Date: June 4, 1975

From: Susan Barr

Subject: User Interface for FAST

It is planned to add a new subsystem to Multics that provides inexpensive "classical timesharing" capabilities. It will be called Fast Access Subsystem for Timesharing (FAST).

This system must have the following properties; it must:

be inexpensive to use,

include both Fortran and Basic,

be easy to use.

It is desirable for FAST to be a closed subsystem with resource limitations so that lower prices may be charged for it.

It is desirable for it to be similar to the Dartmouth System, DTSS: The motivation for this is twofold:

A primary user of FAST will be General Motors who will transfer a large number of DTSS users to this system.

The DTSS user interface (an extension of the original GE 265 system) is very commonly available on other systems. As a result it is well known and has proven to be easy to use.

Components of this subsystem are:

A new Fortran compiler,

The Basic compiler,

The system modifications that allow a process to use a smaller amount of resources,

Prelinking,

MCS,

A new Command System/Editor similar to DTSS.

Performance is the most important factor in designing FAST. When there was a choice between compatibility with DTSS and performance, FAST did not simulate DTSS. Features should not be added to FAST that will degrade performance, since the user requiring a more powerful system could log in under the full Multics system.

DTSS has a simple editor that is entered at command level. This editor uses two temporary files. The "current" file which contains the file being edited and the "alter" file which contains replacement lines or additional lines to be added to the file when a merge is done. Lines that begin with a number are added to the "alter" file. All other lines are assumed to be commands. This editor works on a complete line each time. It allows the user to add, delete or change a line and to list the file being created.

The "current" file is used for input and output for many commands that use files. If the user calls a command that normally reads the "current" file, the "alter" file and the "current" file will be merged and the result stored in the "current" file before the command is called. No permanent files are created in the user's catalog unless the files are explicitly saved. This approach will be used for FAST.

There are three major areas where DTSS and Multics are not compatible: access mechanism; search procedures; typing conventions and terminal control. The DTSS operating system has several editors that provide similar functions. FAST will include the EDIT requests: merge, sequence, resequence and desequence. The DTSS command DEBUG cannot be implemented with the current Basic compiler so the Multics debugger probe will be provided instead.

Attached is the preliminary specification of the FAST user interface. Later MTB's will discuss resource limitations, the runtime environment for languages in FAST, and the adaptation of FAST Fortran for use in regular Multics. Questions about the specification should be directed as follows:

General Interface:	Susan Barr
Fortran:	David Levin
Basic:	Melanie Weaver

User Interface for FAST - Preliminary

FAST (Fast Access Subsystem for Timesharing) is a new subsystem that will make simple inexpensive time sharing available under Multics. The primary FAST design goal is high performance, especially in regard to keeping a process's working set small. Subject to this goal and the requirements of Multics access control, compatibility with DTSS has been emphasized.

This document contains descriptions of the FAST user interface and of the two programming languages available in FAST: Fortran and Basic. The remainder of the document is organized as follows:

Part I -- General User Interface

System commands (from DTSS) to be included:

File and edit	System	Terminal
APPEND	BILL	DIRECT
BUILD	BRIEF	FULLDUPLEX
IGNORE	BYE	HALFDUPLEX
LIST	CATALOG	KEYBOARD
OLD	GOODBYE	TAPE
RENAME	HELLO	
REPLACE	LENGTH	
SAVE	NBRIEF	
SCRATCH	SYSTEM	
SORT	TTY	
UNSAVE	USERS	
EDIT	EXPLAIN	
PRINT		
NEW		

Compilers

COMPILE (for Fortran and Basic)
RUN

The Multics command probe will be added to replace the DTSS command debug.

Access control:

DTSS has two types of access associated with each segment. Access with a password entered by a user and access without a password. Some of the access is for the convenience of the user and does not protect the segment or its information. For example, LIST access means the segment is ascii and can be listed with the LIST command. This feature does not require a separate access code and a check for non-ascii segments can be built into the LIST command for FAST. The APPEND access is not compatible with Multics.

Multics access control is very different from DTSS access control, All DTSS access except for APPEND is available on Multics. Instead of simulating DTSS access, it is proposed to use Multics access with the SAVE and REPLACE commands. If the user gives no access with the SAVE command the Multics default access will be used (read and write access for the user and the system backup only).

