Date: April 8, 1975

From: Susan Barr

Subject: Proposed user interface for FAST

It is planned to add a new subsystem to Multics that provides inexpensive "classical timesharing" capabilities. This subsystem has been given several names at various times - Limited Service Subsystem (LSS), Low Level Entry Subsystem, and Fast Access Subsystem for Timesharing (FAST). The currently proposed name is 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 (to be described in another MTB) 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.

The remainder of this document concerns itself with the command system interface and the issues that are encountered in its relation to other components of the system.

Proposed implementation:

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.

FAST will be different from the DTSS operating system in those cases where compatibility with Multics conventions is necessary. 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 command and a version of the Multics editor edm. The DTSS command DEBUG cannot be implemented with the current Basic compiler so the Multics debugger probe will be provided instead.

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		
PRINT		

Compilers

COMPILE (for Fortran and Basic)
RUN

The following Multics commands will be added:

Access	Misc.
set_acl	edm (special version)
list_acl	qedx
set_iacl_seg	probe
list_iacl_seg	change_wdir

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. (See Appendix I for DTSS access codes.) 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 and Multics access control commands (set_acl, list_acl etc.)

Two alternate proposals were suggested:

1. Use Multics access and allow the user to specify additional access with the SAVE and REPLACE commands.

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

   If no access was specified, the user would get the same default as the full Multics today.

2. Use Multics access, but allow the user to specify with DTSS conventions where applicable.

   SAVE;RXP,RXP would map into
   set_acl current_name re *.*.*

<u>Search rules</u>:

DTSS does not use search rules. Every filename reference implies an exact pathname. DTSS does not remember and use initiated segments. There are many ways to implement FAST that are dependent on other choices that are described in sections below, such as the naming conventions, the implementation of the DTSS LIBRARY statement, and the implementation of the call statement in Fortran and Basic. If subroutine calls are to source code, this is not an issue.

It is proposed that calls are to object code. Prelinking will be used to find procedures needed by Basic, Fortran and the command processor. This will be done for efficiency and so that the standard_system library will not have to be in the user's search rules. There will be search rules that include special FAST system libraries, but not initiated segments.

There is an alternate proposal to simulate DTSS and to have no search rules. Prelinking would still be used for the compilers and the command processor.

<u>Typing conventions and input</u>:

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:

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

   b. Command Processor

      The user can set the current name to use lowercase by using OLD, NEW, or RENAME. This name is shown as it was typed by the user, but when it is used in a SAVE or

REPLACE command, the current name is mapped to uppercase and the filename convention is enforced.

c. Basic

Basic maps filenames to uppercase. For example, the LIBRARY statement has quoted strings for arguments. Each string is the name of a file where subroutines used in the program may be found. These filenames will be mapped into uppercase, but the actual subroutine names may use upper and lowercase. There may be several subroutines in one file.

d. Fortran

Fortran maps all characters not in quoted strings to uppercase, as a result filenames are in 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:

a. 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.

b. System library names will be lowercase. (An alternative would be for these segments to have both upper and lower case names; I think this is unnecessary.) Segment names will not be mapped into uppercase by the compilers and the command processor. These names will be used as given.

c. 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 would 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.

1. Lower case letters will be mapped into uppercase.

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

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

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

There is an alternate proposal to simulate the DTSS conventions.

a. The command processor would enforce the uppercase segment name convention with the SAVE and REPLACE commands.

b. Basic and Fortran would recognize segment name references and map segment names to uppercase.

c. Uppercase only terminals would not have input mapped to lowercase. By system 3.1, MCS would allow the FAST process overseer to change the conversion tables so that uppercase would be mapped to lowercase on uppercase only terminals.

Comparison:

The use of Multics conventions with the extra uppercase option seems to solve the problem and it remains compatible with the full Multics system. The user's concern is only with how the contents of segments appear, not their internal representation.

This proposal has the advantage that a user can create a program on an uppercase only terminal and be able to edit the segment on a two case terminal without having to use the shift key to keep the contents of the segment consistent.

Use of external subroutines:

1. Basic

   DTSS does not use dynamic linking. The COMPILE command tries to compile the current file. The source code for external subprograms is found at compile time using the LIBRARY statement. The LIBRARY statement gives the names of files to be searched for the source code of subprograms referenced in the current file but not found there. These subprograms are compiled as if they had been part of the current file.

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

2. Fortran works differently. The current file can be compiled separately without the subroutines it references. When the program is executed and the subroutine is called, it is then compiled and executed.

It is proposed to use Multics conventions. Users would have to learn these incompatible Multics conventions:

a. Source segments need a language suffix.

b. Object segments can not be renamed arbitrarily. On DTSS the user can give an object segment any name since there is only one entry name for an object program. On Multics, the object segment can have several entry points, so the name of the source segment without the language suffix is used as the principle entry name. Then a call to "name" is assumed to be a call to the principle entry point, "name$name".

c. Subroutines can be linked to at runtime.

