[illegible]

MULTICS TECHNICAL BULLETIN - 393

To: Distribution

From: Bob May

Date: September 20, 1978

Subject: GTSS MTB 393

This MTB contains the proposed definition for the GCOS TSS Environment Simulator. It also contains preliminary documentation for certain tools that are used in the construction of the simulator.

The GCOS TSS simulator provides a user interface that is nearly identical in most respects to that of native GCOS TSS. Its purpose is to provide GCOS users with a means to execute native GCOS software without change. GTSS is not intended to ever be totally identical to native GCOS TSS in either function or performance. It is intended to provide a reasonably complete subset of functions with reasonable performance.

This document is intended to be something more than an MTB. It is expected that it will be the basis of MPM, PLM, and Marketing literature. By providing all relevant information in one place, readers can easily use what they need and ignore what they don't need.

All comments should be addressed to:

Bob May
May.Multics on System M in Phoenix

or

HVN 341-7295/7466
(602) 249-7295/7466

INTRODUCTION

This MTB is intended to fully describe the user interfaces for the GCOS TSS Environment Simulator, hereafter called GTSS. In addition, several of the major subroutine descriptions are included as a means of documenting the internal design of the simulator.

Additionally, the preliminary descriptions for gcos_debug and gcos_library_mgr are included. These tools are being used in the construction of GTSS and will be submitted for installation after cleanup.

NAMING_CONVENTIONS

The GTSS facility consists of a command interface and a large number of supporting subroutines. The command interface will be the gcos_tss (gtss) command. All subroutines will have a prefix of gtss_.

The formal name of the facility will be the GCOS TSS Environment Simulator. The term encapsulation is inappropriate and will not be used.

MARKETING_REQUIREMENTS_FOR_GTSS

The following items are given in an effort to define the "marketing requirements". They are in lieu of a formal document from Marketing.

o No formalized, "Complete" List

  No list of functions required was generated, other than "provide what is provided by native GCOS TSS". It is not sufficient to list all documents related to GCOS TSS as the definition of GTSS because not all functions are documented in manuals. See Appendix A for some unofficial requirements as defined by members of the Bell Canada programming staff.

o As close to "Exactly the Same" as possible

  within the constraints of time and resources, GTSS interfaces will be as close to native GCOS TSS interfaces as possible, with a few exceptions. The exceptions will be in those areas where it is felt that Multics functions are sufficiently desirable that an incompatibility is justified.

o 4/J System Release Level

The native GCOS object code and the executive interfaces provided will be as of GCOS Release 4/J-S (Supplement).

o Identical execution of slave object code, including slave system software

Within the limits of processor compatibility, native GCOS slave object code will be executed "as-is" on the Multics processor.

o Identical content of user files

The user files will be identical in format to those on native GCOS. All considerations due to the Multics virtual memory and segment sizes will be transparent to the GTSS user.

o Identical user terminal interfaces

Multics terminal interfaces are used in GTSS rather than those of native GCOS. It is felt that the benefits of using the Multics interfaces, such as full-duplex, type-ahead, canonicalization, etc., sufficiently compensate for the problems of incompatibility. See below for detailed descriptions of these differences. The reader should note that these differences are with respect only to the terminal I/O, and have no relation to the command syntax.

Paper tape I/O will be provided but not in the initial release. It requires further study to determine what is needed.

o Identical Performance

It is not a goal of this development to provide a facility that is equal in performance to that of native GCOS TSS. A reasonable effort will be made to be as efficient as possible.

o Support command, user libraries

User command file processing will be supported after the initial release (MR6.9).

User libraries will be supported in the initial release.

o Utilities for file/hierarchy transfer between GCOS and Multics

The gcos_fms command provides a GCOS USER RESTORE facility for the bulk transfer of files from GCOS to Multics. It is the subject of another MTB.

GTSS_DESIGN_PHILOSOPHY

The following items define the overall design considerations for GTSS. See Appendix B for a discussion of alternative designs considered.

o Aim toward "Identical Black Box"

This is the primary requirement. Any site that wants GTSS is likely the user of native GCOS TSS. It is incumbent on GTSS to minimize the interface differences between the two facilities in order to minimize the user training and data conversion requirements.

o Direct execution of GCOS object code

Because the Multics CPU is a pure superset of the GCOS CPU, it is possible to run native GCOS by switch the processor to GCOS mode. While operating in Multics mode, there are a few functional differences in instruction functions, ie., use of the BAR mode, and Address Register operation versus Pointer Register operation. These differences can be hidden from the GCOS object programs so that slave GCOS object programs need no alteration. This greatly simplifies, or, more correctly, makes possible, the development of the GTSS simulator package.

o Avoid GCOS hardcore, privileged code

The use of hardcore and/or privileged code from native GCOS was considered and dropped; the task of providing an environment that would allow these types of modules to run as-is, or even with modification, is too great at this time.

o Use primitives of Multics Operating System for management of user process

Much of the GCOS TSS executive must concern itself with the management of multiple time-sharing users. The GTSS facility accommodates one user per process and leaves all process management to the Multics operating system. This greatly simplifies the function of GTSS.

This implies a user interface difference, however. Since the GTSS implementation does not provide any accounting function of its own, there will be no accounting function similar to native GCOS TSS. Thus, billing, which on GCOS includes terminal I/O, will change for the users.

o No Conversion of User Files, Including Programs, Data, Command Files, etc.

  This applies to all user files, whether they are programs or data. Extreme caution has been taken in the design of GTSS to ensure proper operation with existing files that may be brought over from native GCOS. This does not apply for file data that is dependent on processor timings, accounting information and similar information that is machine-dependent.

o GTSS Simulator to interface at the derail level

  The derail instruction and subsequent fault provide a clean separation between the GCOS slave object programs and the GTSS executive. Machine conditions are saved and restored, generally with minor changes, when GTSS returns control to the user.