DTSS:	SAVE;RXP,RXP	PUBLIC access
FAST:	SAVE,re *.*.*	PUBLIC access

Search rules:

It is proposed to simulate DTSS and to have no search rules. All external subroutines will be found with the LIBRARY statement. Prelinking will be used for the compilers and the command processor so they will not require dynamic linking.

Typing conventions and input:

DTSS uses different erase and kill characters and has more control in some areas of input than Multics. It is proposed to use Multics conventions. FAST will have the following differences from DTSS:

1. Erase will be "#" instead of "(CTRL) Z".
   Kill will be "@" instead of "(CTRL) X".

2. The internal representation of source segments will use the Multics convention of one character (new-line) for the end of the line. DTSS uses a two character string (carriage return followed by a line-feed) for the end of the line. This change should be invisible to users, since this is implementation dependent knowledge rather than part of the Basic or Fortran languages.

The following DTSS features will not be implemented on Multics.

1. When a line is deleted using the DTSS kill character, the word "DELETED" is printed and the carriage is positioned at the start of the next line. On Multics there is no echo for the kill character.

2. DTSS permits the user to suppress the normal even parity generation. In this mode, the eighth bit is transmitted as generated by the program causing the information to be transmitted to the terminal. This is used for special purpose terminals. There are plans to implement this feature on Multics in the future.

3. The DTSS system allows the user to skip blocks of output (about 256 characters) by typing CTRL X while the terminal is

printing.

Case conventions:

DTSS and Multics differ in the use of case conventions. DTSS stores input from uppercase only terminals as uppercase, but Multics maps input from those terminals into lowercase. The DTSS printer uses uppercase only. FAST should be easy for users to learn. If a user has always seen programs in uppercase, the use of lowercase can be confusing.

1. Case conventions for filenames on DTSS:

   Filenames can be up to 8 characters long and consist of these characters:
   A-Z
   0-9
   hyphen
   period

   The command processor, Basic and Fortran map filenames to uppercase.

2. Case conventions for non-filename use on DTSS:

   a. Command Processor

      The contents of user files are left as the user created them so upper and lowercase distinctions are preserved.

   b. Basic

      1. Upper and lowercase is significant within quoted strings. For example, the subprogram name must exactly match the name given on the CALL statement. In both cases an upper and lowercase distinction is made.

      2. The upper and lowercase distinction is not made outside of quoted strings. (i.e. CALL, CaLl, and call are all treated as a CALL statement)

   c. Fortran

      Fortran maps all characters into uppercase.

   d. Printer

      The DTSS printer can only print with uppercase. That is a deficiency of that system since users can create files and Basic programs where the distinction of upper and

lowercase is significant.

It is proposed to handle upper and lowercase by using Multics conventions, but allow the user some visual aids. FAST will have these features:

1. The input from uppercase only terminals will be mapped into lowercase as is the Multics convention. The users of these terminals will enter uppercase letters by preceding the letter by an escape character.

2. System library names will be lowercase. Segment names will not be mapped into uppercase by the compilers and the command processor. A command will be supplied that will convert letters in ascii segments from uppercase to lowercase. These names will be used as given.

3. A new command, "uppercase" will be supplied for users who would like to see only uppercase characters on a two case terminal. This command could be requested when the user is at command level. It will make a modes call to the tty_ dim. (It has also been suggested that this be a login option.)

   usage: uppercase [edit]

   If the "edit" option is given, all letters will be printed as uppercase. Non-printing characters will be deleted.

   If the "edit" option is not given, the following conventions will be used.

   a. Lower case letters will be mapped into uppercase.

   b. Uppercase letters will be preceded by an escape character.

   c. Non-printing characters will cause the string "\nnn" to be printed to give the octal representation of the character.

4. The FAST implementation of the PRINT command, which lists on a line printer, will have an uppercase option.

Use of external subroutines:

DTSS does not use dynamic linking with search rules. Subroutines that are not found within the source segment are found with the LIBRARY statement. The LIBRARY statement gives the pathname of files to be searched for the subroutines referenced in the current file, but not found there. The LIBRARY statement may

give one or more files to be searched for the subprogram source code. A LIBRARY file may contain more than one subprogram and may contain source or object code.