There are two additional proposals:

1. Simulate DTSS:

   a. Basic

      The Basic compiler would use the LIBRARY statement to search for the source code for the subprograms not found in the current file. There would be no dynamic linking to object subprograms.

   b. Fortran

      The Fortran compiler would use the LIBRARY satement to search for the source code or object code for the subprograms not found in the current file. There would be no dynamic linking.

2. Use a combination of the DTSS LIBRARY statement and Multics dynamic linking.

   The previous proposal would be implemented with an additional search. Subroutines not found in the source segment and not found with the LIBRARY statement would be assumed to be object segments to be found at runtime.

Comparison of the proposals:

The DTSS simulation has some advantages in terms of storage. The user does not need to keep the object code of each subprogram in a separate segment which must use a minimum of one record. The user does not even need to keep object code for subprograms. The source code for several subprograms may be stored in one segment. This could be a large saving if the user has many small subprograms.

This method has the disadvantage of increased compile time. Every time a change is made in the main program or any subprogram all the source code must be recompiled. If the user stores several subprograms in one segment then editors and the prepass for Basic will be more costly. DTSS simulation means Basic programs will not be able to call Fortran subroutines, since calls are to source code at compile time.

The advantage of following Multics conventions are: FAST and full Multics will be compatible; no prepass is required; the user can use several small object segments instead of one large one; subprograms don't have to be compiled every time some other part of the program is changed.

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: [wd]>name)

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

   (On Multics: >udd>projectid>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  )
```

There were these alternate proposals to allow the users to continue using DTSS pathnames:

1. Have FAST do the conversion to Multics pathnames at runtime. This could be done with a spcial version of the Multics subroutine expand_path.

2. Have the FAST compilers do the conversion to Multics pathnames at compile time. If this is done, then object code produced under FAST and the full Multics system would be the same with respect to pathnames.

3. Supply a FAST command that reads a source file and converts DTSS pathnames to Multics pathnames.

Naming conventions:

On DTSS filenames are arbitrary names given by the user. The user can rename a program after it has been compiled. The naming convention used by FAST is dependent on the previous choice of implementing external subroutines.

If Multics dynamic linking is used, then it makes sense to use Multics naming conventions. Users would have to learn these Multics conventions:

a. Source code must have a language suffix.

b. Object segments can not be renamed.

An alternative proposal could be implemented if external subroutines are found at compile time. All programs compiled under FAST would have the same entry name. The source code for the COMPILE command would be a temporary segment (called the current file on DTSS) in the process directory. This segment would have the name "main.basic" or "main.fortran" so that the compilers would use "main" for the entry name. The compilers would store the object code in a segment in the process directory. When the user copies this segment into his working directory using the SAVE command, he can give it an arbitrary name. When the user calls the EXECUTE command with an object segment "name", the command would call "name$main".

Data chaining:

Chaining is used on DTSS for two reasons: to provide an overlay mechanism in order that a program that exceeds the core maximum can be divided; and to allow users to pass files from one program to another.

The DTSS command EXECUTE uses chaining to pass the current file and a scratch file to programs (as files #1 and #2) There is a method to signal the command processor to exchange these two files when a program returns to command level. This allows users to edit the current file without having to reference it by name. Information about the file position is also passed.

Chaining will be implemented for both Basic and Fortran.

Background:

The BACKGROUND command in DTSS allows user's 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 would 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.

The DTSS TEXT and STRING editors operate on the "current" file but continue to read edit requests until an "exit" request. TEXT recognizes both Basic line numbers and strings in its edit requests. STRING uses only character strings. The function of these editors could be replaced by a version of the Multics editor edm. A new entry could be added to edm, which would pass a pointer to the "current" file. All edm requests would be permitted. The write request would not allow a name argument and would be interpreted as a write to the "current" file. This change to the installed edm is necessary to prevent the user from changing a file in his catalog from within an editor.

QED is very similar to the Multics editor qedx. There are different conventions for the escape characters and differences in some forms of addressing. The Multics editor qedx will be substituted.

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.

Appendix I

DTSS access conventions:

A	append	length may be extended
C	compile	is compiled object code
F	fetch	may be used in program from different user number
G	group	available to users in same group
L	list	may be listed
P	public	may be copied by any user
R	read	may be read from
W	write	previous contents may be replaced
X	execute	is executable machine code

<u>Command</u>	Access needed on "current" file
BACK	RA
COMPILE	R
LIST	RL
OLD	R or X, also G or P or Fif file is in another catalog
PUNCH	RL
RENAME	R
REPLACE	RWA on file being replaced
RUN	R or X
SAVE	R
SCRATCH	RWA
UNSAVE	RWA on file being unsaved

<u>Statement</u>	Access needed on file referenced
FILE	R or W or A (F also needed if saved in another user´s catalog)
INPUT	R
PRINT	A
READ	R
SCRATCH	WA
WRITE	A if file is lengthened W if some elements in file are destroyed