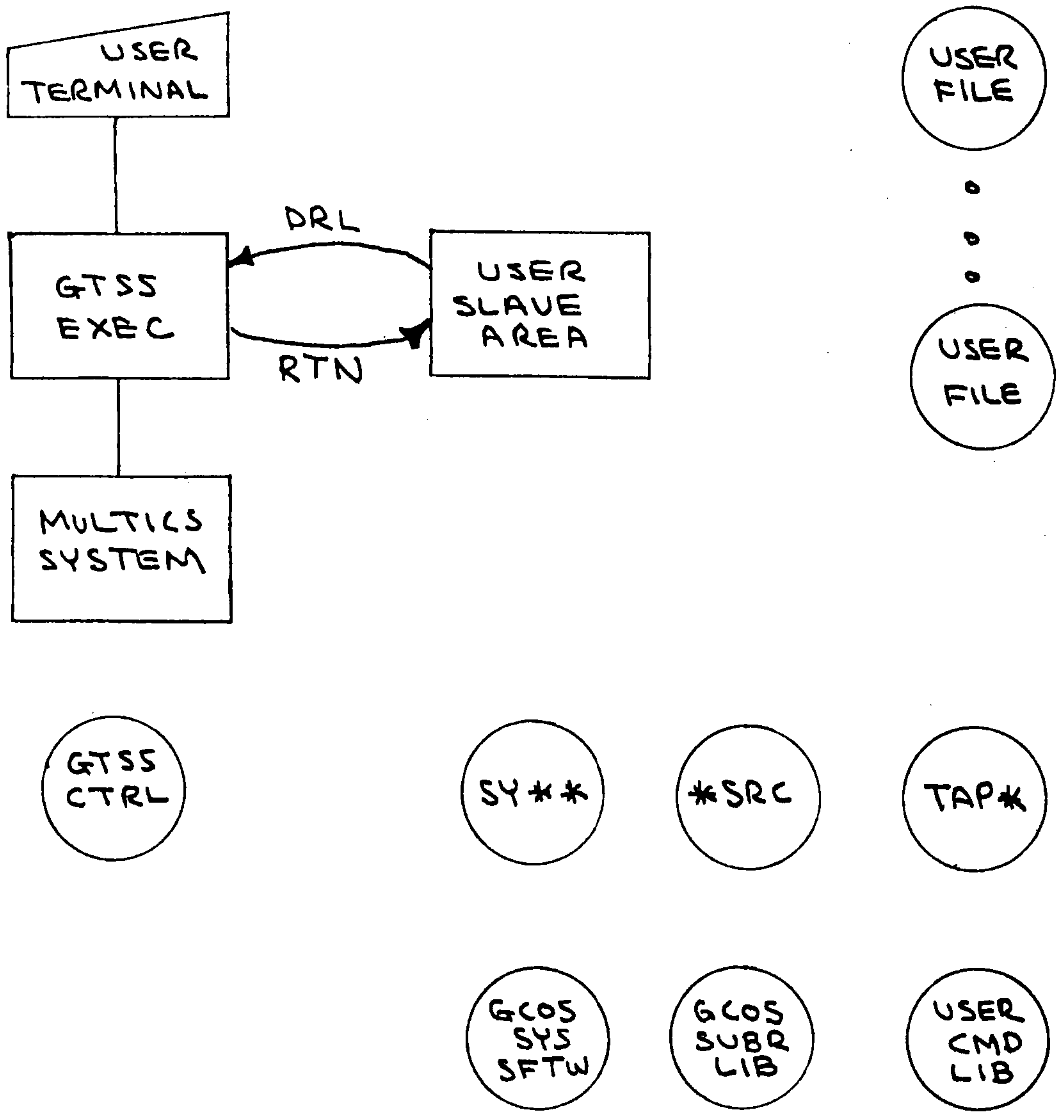
o Write GTSS in Multics PL/I

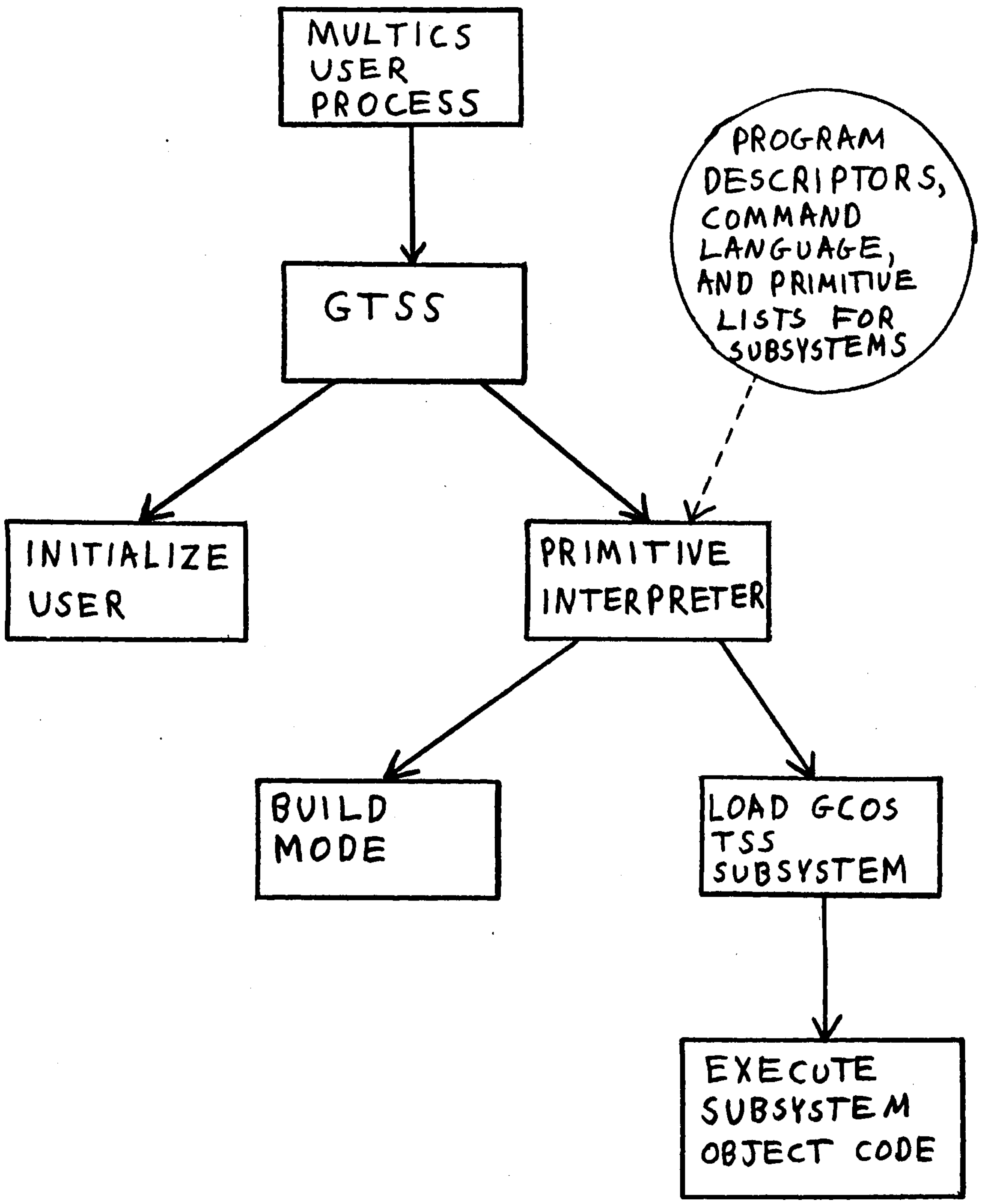
  Since GTSS is only a "black box" interface to the GCOS object programs, there is no requirement to write any GCOS code as part of the executive interface.

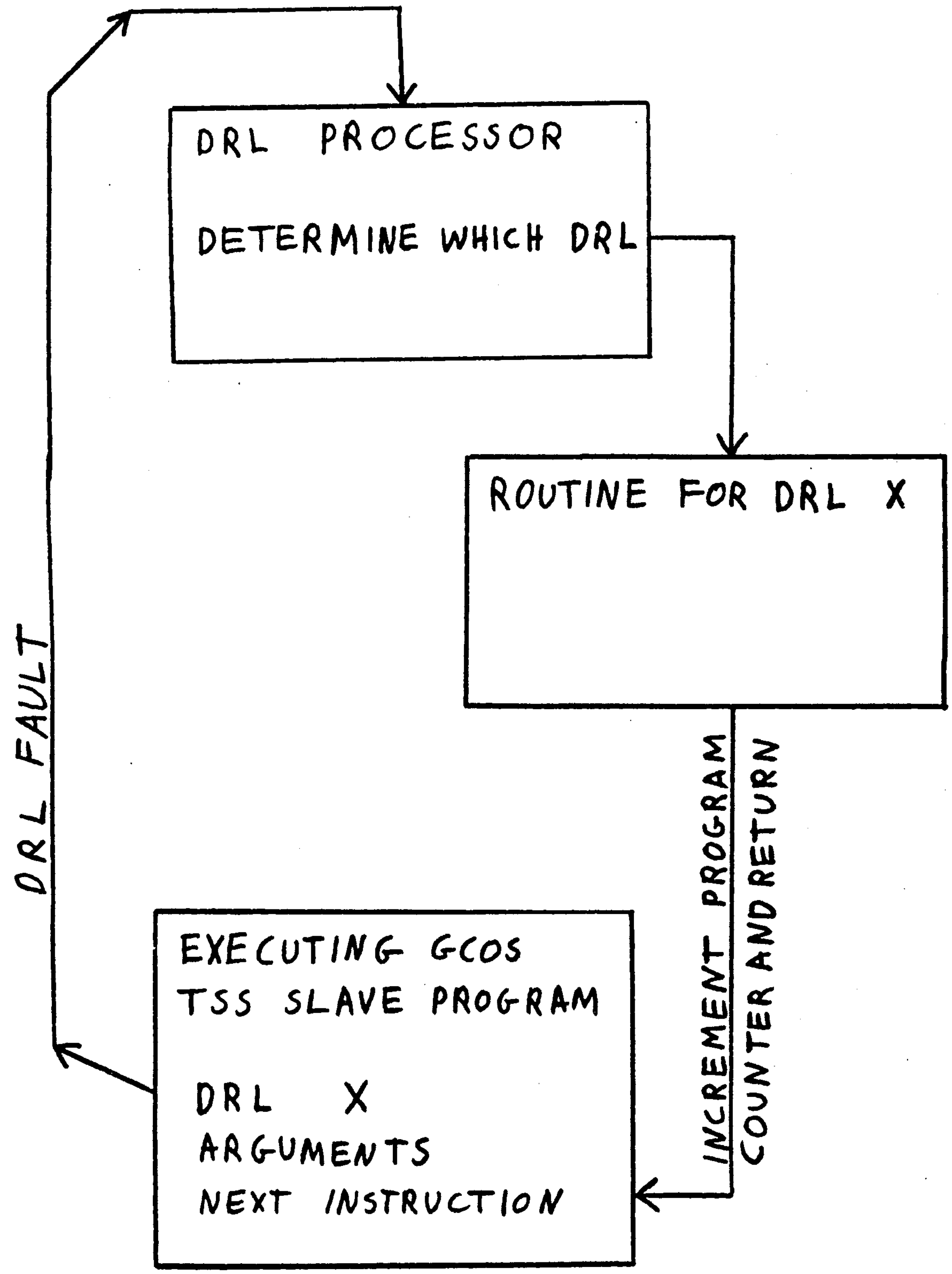
GTSS_ARCHITECTURE

The following figures give the overall structure of GTSS, and indicate the flow of control between the GCOS object code and GTSS.