It is proposed to implement the LIBRARY statement in a similar manner.

Pathname conventions:

DTSS uses different pathname conventions from Multics. It is proposed that Multics pathnames be used. The following list shows DTSS pathnames and the equivalent Multics pathname.

1. <name>
The file is in the user's catalog.

(On Multics: name The file is in the user's working directory)

2. *<user_no.>:<name>
The file is in the main catalog of user with account number user_no.

(On Multics: >udd>project_id>user_id>name The user must know the project name and user name instead of a user no.)

3. <name>***
The file is in DLIBRARY

(On Multics: >ldd>dlibrary>name)

4. :DLIBRARY:<sublibrary>:<name>
<sublibrary>***:<name>
The file is in the sublibrary off the DLIBRARY off the main library DLIBRARY.

(On Multics: >ldd>dlibrary>sublibrary>name)

5. The user's "working" catalog can be changed to be one of his subcatalogs or to a system catalog.

ENTER <subcatalog> Change to sub catalog.

(On Multics: cwd subdir)

ENTER *MYCAT Use main catalog of user.

(On Multics: cwd)

Naming conventions:

On DTSS filenames are arbitrary names given by the user. The user can rename a program after it has been compiled.

It is proposed to use Multics naming conventions Users would follow these conventions:

1. Source code must have a language suffix. (i.e. test.basic, main.fortran).

2. Object segments will be given the source segment name without the language suffix. (When the user successfully compiles "test.basic" the current segment would contain the object code and the "current name" would be changed to "test").

Data chaining:

Chaining will be implemented for both Basic and Fortran.

Background:

The BACKGROUND command in DTSS allows users to run "batch" jobs. This would not be consistent with the idea of the fast limited system because it would allow users to have two jobs being processed.

DTSS users must use BACKGROUND to list segments on the line printer. FAST will supply a replacement for the BACKGROUND PRINT request.

Editors:

The DTSS EDIT operates on the current file and does one request for each call to the editor. It permits the user to merge several files, to move blocks of lines within the file, to resequence the file, and to convert the file to a "string" data file for input to Basic programs.

Edit will be implemented. The other DTSS editors will not be implemented.

Catalog:

The CATALOG command prints information about the user's catalog. On FAST the user will use a subset of the Multics list command arguments with the CATALOG command. No attempt will be made to format the resulting output to look like DTSS.

Part II -- FAST Fortran

This section describes the proposed FAST Fortran by listing the differences between it and Multics Fortran as defined in AJ28, Rev. 0. The differences are classified as follows: Restrictions, Incompatibilities, Extensions, and Undecided Points.

Multics Fortran Restrictions

1. A subscript expression may be integer, real, or double precision. If it is real or double precision, it will be truncated to integer before use. Multics Fortran allows subscript expressions to be complex.

2. An implicit statement may only appear immediately after the subprogram statement, if present, and before all other statements. The ranges of letters specified may not overlap. Multics Fortran allows more than one implicit statement which may appear anywhere in the program body and does not check for range overlap.

3. Block data subprograms may not contain executable statements. Multics Fortran allows this extention because of the implementation of block data subprograms. This extension is at variance with ANSI.

4. Recursive programming will not be supported by the language. There is no automatic storage in FAST Fortran.

5. Extents of parameter arrays may be parameters or constants only. Multics Fortran also allows the extents to be in common or internal static storage.

6. The number of subscripts in equivalence statement must be 1 or N. Multics Fortran allows the number of subscripts to be from 1 to N.

7. As a first release restriction, namelist I/O and the namelist statement will not be supported. They will be supported in a later release.

8. As a first release restriction, alternate return statements and label valued arguments will not be supported. Both Multics Fortran and FORTREV have extensions of this form although they differ on the syntax. The semantics are the same.

9. As a first release restriction, expressions may not appear within an output request implied do-loop.

10. As a first release restriction, entry statements will be prohibited.

11. The maximum character string constant or variable length is 256 characters. Multics Fortran character string constants are limited to 256 characters, although variables are limited to one segment (256K times 4).

