

All sentinels are kept in a file whose name is SENT$; this file is automatically catalogued by the SENTINEL processor if it is not already present. Because of this feature, it is necessary for the run which tests a sentinel to have the same project field as the run which sets it. To set or test sentinels set by another project, the "@QUAL" card may be used.

TAPE -- Verify and write tape labels.

The TAPE processor allows the teletype user to protect himself against operator errors of mounting the wrong tape. The TAPE processor reads and writes standard-format tape labels, and checks them to make sure that the tape label matches the intended reel number.

TO CHECK THE LABEL ON A TAPE:

@TAPE,options (filename)

This call will cause the TAPE processor to read the label on the tape whose name is (filename) and check that the label matches the reel number. If there is no such tape assigned, the TAPE processor will exit after printing a message to that effect. The various options available are:

Option	Meaning of option
(none)	If the reel number does not check, print a message to the teletype user and ask for instructions.
V	If the reel number does not check, remove the tape and print a message to the operator asking him to mount the correct tape.
X	If the reel number does not check, abort immediately. This option is intended for batch processing only.

TO WRITE A LABEL ON A TAPE:

@TAPE,options (filename),reel

This call will cause the TAPE processor to read the label on the tape, rewind, and then possibly write a new label. The "options" must include the letter "W" (for "write") if any label writing is to be performed. The exact action performed by the TAPE processor depends on the options.

Option	Meaning of option
W only	If there was no label on the tape previously, then write a label with the specified reel number. If there was a label, then print the reel no. from the label and ask the teletype user for instructions.
WV	If there was a label on the tape, ask the operator for hand verification of the reel number. If the reel number given by the operator checks with the intended new reel label, write it. If it does not check, print labels to teletype user and ask for instructions.

WF Write a label on the tape regardless of its original contents. This option should be used with extreme caution if the label protection is to be of any utility whatever.

The format of the tape label is as follows:

```
Word  1:   '1HDR  '  (BCD label sentinel)
      2:   BCD representation of reel number
      3:   Time and Date that label written (TDATE$ format)
      4:
      5:
      6:
      7:
      8:
      9:
     10:
     11:
     12:
     13:
     14:   '1HDR  '  (BCD label sentinel)

(end of file)
```

All labels are written at 200 BPI and odd parity. All labels are followed immediately with an end-of-file. If the reel number entered has the letter 'N' or 'R' (for NORING or RING) as the last character, it will be replaced with a blank. This letter serves only to inform the operator whether or not the tape should be mounted with a ring.

TIME -- Examine clocks of various kinds.

This program analyzes the run's Program Control Table (PCT) to extract the information pertinent to program timing. The three relevant items are

1) Time of day
2) Elapsed processing time (CPU time)
3) Elapsed memory time

Each call to TIME causes it to note and print all of the above quantities. The second and successive calls to TIME will cause it to subtract previous values from current values and print an increment as well as the current value, i.e., the amount of time which has elapsed since the previous call to TIME.

The call to TIME is simply

@TIME

The values printed out are labelled in a short form notation:

Symbol	Meaning
CT	Total elapsed CPU time
DCT	Differential elapsed CPU time
MT	Memory time
DMT	Differential elapsed memory time.
DT/DMT	Ratio of elapsed differential CPU to memory, which is a rough indicator of the billing efficiency of the program.

TTPLOT -- Teletype Plot Routine with Automatic Scaling

The ability to plot data with a high efficiency on the teletype is a very desirable feature of a time-sharing system. Lacking a pen-and-ink electronic plotter, the next best thing is a line plot on the teletype, using the print characters as plot symbols. Subroutine TTPLOT was written to permit ultra-simple high-quality line plots on a teletype.

TTPLOT can plot up to four ordinates against one abcissa. The points must be stored in core arrays, but they need not be in any particular order. The function plotted need not be single-valued: plots of circles and spirals are just as simple as plots of polynomials. To set up data for TTPLOT, one needs an array of X values, and one array for each Y to be plotted against this X. To represent a point (X,Y), one merely lets X(J)=X and Y(J)=Y for some J.

The calling sequence to TTPLOT is as follows:

CALL TTPLOT(X,NPTS,LENGTH,WIDTH,Y_1,Y_2,. . .)

Where

X	is the array of X values
NPTS	is the number of points to be plotted
LENGTH	is the length of the plot in print rows
WIDTH	is the width of the plot in print columns
Y_1	is the first array of Y values (for variable 1)
Y_2	is the second array of Y values
	.
	etc.

Up to four Y arrays may be specified. The plot will be square if LENGTH and WIDTH are in the ratio of 3 to 5. WIDTH may not be more than 50; there are no restrictions except the patience of the user on LENGTH. X and the Y arrays are floating-point, while NPTS, LENGTH, and WIDTH are integers.

As an example, the following program will plot a circle:

```
      COMPLEX THETA,VAL
      DIMENSION X(150),Y(150)
      DO 10 I=1,150
      THETA=CMPLX(0,6.29*I/150)
      VAL=CEXP(THETA)
      X(I)=REAL(VAL)
      Y(I)=AIMAG(VAL)
10    CONTINUE
      CALL TTPLOT(X,150,30,50,Y)
      STOP
      END
```

.O -- Identify Site ID.

One of the simplest programs in a whole passel of simple programs:
ype in @WHO and it will tell you your site-id, channel and line number.
lo options, no variations, no nothin'.