GTSS ARCHITECTURE




MULTICS USER PROCESS
GTSS
PROGRAM DESCRIPTORS, COMMAND LANGUAGE, AND PRIMITIVE LISTS FOR SUBSYSTEMS
INITIALIZE USER
PRIMITIVE INTERPRETER
BUILD MODE
LOAD GCOS TSS SUBSYSTEM
EXECUTE SUBSYSTEM OBJECT CODE


DRL PROCESSOR
DETERMINE WHICH DRL
ROUTINE FOR DRL X
INCREMENT PROGRAM COUNTER AND RETURN
DRL FAULT
EXECUTING GCOS TSS SLAVE PROGRAM
DRL X
ARGUMENTS
NEXT INSTRUCTION

GTSS IMPLEMENTATION

The approach to GCOS TSS simulation involves an interactive user interface which looks like GCOS timesharing and a simulated environment which allows a user to execute GCOS TSS subsystems. This approach is broken down into the following list of functions:

1. Recognition of GCOS command language for each subsystem using tables from TSSA. Each command recognized has a corresponding list of primitives to be interpreted.

2. Interpretation of TSS primitives. There are a total of 12 primitives to be interpreted. They provide for stacking the current subsystem and calling a new one, initiating the loading and execution of the current subsystem program, initiating the building of input, returning to the subsystem at the previous level, manipulating bits in the subsystem switch word, and conditionally executing blocks of primitives under control of specified bits in the subsystem switch word.

3. Providing the command loader function which allows a user program stored on an H* file to be loaded and executed. The command loader is invoked whenever an unrecognized command is given.

4. Providing a basic line editor which takes input from the terminal and merges it in line numbered sequence with the current file.

5. Providing a static DRL handler similar to the MME handler in the GCOS batch simulator. This handler uses a transfer vector to cause the appropriate routine to be executed for each DRL.

   The derail processing functions are implemented as separate routines so that development of GTSS can be cleanly divided among multiple developers.

GTSS_IOS_Overview

The procedure gtss_ios_ supports disk I/O for the batch and timesharing environment simulators. This procedure simulates GCOS physical I/O and the "ios" in its name refers to the GCOS Input Output Supervisor.

The main entry point to this routine is gtss_ios_$io. This entry point interprets GCOS I/O select sequences to perform the disk I/O for MME GEINOS for batch simulation and DRL DIO for timesharing simulation. Refer to DD 19 (General Comprehensive Operating Supervisor) for an explanation of MME GEINOS, DD 17 (TSS System Programmer's Reference Manual) for an explanation of DRL DIO and DB 82 (GCOS I/O Programming) for a general discussion of GCOS I/O. The gtss_ios_$io entry point supports file input and output and (for batch) file spacing for linked files.

There are five additional entry points. Accessing and deaccessing files are accomplished by gtss_ios_$open and gtss_ios_$close. File size may be changed by gtss_ios_$change_size. The last two entry points are specifically for timesharing. The DRL SWITCH which exchanges two temporary file names is supported by gtss_ios_$exchange_names. The DRL FILSP which does file spacing on linked files is supported by gtss_ios_$position. More information about these six entry points is given in the subroutine description of gtss_ios_ which follows later in this document.

The gtss_ios_ module provides the ability to open and close files and supports all functions that are performed on open files. This is the rationale behind grouping these functions into one module.

The data structures necessary for file I/O may be classified as those common to batch and timesharing and those specific to one of the two simulators. The PL/I include files gtss_file_attributes and gtss_disk_file_data contain the data structures which will be common to both simulators (once the batch simulator has been updated to use gtss_ios_). The routine gtss_ios_ uses only these data structures. The only information specific to each simulator is the symbolic name by which each file is referenced. Time sharing uses an 8 character ASCII name stored in the AFT (Available File Table.) Batch uses a two character BCD file code.

For timesharing this information is maintained as a hashed list of names in a structure described by gtss_aft_.incl.pl1. This structure called gtss_ext_$aft is maintained by the gtss_aft_ routine which has entry points for adding, finding, and deleting names. The corresponding batch structure for storing BCD file codes has not yet been defined.

The data structures common to timesharing and batch may, for permanent files, be further broken down into information which is needed only while the file is open and information which must be stored permanently in the file system. Those needs are covered by gtss_file_attributes.incl.pl1 and gtss_disk_file_data.incl.pl1, respectively.

We have adopted the goal of storing no Multics control data in the user's physical file space. This means that there must be a place provided for storing permanent file attributes in the Multics storage system separate from the user's file. It would be possible to store the attributes for a group of files in a single segment, but we have initially taken the simpler approach of using added names to hold the required attribute information. GCOS file names are restricted to no more than twelve characters so it will always be possible to add these names. See the description of gtss_attributes_mgr_ for details of the naming syntax.

The attributes data has a structure defined by gtss_file_attributes.incl.pl1. The attributes structure currently includes the current size of the file, the maximum size that the file can grow to, one word of user attributes, and a one word file description as supplied by timesharing's DRL PASDES. Potentially this data will also include control information for regulating concurrent file accesses and other attributes as needed (by IDS for example).

The structure gtss_ext_$disk_file_data contains information which must be maintained about each open disk file. This is an arrayed structure which has entries for 41 files. This corresponds to a maximum of 40 open files for batch plus one entry for temporary space used by gtss_ios_$exchange_names. Under time sharing a maximum of 20 files can be open.

Under timesharing the entry for a particular file is located in gtss_ext_$disk_file_data by using the index corresponding to the index of the file name in gtss_ext_$aft. This index is returned by any of the entry points gtss_aft_$add, gtss_aft_$find, and gtss_aft_$delete. Figure 1 shows an example of a timesharing file called MYFILE with the file information being located by using the index corresponding to the position of the file name in the hash list.

The information maintained in gtss_ext_$disk_file_data includes a copy of a GCOS PAT (Peripheral Attach Table) body. This includes a flag which indicates whether the file was opened in random or linked mode. (A linked file can be opened in random mode but not the other way around). For linked files the current position within the file is maintained.

Other information about the file includes whether it is a multisegment file (MSF) and the permissions requested when file is opened. Under this implementation the various possible GCOS permissions are collapsed to read and read/write.

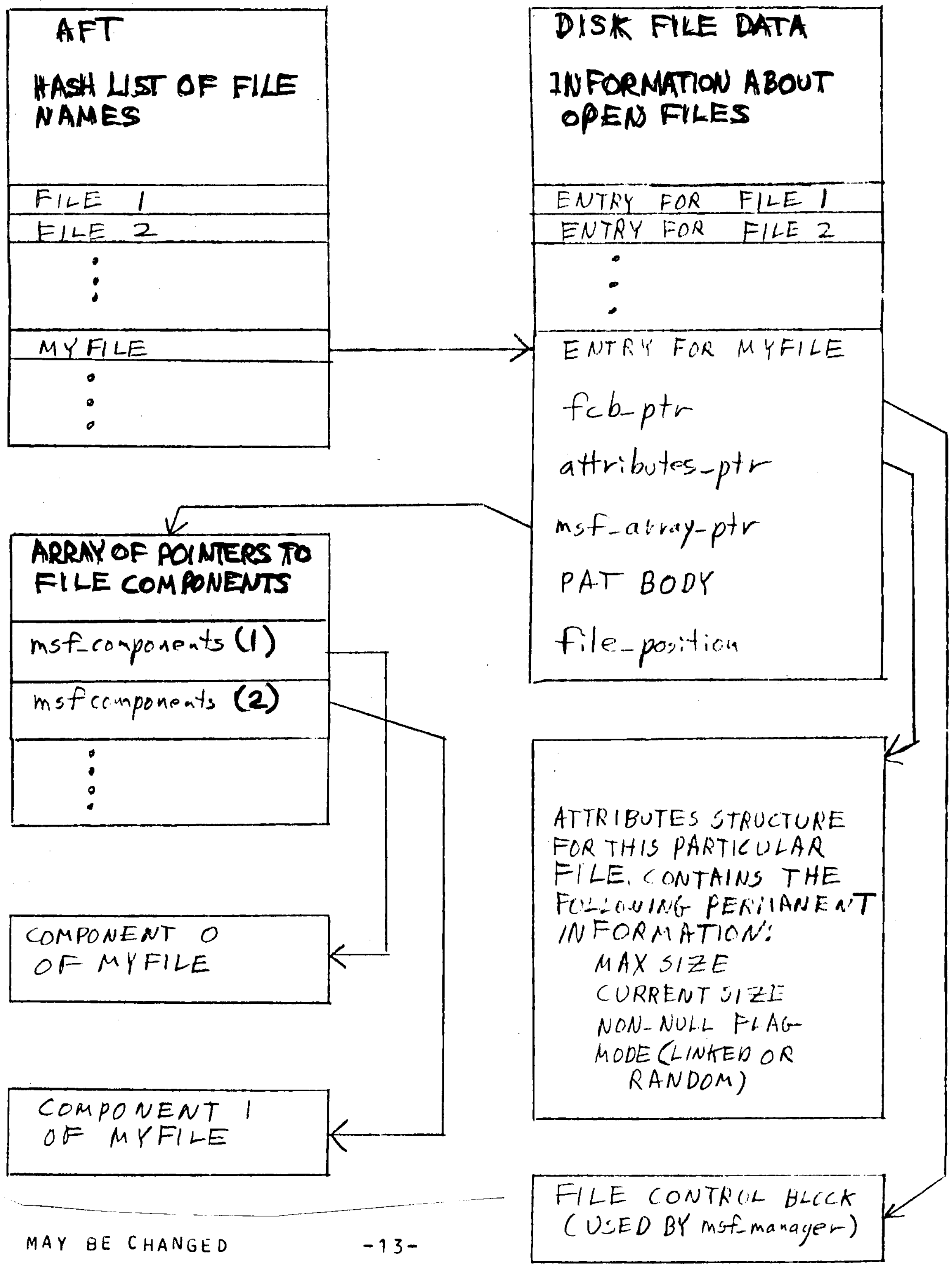
If the file is a MSF there is a pointer, msf_array_pointer which points to an array of pointers, msf_components, to each component of the MSF. This possibility is illustrated in Figure 1. The array msf_components is allocated only for multisegment files. The current implementation provides for a maximum of 500 components in an MSF.