12. Restricted assignment statement semantics. If the right hand side is a character string constant, and the mode of the left hand side is integer, the bit-string representation of the right hand side is assigned without conversion to the left hand side. Multics Fortran allows the left hand side to be any arithmetic mode.

13. Character string constants delimited by ' (apostrophe) or " (quote) may not contain the delimit character as part of the constant. Hollerith constants may be used to define any character string constant.

14. The token in a stop or pause statement may only be an integer constant or character string constant. This permits possible future extension to allow a variable name.

15. As an intial release restriction, data statements may not contain implied do-loops.

16. Only one statement is allowed on a line.

17. A do-loop index must be a scalar. Multics Fortran allows both array names and array element names.

Multics Fortran Incompatibilites

1. Do-loop semantics are not compatible. Multics Fortran dynamically calculates m2 and m3 every time the range of the do-loop is executed. The value of i is not compared to m2 until after the range of the do-loop has been executed once. If the program changes the value of i, m2, or m3 this affects the number of iterations performed. The value of m3 must be positive. In DTSS Fortran the expression (m2-m1)/m3+1 is calculated before the range of the do-loop is executed. The values of m1 and m3 are saved at this point. The number of times the range of the do-loop is executed is equal to the value of the above expression. The range of the do-loop is not executed if the value is zero or negative. Modifying the values of i, m2, or m3 has no affect on the number of iterations performed.

2. Multics Fortran implicit line continuation conventions will be replaced by DTSS Fortran free format input. This format is identical to one of the FORTRAN-Y input format options. There will be no card-image format available.

3. There will be no distinction between alphabets. All letters, not in character string constants or hollerith constants, will be folded to a single alphabet. (This feature is available in Multics Fortran with -card.)

4. List-directed input will be continued on the following record only if the last nonblank character of the current record is a comma. List-directed input terminates when the last nonblank character of the current record is not a comma. Any remaining items in the input list are ignored. In Multics Fortran, a comma at the end of the line indicates that a null value follows it. Multics Fortran also reads as many records as are necessary to satisfy the entire list.

5. A prompt character, "?", will be printed when input is requested from the terminal.

6. The first record of a keyed file will have a record number of zero. The first record of a Multics Fortran keyed file had a key of one.

7. The execution of an openfile statement will position a file after the last record.

8. The endfile statement will not rewind a file.

9. At the beginning of a coreload, all variables will be initialized to binary zero by zeroing core. Then any data initializations are performed. Variables will be reinitialized at each invocation unless specified in a save statement.

10. All double precision and complex variables and constants will be aligned on double word storage boundaries. This will affect common block storage allocation, as words may be added to common blocks to force proper alignment; those equivalence groups which attempt to force odd word alignment will be prohibited. The code generated by FAST Fortran will assume double word alignment. Multics Fortran allows parameters, members of common, and equivalenced variables to be unaligned.

11. Variable names are limited to eight characters with no distinction between cases. Multics Fortran allows up to 256 characters in a variable name. Multics Fortran variable names may contain the characters "$" or "_" and the upper and lower cases are distinct.

12. All real constants will be considered single precision. Only real constants with double precision exponents will be considered double precision. Multics Fortran treats long real constants, constants with more than nine digits, as double precision.

13. Endfile semantics. An endfile statement will truncate the file to the current file position. This includes keyed files. The file will not be closed or rewound. Multics Fortran endfile statement only truncates sequential files. All files are rewound and closed.

Multics Fortran Extensions

1. Increase the number of valid mode combinations for exponentiation. The mode of the result will be equal to the highest mode. (See restrictions concerning complex and double precision.)

2. If the first character of a line is "*", that line is a comment line. (This feature is available in Multics Fortran with -card.)

3. The character "**" will be added to the character set. It will be an exponentiation operator. (This feature is available in Multics Fortran with -card.)

4. Equivalence statements will be treated like other declarative statement. Multics Fortran requires that equivalence statements follow all other declarative statements.

5. More than one subprogram may be compiled during one invocation of the compiler. If one of the subprograms is a main subprogram, it must precede the others. Multics Fortran only permits one compilation per invocation.