There is also a pointer, attributes_ptr which points to the previously discussed attributes data for a permanent file. For a temporary file the same attributes structure must be allocated in a work area and initialized prior to calling gtss_ios_$open.

Finally there is a pointer to an msf_manager file control block. The msf_manager is used to obtain pointers to file components and grow and shrink files as appropriate.

FIG. 1

DATA STRUCTURES FOR A MULTISEGMENT TIMESHARING FILE



TERMINAL INTERFACE

GTSS will use existing Multics terminal interfaces wherever possible. It is felt that the GCOS TSS interface is too restrictive (no echoplex, full duplex, canonicalization, type-ahead, etc.), and that the users will be willing and able to learn the Multics interfaces in order to get the Multics functions.

See below for a discussion of terminal interfaces as they relate to the GCOS user community.

Some extensions to Multics will be required to accommodate certain GCOS TSS requirements. GCOS TSS uses trailing white space (blanks) on input to input a blank line. (There is no null line concept as in Multics; a null line in GCOS is a line with some number of blanks.)

Additionally, GCOS TSS accommodates paper tape I/O. PPT I/O is a requirement for Bell Canada. The mechanism for paper tape is still under study.

The following paragraphs describe the steps to be taken by a user who wants to use GTSS:

o User first logs on to Multics

  The user will log into Multics as a normal Multics user. As such, the user will be subject to normal Multics Answering Service controls as applied by the system and project administrators.

o GTSS is called

  To enter the GTSS facility, the user types the gcos_tss (gtss) command. Options are provided to control certain Multics-related functions.

  When the user types BYE under gcos_tss, the user will be returned to command level. For those individual users who do not plan to use the Multics command functions, a simple abbrev/exec_com can be set up to automatically log out the user after a BYE.

o User Id and password input

  There will be no additional password required of the user. Under certain modes of operation, the user must give the GCOS TSS-like USERID. This value is used in the mapping of GCOS pathnames into Multics pathnames.

o Break key functions normally (as on native GCOS)

The use of the break key will be processed as on native GCOS TSS, where it causes the currently executing subsystem/command to be reset. An option is provided for the gcos_tss to override this and cause the user's process to go to Multics command level. See the description of the gcos_tss command for additional details.

o System responds with line feed after each carriage return

On native GCOS, the user type a CR to indicate the end of the input line. GCOS TSS generally prints a New Line and asterisk to indicate that it is ready to accept the next input line.

o GCOS erase and kill characters provided (Multics set_tty command option)

The GCOS TSS erase and kill characters will not be set by the gcos_tss command. It is felt that the standard Multics erase and kill characters should be used to facilitate growth into native Multics. For those users who must have this, the set_tty command may be used to set these values. There are no plans to have the system respond with "DEL" upon receipt of a line delete.

o Terminal input/output like GCOS with few exceptions

If GTSS can provide additional detail for the user when reporting errors, it will.

TERMINAL_INTERFACE_DIFFERENCES

This section describes the user terminal interface differences between the Multics GTSS and the native GCOS Time Sharing System.

USER_SIGN-ON_PROCEDURE

The sign-on procedure and system greeting message pertaining to Multics log-in is described in the Multics Introductory User Guide (AL40).

The GCOS Time Sharing System log-on procedure is described in the TSS General Information Manual (DD22).

INPUT_LINE_TRANSMISSION_CONVENTION

The GCOS Time Sharing System convention to indicate the completion of the typed input line transmission is by a carriage return, an ASCII RETURN character (octal code 015).

The Multics GTSS convention to indicate the completion of the typed input line is either the ASCII LINE FEED character (octal code 012), or the Carriage Return (octal 015). The default is the New Line, but can be changed with the set_tty command to be the Carriage Return (set_tty -modes lfecho).

EDITING_CONVENTION

Two editing editing capabilities on the typed line are available. They are:

1. the ability to delete the latest character or characters.
2. the ability to delete the entire line.

Characters or line deletions are effected by means of two special characters designated as control characters. These two characters may differ between terminals.

GCOS Time Sharing System editing control characters are

For_teleprinter_terminals.

character	control_function
@ (commercial at sign)	character deletion
CTRL plus X keys	line deletion

For_IBM_2741_or_DATEL_terminals

character	control_function

1/4 (or degree symbol)	character deletion
±	line deletion

NOTE: Line deletion does not occur until a carriage return is given or ATTN (IBM 2741) or INT (DATEL) is pressed.

The editing rules are as follows:

1. Use of the character-delete control deletes from the line the character preceding the deletion character; use of n consecutive deletion characters deletes n preceding characters (including blanks) up to the beginning of the line. Although the character delete character is a printable symbol it does not become part of the line.

   For example:

   *ABCDF@E would result in ABCDE being transmitted.

2. Use of the line-delete control deletes the entire line. The characters DEL are printed to indicate deletion.

   For example:

*ABCDEF CTL/X DEL	(all characters deleted; a carriage return is automatically supplied)
	- ready for new input.

The Multics GTSS character and line deletion control character conforms to the Multics editing convention. The two editing control characters for teleprinter, IBM 2741 and DATEL terminals are:

character	control function
# (number sign)	character deletion
@ (commercial at sign)	line deletion

The editing rules are as follows:

1. The character delete control symbol deletes from the line the character typed preceding the deletion character. Several successive number signs deletes an equal number of typed characters preceding the number sign. When the character-delete control is the only symbol in a print position, it erases itself and the contents of the

previous print position. Although the delete character is a printable symbol it does not become part of the line.

One character-delete symbol typed immediately after "white space" causes the entire white space to be erased. (White space is defined as: any combination of spaces and horizontal tabs)

The benefits of the white space concept are:

a. Reduces the number of keystrokes necessary to remove any white space
b. Eliminate the need for a user in remembering how many spaces or horizontal tabs have been typed on a line

   For example:

   TheSSne###next

      or

   TheST#next

   where S is a space and T is a horizontal tab produces:

   Thenext

2. The line delete-control symbol deletes the contents of that line up to and including the line delete control character.

   For example:

   This is atektd@What is this

   produces:

   What is this

The Multics user terminal interface provides the user with the ability to define the characteristics and modes of a specific terminal associated with terminal input/output by using the set_tty command. With this command the user can set various modes to effect certain terminal action such as specify the character-delete and line-delete symbols, "echo" a carriage return and or line feed etc. For detail description of the set_tty command, refer to the Multics Command and Active Function Manual (AG92).

ESCAPE CHARACTER CONVENTION

The Multics GTSS conforms to the established character escape convention of Multics, represented by the left slant (\). Escape conventions are provided for terminals that do not have full ASCII character set and are described in the Multics Programmers' Manual Reference Guide (AG91).

The universal escape conventions are:

1. The string \d1d2d3 represents the octal d1 d2 d3 where di is a digit from zero to seven. Any arbitrary character can be represented this way.

2. Local (i.e. concealed) use of the newline character that does not go into the computer-stored string on input and is not in the computer-stored string on output is effected by typing \<newline character>.

3. The character \# places the delete control character into the input string.

4. The character \@ places the line delete control character into the input string.

5. The character \\ places a left slant character into the input string.

6. The solid vertical bar ( ) and the broken vertical bar (|) are equivalent representation of the graphic corresponding to ASCII code 174.

The escape conventions described in items 1 through 5 above apply only if none of the characters involved overstruck.

USER_PROGRAM_INTERFACE

For those derail functions that are implemented, it is the goal that slave programs will experience the same interface as they would on native GCOS.

o Programs use normal derail to obtain TSS services

The derail fault will be caught by Multics and passed to GTSS. The machine conditions are examined to determine the nature and validity of the fault. Legitimate derail requests are processed, the IC is adjusted to pass over the derail calling arguments, and control is returned to the user.

o Privileged code not accommodated

The effort to accommodate privileged code is not justified at this time.

o Existing user subsystem, linked object files usable as is.

This must be, as part of the "no conversion" requirement. i.e., recompilation and linking not required

SIMULATOR ACCOMMODATION OF THE GCOS FILE SYSTEM

Many of the GCOS File System functions can be mapped onto Multics. These will be done. However, there are many comprehensive facilities with GCOS for operating system managed file integrity and concurrent access control. These functions are not planned at this time.

FILE SYSTEM NAMING CONVENTIONS

The GCOS Time Sharing System character set for names may be composed of alphanumerics,period and minus signs.

A name consisting of zeroes is specifically prohibited. Blank are not permitted. If multiple word names are desired then the words must be separated by periods or minus signs, not blanks.

A maximum of eight characters or less is length is normally used for file names. Catalog names may be up to 12 characters in length and composed of the same characters as file names.

To access a file with a name longer than eight characters, an alternate name must be given from one to eight characters in length. The renaming is local and temporary.

The Multics GTSS character set for names may be composed of at least one nonblank up to a maximum of 32 characters, chosen from the full ASCII character set.

The greater (>) character is specifically prohibited in entrynames, since it is used to form pathnames. Other characters not recommended for entrynames are:

> less-than (<), asterisk (*), question mark (?), percent (%), equal sign (=), dollar sign ($), quotation mark ("), left slant (\), all ASCII control characters (tab, carriage return, etc) and parenthesis.

Non ASCII characters are not permitted in entrynames.

Entrynames may consist of uppercase and lowercase alphabetic characters,digits, underscores (_), and periods (.). The underscore is used to simulate a space for readability. The period is used to separate components of an entryname, where a component is a logical part of a name. (i.e. a PL/I source segment might be named square_root.pl1).

FILE_SYSTEM_ACCESS_MODES

Multics_Access_Modes

The access modes for segments:

r	read	the process can execute instructions that cause data to be fetched (loaded) from the segment.
e	execute	an executing procedure can transfer to this segment and words of this segment can then be interpreted as instructions and executed by a processor.
w	write	the process can execute instructions that cause data in the segment to be modified.
n	null	the process cannot access the segment in any way.

The access modes for directories are:

s	status	the attributes of segments, directories, and links contained in the directory and certain attributes of the directory itself can be obtained by the process.
m	modify	the attributes of existing segments, directories, and links contained in the directory and certain attributes of the directory itself can be modified; and existing segments, directories, and links contained in the directory can be deleted.
a	append	new segments, directories, and links can be created in the directory.
n	null	the process cannot access the directory in any way.

GCOS_Access_Modes

The access modes for both files and catalogs:

r	read	Allow transfer of information from file to program but not from program to file.
w	write	Allow transfer of information both from file to program and program to file. Anyone with write permission, thus, has read permission.
a	append	Same as read permission.
e	execute	Allow run on file only in time sharing mode. Execute permission is restricted to time sharing mode.
rec	recovery	Allow write when the file is abort locked or has defective space. Also accept MME or directive to abort lock the file or to reset an existing abort lock. Anyone with recovery permission is also given permission to write and hence read.
p	purge	Allow file to be deleted or catalog to be deleted and all subordinate files to be deleted. Anyone with purge can also perform any of the actions permitted by recovery, including write and hence read.
c	create	Allow catalogs and files to be entered as subordinate to this catalog
l	lock	Allow MME or directive to security lock the file or catalog or to remove an existing security lock. A security lock does not apply to a user with lock permission.
m	modify	Allow catalog or file descriptor to be modified. Allow entries to be made in catalog for subordinate files or catalogs. Anyone permitted to modify is allowed to perform any actions. Hence modify includes create, lock, and purge, that in turn includes recovery and hence write and read.
x	exclude	The specified user has no access to the catalog or file.

MAPPING_PERMISSIONS_FROM_GCOS_TO_MULTICS

(create catalog)

		r	w	a	e	p	m	l	c	x
	r	x	x	x	x	x	x		x	x
A	w		x			x	x		x	x
	e	x	x	x	x	x	x		x	x
	s						x			
B	m					1	2		1	
	a						x		x	

(create files)

		r	w	a	e	p	m	l	x
	r	x	x	x	x	x	x		x
C	w		x			x	x		x
	e	x	x	x	x	x	x		x
	s								
D	m					1	2		
	a								

NOTES:

1 Needed to allow deletion by anyone else. This causes problems in that someone can also delete other segments as well.

2 Modify necessary to allow the changing of the file's permissions. However, this allows deletion of all segments under the directory.

A Permissions set on initial acls on segments

B Acls set on the created directory

C Acls set on the segment created

D Acls which must have been set on the superior directory.

Permissions are carried only to next level and are not propagated down through the subtree, as in GCOS.

GCOS_File_Attributes

An interim mechanism is used for the processing of GCOS file attributes. GCOS file attributes are required because GCOS TSS allows the user to specify that default modes of processing are to be applied. File attributes include the sequential/random creation mode and the maximum file size (as opposed to current file size). Files larger than 256K are not uncommon on GCOS.

The appropriate place to store information of this nature is the branch property list, proposed in MTB 210; since this facility is not available and cannot be developed within the time constraints of GTSS delivery, a simpler, user-ring facility is provided.

The attribute data required by gcos_tss will be converted to ASCII representation and saved as added names on the branch. The general form of the attribute name is

```
<entryname>.<attributename>.<attributevalue>
```

There is no problem with overlength names; GCOS file names are limited to 12 characters in length. The added name manipulation functions are isolated in one subroutine, gtss_attributes_mgr_, for easy conversion at a later date.

An exec_com will be provided to allow GCOS-oriented users to manipulate these added names.

See the description of gtss_attributes_mgr_ for details of the interface.

FILE_SYSTEM

The following discussion applies to both the batch and the time-sharing simulators; although the interfaces are slightly different, the functions are the same. References that are simulator-specific are given as such.

The GCOS file system is not simulated. Instead, references to permanent files from $ PRMFL or $ SELECT control cards or from MME GEFSYE are mapped by the batch Simulator into references to files in the Multics file system.

The Multics file system has several similarities to the GCOS file system. Multics files are identified by pathnames, which are analogous to the GCOS file string. They comprise a series of directory names, which are analogous to GCOS catalog names, followed by an entry name, which is analogous to a GCOS file name. Passwords are not included in a Multics pathname.

Multics literature uses the term "segment" when referring to items contained in the Multics file system. The term "file" is used in the special case of a segment that is being accessed by explicitly programmed I/O rather than via the normal Multics method of direct-segment addressing.

References to permanent files can be made from the Simulator via a Multics pathname or via a GCOS file string. The Multics pathname can be used in place of the GCOS file string on the $PRMFL card.

Each file has associated with it an access control list (ACL), which is set by the owner of the file. The ACL can specify combinations of read, write, and execute permissions to individual users or to all users in a specific group.

Access to permanent files from the Simulator is determined only by Multics access control and is based on:

1. The person.project of the process in which the Simulator is running.

2. The access granted to that person.project by the ACL's of the files being referenced.

When the Simulator is running in an interactive user process, access privileges to any permanent files are the accesses granted to that person.project via the ACL's of the referenced files. The presence or absence of a $ USERID control card has no effect on this access. (The $ USERID card is ignored by the Simulator.)

When the Simulator is running jobs submitted by the GCOS daemon, there is a security problem. Normally, access to a Multics process (and the file access privileges that it has) is validated by a password typed by the user at login time. However, a GCOS password is contained on a $ USERID card and, therefore, is much more susceptible to theft. Thus, in addition to the normal Multics file access controls, some additional restrictions are placed on jobs submitted by the daemon to protect the security of the Multics file system (see "Restrictions on Daemon Jobs" for descriptions).

MAPPING OF FILE STRINGS TO PATHNAMES

Because the structure of the Multics file system is different from that of the GCOS file system, the appropriate method of mapping a GCOS file string into a Multics pathname is not an obvious one. The default method used allows many GCOS jobs to run immediately and requires that some initialization be performed in the Multics file system before other jobs can run. Other methods can be specified that provide more flexibility and completeness.

Both the GCOS and Multics file systems are organized in tree-structured hierarchies. However, while the GCOS file system holds only user files, the Multics file system holds the entire Multics system; user files are held in a subdirectory of the total hierarchy.

The user file subdirectory contains project directories and each of these contains individual user directories. The user file subdirectory is analogous to the system master catalog (SMC) of the GCOS file system. However, the project directories that it contains are not analogous to the user master catalogs (UMC) in the GCOS file system, since Multics users are not normally permitted to create or modify files in the project directories.

User directories are more nearly analogous to the GCOS UMC's, but they differ in that they have two-component names, while UMC's have only one-component names. This makes it impossible to map UMC names directly into user directory names without obtaining additional information from some source or making certain assumptions.

The following discussion describes in detail how each mapping method works.

A user directory, which is contained in a project directory, is known as a home directory or a default working directory in Multics terminology. The form of a home directory pathname is:

```
>udd>project>person
```

The greater-than sign (>) is used to separate components of a Multics pathname (instead of the slash (/) that is used in GCOS file strings). The leading > indicates the pathname is relative to the root of the hierarchy rather than relative to the working directory. The directory udd (user_directory_directory) contains all project directories. Every user's home directory is contained in some project directory. For example, the home directory pathname of the user Smith.Applications is:

```
>udd>Applications>Smith
```

User files can be placed in the home directory. Users also can create subdirectories in the home directory and can place files in them to organize and/or restrict access to groups of files. Multics does not associate passwords with individual directories or files; access is controlled only by the ACL of the directory or file in question.