6. All mode statements (integer, real, double precision, complex, logical, and character) may have an optional length in bytes field. It is ignored for all but real and character statements. For real statements, a value greater statements than 7 indicates double precision, otherwise, it has no effect. For character statements, the value indicates the number of characters per element. (This feature is available in Multics Fortran with -card.)

7. The save statement will be implemented.

8. The parameter statement will be implemented. (This feature is available in Multics Fortran with -card.)

9. Extended assignment statement semantics. Assignment between logical and integer is allowed. Assignment is performed without conversion.

10. File reference 0 is the terminal. Multics Fortran has no file 0.

11. The input statement will be implemented.

12. The margin statement will be implemented. It sets a maximum record length for a unit and therefore may cause records to be transmitted. (The concept of transmitting a record because the next item will not fit is contrary to the Multics Fortran interpretation of ANSI.)

13. Implement a new format specification, s-format. This allows line numbers to be stripped off input file records, and line numbers prefixed to output file records.

14. The openfile statement will be implemented.

15. The closefile statement will be implemented.

16. Formatted random string files will be supported. This will allow formatted keyed files. They will be completely compatible within the Fortran and Basic languages.

17. Random numeric files will be supported. They will be completely compatible within the Fortran and Basic languages.

18. Both keyed and sequential I/O requests will be permitted for a keyed file. This will include a special open mode (to allow open mode "update" even if file does not exist), write requests will perform a rewrite if the designated record exists, and endfile will truncate the file.

19. The library statement will be implemented. Its meaning is the same as in DTSS Fortran except that Multics Fortran pathnames will be used and, possibly-but-not-likely, source libraries will be found at compile time.

20. The chain statement will be implemented.

Undecided Points

1. What Multics flexibilities should be allowed/disallowed in FAST in regards to carriage control, file manipulations, and other I/O fine points? (For example, will read or write statements perform an implied openfile?)

2. Precedence for prefix operators. Multics Fortran, FORTRAN-Y, and ANSI specify that prefix operations have higher precedence than exponentiation. DTSS Fortran and FORTREV specify lower precedence.

3. Precedence of A**B**C. Should exponenentiation be done left to right (DTSS, FORTREV) or right to left (Multics, FORTRAN-Y)?

4. Should implicit conversion between complex and double precision be supported? DTSS Fortran prohibits this although Multics Fortran and FORTREV do not. If it is elected to restrict conversions in this fashion, a subsequent release will probably support it.

5. In a data statement, the following construct will be supported although the semantics are not completely specified yet.
```
data array(1)/"A very long character string"/
```

6. Some library programs (or intrinsic functions) may be added. In particular, the DTSS Fortran functions afile, ifile, and pfile may be added.

Part III -- Comparison with DTSS Fortran

This section describes, to the best of our current knowledge, the differences between the proposed FAST Fortran (first release) and DTSS Fortran. The differences are classified as Restrictions, Incompatibilities, Extensions, and conflicts with the DTSS Fortran manual (TM048). Undecided language points are not covered here. (See preceding section.)

DTSS Fortran Restrictions

1. All arrays will have a lower bound of one. The extension provided by DTSS Fortran to allow the user to supply an arbitrary lower bound will not be supported. (This extension was not available with H440 Fortran.)

2. DTSS direct files will not be supported.

3. The newline character (octal 012) may not appear in a character string constant. This restriction eliminates an ambiguous meaning for the newline character (end-of-line character or a substring of a character string constant).

4. Implied do-loops will not be permitted in data statements. This may not be a restriction as DTSS Fortran is not clear as to whether this feature is supported. This restriction will be removed in a subsequent release.

5. If two or more subprograms are compiled together and one of the subprograms is a main subprogram, it must be the first subprogram in the source segment. This is the same format required by H440 Fortran and TM048.

6. Some of the builtin functions provided by DTSS Fortran will not be supported by FAST Fortran. These include: fsavfl, funsfl, fresfl, fwait, pos, seg, len, cct, lcase, ucase, rnd, rrand.

7. File reference numbers are limited to the range 0 through 99.

8. Assignment statements may only have one left hand side. Assignment statements of the form, a=b=c=expr, are prohibited.