Each process has a working directory. Initially, this directory is the home directory of the user. However, it can be changed by the user.

with Multics conventions, files can be referenced by an absolute pathname or by a relative pathname. An absolute pathname begins with a greater-than sign (>) and contains the names of all the directories superior to the file in the hierarchy. For example:

```
>udd>Applications>Smith>data_file
```

A relative pathname does not begin with a greater-than (>) sign and the complete pathname is assumed to begin with the pathname of the working directory. The simplest example of a relative pathname is an entry name (analogous to a GCOS file name):

```
data_file
```

This identifies the same file as the previous example, provided the working directory is:

```
>udd>Applications>Smith
```

Similarly, a GCOS catalog/file string that uses a leading UMC name can be considered to be an "absolute pathname" and a file or catalog string that does not have a leading UMC name is considered to be a "relative pathname".

Multics absolute or relative pathnames can be used on $ PRMFL and $ SELECT cards. They are interpreted as previously described. If a GCOS file string is used on one of these cards or in a MME GEFSYE, it is mapped into a Multics pathname.

Rules common to all mappings from GCOS to Multics follow:

1. All passwords, along with the dollar signs that precede them, are removed from the string and ignored.
2. All slashes (/) are changed to greater-than signs (>).
3. If no catalog string precedes the final string, the final string is appended to the pathname for the user's current working directory.

Home_Dir_Mode

The first catalog name in the string (that of the UMC) is replaced by the home directory pathname (not the working directory pathname).

Therefore, the file string

```
SMITH/JONES$CAT/Y$DOE
```

is transformed to the pathname

```
>udd>Applications>Smith>jones>y
```

for the user Smith.Applications. Note that the retained portion of the file string is indicated in lower case letters, while the original file string is indicated in all capital letters. This illustrates a common situation in which a file string is encountered on a BCD card that was used as input via the GCOS daemon. (Alphabetic BCD characters are translated into lower case ASCII characters for internal processing by the Simulator.) However, if the input is a Multics ASCII file, the original characters (upper or lower case) in pathnames and file strings are preserved. A complete description of the use of the ASCII and BCD characters sets is included in Section II.

Similarly, if the file string were just

    yyy

and the user's current working directory were >udd>Applications>Smith>smith, the resulting pathname would be >udd>Applications>Smith>smith>yyy.

Problems that arise while mapping file strings into pathnames (while accessing the files of another user) can be solved in two ways:

1. Repunch the cards using Multics pathnames.

2. Place links in the home directory, which points to the files of interest in the other user's directory. (See the MPM manuals for information on this.)

For upward compatibility, the home_dir mode is the default mode for both batch and time sharing simulators.

Working_Dir_Mode

This mode is nearly identical to the home dir mode. The first catalog name in the string (that of the UMC) is replaced by the the working directory pathname.

If the user's working directory is >udd>Applications>Smith>temp_dir, then the file string

    SMITH/JONESCAT/YDOE

is transformed to the pathname

    >udd>Applications>Smith>temp_dir>jones>y

UMC_Dir_Mode

This mode of pathname mapping converts the leading UMC name in the GCOS catalog/file string into the string ">udd>umc_name>umc_name". The purpose of this mode is to allow direct mapping of pathnames in either direction with no explicit action on the part of the individual user. This mode is used for the loading of GCOS user SAVE tapes onto Multics.

It does require that the Multics System Administrator add the lower-case version of the UMC name to the project directory under >udd. The Project Administrator must create a directory by the same name directly below the project directory. This second directory is the equivalent to the catalog for the given UMC on native GCOS.

Example:

A project on GCOS has the UMC name of DEBUG. This project is also registered on Multics, but with the name GDEBUG. The following steps must be taken to use the UMC_dir_ mode of pathname mapping:

```
1.   add_name      >udd>GDEBUG   debug

2.   create_dir    >udd>debug>debug

3.   set_acl       >udd>debug>debug sma *.GDEBUG

4.   set_iacl_seg  >udd>debug>debug rw *.GDEBUG
```

With this mode, the GCOS catalog/file string of SMITH/JONESCAT/YDOE would be mapped to:

```
>udd>smith>smith>jones>y.
```

SMC_Dir_Mode

This mode of pathname mapping sets a directory pathname specified by the user to be the SMC for all subsequent mappings. All UMC's will be looked for or created directly under the SMC directory. Thus, the user has complete control over the mapping of GCOS catalog/file strings to their targets on Multics. Typically, the user will create links with the names of UMCs that point to the corresponding directories containing the desired subcatalogs and files.

Example:

The user has specified "-set_smc_dir_mode >udd>GDEBUG" in a command invocation of gcos_tss. With this mode, the GCOS catalog/file string of SMITH/JONESCAT/YDOE is mapped to >udd>GDEBUG>smith>jones>y.

GTSS SYSTEM FILES, DATA BASES

The following paragraphs describe the various files and data bases maintained by GTSS. Some of these are also directly addressable by user software; for example, the system builds the SY** file from the user's terminal input and then passes it to the BSED subsystem for merging into *SRC.

o bound_gcos_tss_ gcos_tss, gtss

  This module contains all executable modules. There are no other bound modules for GTSS.

o gtss_ext_

  This module contains gtss per-process control information. The gtss data base gtss_ext_ is a Multics object containing entries (external variables) used to communicate information from one gtss module to another. These variables are only relevant to the GTSS implementation and would not be known to a user of gtss.

  The include file gtss_ext_.incl.pl1 provides the centralized declaration (PL/I) for these variables. There are a number of structures ($flags, $statistics, $aft, $fast_lib) relating to particular GTSS functions. The remaining are scalar variables used to regulate unrelated functions. There is comment information in the include file to designate particular usage.

o gtss_tfa_ext_

  The file gtss_tfa_ext_.incl.pl1 contains the declaration of the data structure, gtss_tfa_ext_, to provide an array of file attribute structures. A row is provided for each of the potential 20 files GCOS time-sharing allows. Each structure (row) provides a set of values that designate the "attributes" of the corresponding file in the AFT.

  The attributes (see gtss_file_attributes.incl.pl1) provide information about a file's size, type of device, blocking, whether random or sequential (linked), whether it is a permanent file or temporary, and "user" attributes provided by the user of the file.

o gtss_install_values_

  This separate segment contains those runtime values used by

GTSS that may be changed by a site administrator. Primarily, this data is used to find the GCOS system software. See below for a description of the maintenance of this data.

o gtss_prgdes_ext_

The gtss_prgdes_ext_ data structure contains information which is automatically extracted from the TSSA module of native GCOS timesharing by the use of editor macros. The information consists of the program descriptors for timesharing subsystems, the command language lists for these subsystems, and the lists of primitives to be interpreted for each timesharing command. This information is used primarily by the gtss_interp_prim_ module of gtss. Section IV of the TSS System Programmer's Reference Manual, DD 17C, REV 0 describes the functions of the program descriptors, command language and primitives.

Data in gtss_prgdes_ext_ is logically equivalent to the corresponding data structures in TSSA but the exact storage layout has not been maintained. For example, the program descriptors have been expanded from 9 words to 12 although the first 9 words still contain the information the user would expect to obtain with a DRL PRGDES. This extension in format is transparent to the user.

o gtss_ust_ext_

The gtss_ust_ext_ data structure represents the user status table (UST) as maintained by native GCOS timesharing. All of the same fields are defined as provided by native GCOS and values are stored in the user status table in imitation of native GCOS timesharing. The user status table is documented in Section II of the TSS Program Logic Manual DB84A, Rev 0.

o sy**, *src, tap*

GCOS allows the use of the asterisk in file names. Because this conflicts with the Multics star convention usage, asterisks are converted to the plus character (+) when generating Multics entrynames.

These files are user-visible system files. They are fully described in the GCOS TSS System Programmers' Manual, DD17. Briefly, the sy** file is the collector file; it collects the user's raw terminal input at system level. The *src file is the user's current file. It contains the old current file data, merged with any new inputs. The tap* file is used to collect bulk input from paper tape. Bulk input refers to the reading of multiple line input without line breaks, with a single read request.

o Command, subroutine libraries

This category refers to files supplied by the user in addition to the system supplied procedures.

The user can specify that a certain command library be searched with specific command syntax. In addition, GTSS will search the file gcos_second_software_ if the -userlib control arg is given in the gcos_tss command line.

o gcos_system_software_
gcos_library_subroutines_

These are the system supplied procedures. The files of gcos_system_software_ are pre-linked slave programs such as the FORTRAN compiler, the abacus subsystem, etc. The files of gcos_library_subroutines_ are the run-time support library routines. These two files are also used by the batch simulator and are taken from the GCOS release system tapes.

o gtss_slave_area_seg_ (1-4)

This segment contains the executable GCOS object code. There are actually four different segments used to implement the three-level subsystem push/pop facility. This effectively accomplishes the GCOS TSS swap-out/swap-in mechanism.