DTSS Fortran Incompatibilities

1. When two character strings are compared, DTSS Fortran always considers the shorter string to be less than the longer. FAST Fortran will pad the shorter string on the right with blank characters and thus compare two strings of the same length. (This is compatible with FORTREV.)

2. The format for an octal constant will be the letter o followed by a string of octal digits. Use of octal constants is limited to data specifications. This is the same treatment used by H440 and Multics Fortrans. DTSS Fortran uses the percent character (%).

3. In the initial release, all parts of a logical expression will be evaluated even if the truth value of the expression has been completely defined. In subsequent releases, efforts will be made to end evaluation as soon as the truth value is defined.

4. The effect of executing an end line will depend on the type of subprogram being executed. In a subroutine or function, an end line will be equivalent to a return statement. In a main program, it will be equivalent to a stop statement. In DTSS Fortran, an end line is always equivalent to a stop statement.

5. The ANSI rules for repeating a format specification will be used. The rules used by DTSS Fortran are not compatible with ANSI.

6. Except when a function name is being passed as an argument, a function reference will always be followed by an argument list, even if the function requires no arguments. Therefore, references to the builtin functions date and time must be of the form date() or time().

DTSS Fortran Extensions

1. The pause statement will be supported.

2. FAST Fortran will remember order, number, and mode for all statement function arguments. At compile time, actual arguments which do not agree in mode with the dummy arguments will be converted to the appropriate mode. DTSS Fortran is oblivious to the modes of the actual and dummy arguments.

3. Some new library subprograms may be included. (List unknown)

4. The pause and stop statements will allow an optional string to be printed. The user may either specify an integer or a character string constant. (This is compatible with FORTREV, Multics Fortran, and FORTRAN-Y).

5. The maximum number of open files is 99, not 16.

6. Mode and dimension statements specifying the same variable name may appear in any order. TM048 requires the mode statement to precede the dimension statement.

DTSS Language Manual Incompatibilites

1. G-format will be implemented as in Multics Fortran, ANSI, and FORTRAN-Y. TM048 describes extensions and capabilities not present in the implementation as well as incompatible with ANSI.

2. The following two statements have the same semantics.

```
data a, b /1.0, 2.0/
real a, b /1.0, 2.0/
```

TM048 claims that the real statement above is syntactically wrong, while the implementation supports it correctly. (Page 43)

3. Complex constants will not be allowed as free format input. TM048 claims they are the only valid input for complex items yet the implementation prohibits them.

4. The letter c as the first nonblank character on a line can indicate a comment line. It may be preceded by at most one blank. It must be followed by at least five blanks. (Because of the canonicallizer, a line consisting of only the letter c will also be considered a comment line.) While TM048 claims this form also, DTSS Fortran does not. (Page 20)

5. The appearance of the character "!" outside of a character string constant indicates that the remainder of the line is commentary and will be ignored. TM048 also mentions this but it is not supported. (Page 20)

6. TM048 talks about the formatted output routines replacing meaningless low order digits with the character "?". It does not discuss what a meaningless digit is. The implementation does not to support this feature.

Part IV -- FAST Basic

Changes to Multics BASIC

The BASIC language provided will be as described in AM82, Rev. 0 with the following changes.

1. The characters "-" and "." will additionally be allowed in subprogram names.

2. The library statement will be supported.

3. The chain statement will be supported.

4. On an input error, the message will make more clear where to retype from.

5. The PER function will check the file's write access for print, write and scratch.

6. The requirement that a $ used in a format statement as a field delimiter must be followed by "+" or "-" will be dropped. In this case, "-" is assumed.

Differences from DTSS BASIC

As far as we can tell from DTSS TM028 (Feb. 1973) and some experimenting on a DTSS system, The resulting FAST BASIC will differ from DTSS as follows.

1. A file number must be <= 16.

2. The ASC function does not handle the actual characters for space, tab, quote or apostrophe.

3. Spaces, tabs and nonprinting ASCII characters are not allowed in unquoted strings in data statements. Uppercase letters in unquoted strings are folded to lowercase.

4. File names may not begin with a colon.

5. Letters outside of quoted strings are folded to lower case rather than uppercase.

6. If a file named in a file statement does not exist and the first operation on it is for output, the file is created.

7. The USR$ function returns the user name rather than an identification number.