USER_FILES

All files, including user files, are stored within the Multics virtual memory storage system. Since GCOS users can only access their files by doing "physical I/O" into their buffer space, the simulator can make the physical differences in the two file systems transparent to the user.

o Normal GCOS "AFT" reference to files

GCOS TSS users must first (implicitly or explicitly) access the files to be used and place the necessary control information in the available file table (AFT) on a per-user basis. The AFT is contained strictly within the GCOS TSS executive's privileged space; users cannot directly address this information with their programs.

The GTSS interface is identical to that of native GCOS provided that the permissions requested are only read, write, execute, and append. The other forms of access provided by native GCOS, such as conflict control and file recovery, are not provided.

o Catalog designation mapped to Multics pathname

See below for a comprehensive description of the pathname mapping facilities.

o GTSS files are identical in content to those on GCOS

As mentioned earlier, it is a primary goal to store the GCOS file content in a format completely identical to that of native GCOS. This eliminates all possibility that the data read and written will not conform to what is intended. This caution is based on the fact that any file, regardless of how it is created and filled with data, can be accessed randomly, in I/O records of any length. The initial word address being mod 64 is incidental.

o GTSS files can be referenced by user Multics software

It is planned to provide an I/O module interface so that Multics programs can easily access GCOS-format data bases. It is easy to write procedures that process these files; the gcos_card_utility command already does this.

GTSS_ERROR_PROCESSING

There are three levels of error processing that should be addressed: it is planned to take advantage of the Multics system features to make error reporting more comprehensive than is available on native GCOS.

- Errors found by slave object code

  These errors are detected only by GCOS software and thus must be reported as-is. Error messages that are specified by number to the executive are identical to that produced on native GCOS.

- Errors found by GTSS simulator

  Since the simulator performs services on behalf of the GCOS object programs, errors detected can be reported to the interactive user before returning to the GCOS procedures. This provides the user with the opportunity to correct the situation and continue execution rather than aborting the function.

- Errors found by Multics

  Errors first detected by the Multics operating system will be reported back to GTSS. GTSS can then attempt correction of the situation as before.

GTSS PERFORMANCE

This section attempts to provide a preliminary insight of GTSS performance. There has been no exhaustive analysis prior to the start of implementation to determine what the GTSS performance would be; however, past experience can give some justification to this insight.

The reader is reminded that GTSS is not intended to be equal in performance to native GCOS TSS. The implementation as a user-ring facility, subject to all the normal user interfaces, precludes total optimization of the interfaces. Within these constraints, however, every effort will be made to be as efficient as is reasonably possible.

o Direct execution of user and slave system software

  Since the Multics CPU is essentially a superset of the GCOS CPU, even when running in Multics mode, direct execution of the GCOS object code is used. Thus, only the additional address formation time for the virtual memory need be considered. Multics derives some benefit from a more selective cache control.

o Similar processing of derail "faults" by the operating systems

  Both systems intercept faults in the same manner; Multics is required to store and restore certain additional register information.

o Efficient file, terminal I/O

  The batch simulator uses 50-100% more CPU time in the servicing of user I/O requests than native GCOS. For this reason, an new I/O mechanism is used, tailored to the interfaces of the batch MME GEINOS and TSS DRL DIO. This new mechanism uses a subroutine interface rather than a Multics I/O module interface, and is expected to give a considerable improvement in performance over the old mechanism.

o Optimized PL/I very efficient

  The entire simulator is written in PL/I, with the exception of a very few lines of ALM code for BCD/ASCII translation and the transferring of CPU control to the GCOS object programs (Transfer and Set Slave). This provides for easily optimized code, both in the optimize feature of the compiler and the profile monitoring facilities.

o Simultaneous, multi-processor execution of GTSS

Since GTSS runs in a normal user process, it can be run on any and all processors simultaneously. Multics provides somewhat better processor utilization than GCOS; one reason is that any Multics CPU can answer any interrupt.

o Virtual memory file processing

Since only the busy pages stay in core, there will be some efficiency in main-memory utilization. If programs are written to run on GCOS but only the simulator on Multics, more efficiencies can be gained by assuming program sizes up to 255K (less the space for the loaders) do not require overlay processing. This has been used to advantage in the batch simulator by programs that use the free space provided to the program with the $LIMITS card.

o Terminal type-ahead

This Multics feature is usable under GTSS. As with normal Multics users, this feature is most useful with full duplex terminals.

o Extensive metering and tuning facilities

GTSS can take advantage of all Multics tools for performance measurement and enhancement.

o System Scheduler

The Multics system scheduler can be used to provide guarenteed and controlled responses to individual users or groups of users.

o Paging Consideration

It is possible that certain I/O on native GCOS that is overlapped with program execution (asynchronous I/O) may be done in a synchronous mode on Multics. This is necessary to ensure that the page containing the physical I/O buffer is in main memory.

TRANSITION_TOOLS

In addition to the gcos_tss command itself, other commands are provided to facilitate the movement of programs and data between GCOS and Multics.

Batch_Simulator

The batch simulator complements GTSS in its functions. Most slave user functions are available. The current MME and control card level of the batch simulator is 2/H. An upgrade to 3/I and then 4/J is in process.

Files not created by GTSS, but intended for use by GTSS, must have their file attributes defined with added names on the branch. An exec_com is provided for the user to manually set these attributes until all GCOS tools provide this function.

Multics_Facilities

Many additional tools already exist in Multics that can assist in GCOS and GCOS TSS program development. The GCOS-related tools are listed immediately below.

1) gcos_card_utility

2) gcos_pull_tapefile

3) gtss_library_mgr (glom)

4) gcos_fms Save/Restore Utility

5) I/O modules for direct access of GCOS files and tapes by Multics Programs (in planning)

COMMAND_AND_SUBROUTINE_DESCRIPTIONS

The following pages describe the various commands related to the GCOS TSS Simulator and some of the more definitive subroutines. The following modules are included:

```
gcos_tss
gcos_debug
gcos_fms (described in a separate MTB)
gcos_library_mgr (glom)

gtss_attributes_mgr_
gtss_expand_pathname_
```

Name: gcos_tss, gtss

The gcos_tss command invokes the GCOS TSS environment simulator to run a single GCOS TSS user, immediately, in the user's process.

Usage

```
     gcos_tss {-control_args}
```

where control_args can be selected from the following:

-gtss_umc umc_name
: set an internal parameter for UMC name to umc_name. This value is required for certain implicit GCOS TSS functions.

  If this option and its argument are not given when either the -set_smc_dir_mode or -set_umc_dir_mode control arguments are given, gcos_tss will request the umc_name before continuing.

-set_multics_break_mode, -smbm
: sets a mode in gcos_tss to cause the user's process to go the Multics command level whenever the user hits the break/interrupt key on the terminal. The user can then type any number of Multics commands for immediate execution. Execution of gcos_tss can be resumed by typing "start".

  If the user types "program_interrupt" ("pi") after quitting GTSS, gcos_tss will reset execution of the current TSS command/subsystem as in native GCOS TSS.

Control arguments specifying the disposition of output from the simulator:

-list, -ls
: convert APRINT and BPRINT print files (both are SYSOUT) from GCOS ASCII and BCD, respectively, to Multics ASCII and delete the intermediate copy but do not submit the dprint request for these files. (This conversion is performed by a call to the gcos_sysprint command for each file.)

-dprint_options "options", -dpo "options"
: queue the converted print files for printing by the I/O daemon, but use the dprint

control arguments supplied in the options string instead of the default of -delete. The options must be enclosed in quotation marks if they contain blanks or other delimiter characters recognized by the command processor. The dprint command is called via cu_$cp so that a user-defined abbreviation for dprint (that supplies default heading and destination arguments, for example) would be used in this call. Use of this control_arg overrides the use of the -list and -hold control_args.

-raw
: convert BPUNCH punch files from BCD to an internal format suitable for punching by the Multics I/O daemon in raw mode (960 bits per card image) and delete the BCD copy, but do not submit the dpunch request for these files. (This conversion is performed by a call to the gcos_syspunch command for each file.)

-dpunch_options "options", -dpno "options"
: queue the converted punch files for punching by the I/O daemon, but use the dpunch control arguments supplied in the options string. The -raw argument is always used for dpunch, since the converted punch files are not suitable for punching in any other mode. The explanations under -dprint_options above, regarding quotation marks and abbreviations, apply to this argument as well. Use of this control_arg overrides the use of the -raw and -hold control_args.

-hold, -hd
: do not perform the default conversion and daemon output of print and punch files. The default is:

```
     -dpo "-dl" -dpno "-dl -raw"
```

Since the default for each file type (print or punch) is overridden when any of the above arguments are specified for the given file type, the -hold argument is only required when one of the file types is to be left in GCOS standard system format, with no conversion or daemon output being performed.

Control arguments governing the creation of files by the simulator:

-temp_dir path, -td path
: use the pathname of a directory specified by path for temporary GCOS TSS files. By default, the process directory is used.

-syot_dir path, -sd path
: use the pathname of a directory specified by path for the GCOS TSS format copies of print, punch, and sysout files. By default, the working directory is used. (The converted copies of these files are always placed in the working directory.)

-set_smc_dir_mode path, -ssdm path
-set_umc_dir_mode, -sudm
-set_working_dir_mode, -swdm
-set_home_dir_mode, -shdm
-reset_dir_mode, -rsdm
: Refer to "Mapping of File Strings to Pathnames" in Section I for a description of these control arguments.

Other control arguments:

-userlib
: enable the use of GCOS slave software libraries supplied by the user instead of, or in addition to, the copies of the libraries installed in the system. The use of this argument is described in Section II under "DATA BASES".

-debug, -db
-probe, -pb
: inform the simulator that: 1) it is being run interactively; 2) by a user who is familiar with the Multics debug or probe command, respectively, and other Multics error recovery facilities; and 3) the user wishes to be given the opportunity to use the facilities to determine the cause of, and possibly correct, any error that would otherwise cause the simulation of the job to be aborted.

-trace args
: trace the events specified by args, where args can be one or more of the following:

    derail, drl, causes the derail name and its location in the execution program to be written to the user-output switch.

subsystem, ss, causes the name of the called subsystem to be written to the user_output switch.

-gcos_debug, -gdb path
where pathname specifies the Multics segment to be used for the gcos_debug command data base. If the entryname of path does not have the suffix gdb, it will be appended.

See the description of the gcos_debug command earlier in this manual for a definition of the gcos_debug control syntax and functions.

Name: gcos_library_mgr, glm

The glom command "obtains" modules from a GCOS library segment (commonally a multi-segment file). Modules may be extracted from the library to form a fast search library and (or) a segment containing information about the location of GCOS objects on the library, or the library extracted onto, (in terms of Multics msf component numbers and segment offsets) can be obtained.

Syntax: glom in_lib {-nm module_name ...} {-ol out_lib} {-cf names_seg} {-no_cat} {-pr_cat} {-olli olli_path} {-brief}

Arguments:

in_lib   Name of segment or multi-segment file input library. This file can be copied from a GCOS total system tape, or it might be a simulator format library commencing with a catalog as produced by the gcos_build_library (gcbl) command. It is assumed to begin with a catalog unless the -no_cat control argument is given.

Control Arguments:

-nm   module_name is an library object name (<= 6 characters) specifying the modules to be obtained. This list of names is catenated with names supplied in the names_seg (see -cf control argument). If no names are supplied by either option then all modules on the library is implied.

-ol   out_lib designates that the modules are to be catenated to the named segment or multi-segment file. If this file does not exist it will be created. out_lib can be a full pathname.

-cf   name_seg is a Multics segment containing a list of module names (Note: module_name option above). This segment must (only) contain one name on each line, no white space and no empty lines. This segment can be formed by executing gcos_library_summary (gcls) or glom (itself) under the file_output (fo) command and editing the report produced.

-no_cat   Designates that the input library does not commence with a catalog.

-pr_cat   Print input library catalog information (names and locations). No catalog will be printed if the caller specifies both -pr_cat and -no_cat. If -brief is specified along with

-pr_cat and there is no output library nor and module names specified (either in the command line or in any -cf file) printing the catalog will be the only activity of glom.

-olli olli_path is the name of the file that the output library list of information is put into. This file will be overwritten if it already exists. This file is printable information designating each module placed on the output library, in which multi-segment component it was placed, the offset to the object and information provided for the gtss fast library loading process.

-brief, -bf Do not report each module moved to any output library (on error_output).

Notes: The glom command incorporates functions of the gcos_library_summary and gcos_extract_module commands. The glom command uses the msf_manager_ subroutine to manage its files rather than ios_ used by the latter two.

Name: gcos_debug, gdb

This Multics command provides for a debugger to be used in conjunction with the GCOS simulator (See: help gcos).

Introduction: "gdb" is an online debugger to work specifically for callers of the Multics GCOS simulator The gcos simulator, i.e., the "gcos" command, calls upon gdb when failure occurs. The user then types in a series of instructions to direct the debugging activity. Upon termination (the gdb "quit" command) the simulator concludes its execution. gdb "knows" the segment used by the simulator to simulate gcos memory. In addition gdb "knows" how file control blocks and (GCOS Version 1) PL/I automatic stack frames are linked together in the simulated memory.

Calling gdb: The Multics "gcos" command provides for the "-debug" (or "-db") control argument. When the GCOS simulator, i.e., the "gcos" command, is called using this argument AND there is failure in execution the simulator calls the procedure "db". Normally calling a procedure "db" would result in the Multics "debug" being called. Anticipating this circumstance the gcos debugger, "gdb" (also with entry "db") can be called instead.

To provide for calling gdb the following steps should be taken: BEFORE calling "gcos" (and providing that it is called with the "-debug" or "-db" option), terminate any reference to "db":

```
tmr db
```

then initiate the gcos debugger:

```
initiate >udd>Gcos>gdb>db
```

then execute "gcos .... -debug ...". Note that the terminate, "tmr", is only required if a reference to "db" has been set previous to the gcos debugger being called. The initiation of "db" in gdb is only needed once during each Multics "process", i.e., once "db" is initiated "gcos" can be called many times.

The caller of "gcos" is signalled that the debugger is about to be called by the sequence:

CALLING DEBUG:

output on the terminal. At this point the gcos debugger awaits the user typing instructions to be carried out. Typing "?" will cause a "help" session.

It is also possible to call gdb directly as a Multics command (at either its "gdb" or "db" entry, they are the same). In this case the process directory segment used to simulate gcos memory must be "viable", i.e., the simulator must be in the process of execution. This will be the case if the execution of "gcos" is interrupted or if it does not conclude normally and NO "new_proc" has been executed.

Using gdb: Once gdb (entry db) has been called by the gcos simulator (as described above), gdb expects the user to supply instructions to be done. The user types the instructions at his terminal. The instructions are typed in a free format. They are separated from each other by semicolons (";") or newlines ("return" key). Each instruction is in one of three forms:

1. an address followed by a command,
2. just a command, or
3. just an address.

In case 2 the command either requires no address or utilizes the last address specified. Case 3, just an address defaults the command to being the octal dump. Commands fall (loosely) into three categories:

1. requests for information about the current state of the execution of the gcos simulator, i.e., information about the user's programs under simulation,
2. requests for information about the state of the gcos debugger, and
3. escaping to call upon facilities outside of the debugger (escape to Multics command level) without exiting from the debugger (or the simulator).

If commands are typed that are unknown to gdb (spelling mistakes, whatever) the user is requested to supply a substitute word. In addition the caller can always interrupt gdb (depress the "interrupt" or "break" key) and then type "pi" ("program_interrupt"). These two steps will place the caller back in gdb ready for another instruction.

To provide assurance that the GCOS debugger is in use when an input line contains only a back-slash question mark ("\?")) the message "gcos_debugger" is printed.

Addresses: For various gdb commands the user must indicate to what series of gcos memory words they are to be applied, e.g.,

0,100 bcd

designates displaying memory locations from octal 0 through octal 100 as bcd character values.

Addresses are in one of 3 formats:
1. first address followed by a comma followed by a last address,
2. first address followed by a colon followed the number of words, and
3. just a first address.

In the last case (3.) the first and last address are the same. The first and/or last address are in the form of optionally signed "expressions". In case 2., the number of words is an unsigned expression. The simplest form of expression is a numeric constant. If the number ends with a period (".") a decimal number is specified, otherwise the number is an octal number.

An expression can be a single "value" or a series of "values" operated upon by the operators: +, -, * (multiply), / (divide), or | (modulo). A "value" can also be parenthesized expression. As was stated the simplest form of "value" is a decimal or octal number constant. A "value" can also be whatever is contained in a specified register, e.g., x3 as a "value" implies using the contents of index register "x3". The "a" and/or "q" register can be specified or the contents of either of these two register's upper or lower half, e.g., a u implies use the upper (left) half of the contents of the "a" register as a value.

The address expression value is always biased by (added to) a current "offset" value. The offset is initially zero and it can be set by the "offset" command.

Commands: The following is a description by command. Note that any command may be preceded by an "address". In some cases, e.g., the escape command, the address information is not used, though it does reset the

current default address values. No comment is made if the address was not required. The command keywords usually have 3 forms:
1. a single letter,
2. a 3 character mnemonic and
3. a "long" form, e.g., "d", "dec" and "decimal".
The "? commands" prints a table of commands.

escape: (e | esc | escape) Followed by one or more spaces (or tabs) results in the remainder of the line being sent to the Multics command processor (through the abbreviations).

ascii: (asc | ascii) Print selected memory words as 4 ascii characters apiece.

bcd: (b | bcd) Print selected memory words as 6 "bcd" characters apiece.

decimal: (d | dec | decimal) Print selected memory words as decimal numbers.

fcb: (fcb) Print memory selected by the first address as a file control block (at its zero-th entry).

float: (f | flt | float) Print selected memory words as floating point numbers.

huh: (huh) Display various debugger control values, e.g., current first and last address values.

instruction: (i | ins | instruction) [NOT IMPLEMENTED] Print selected memory locations in assembly language mnemonics.

list file control blocks: (lfs | list_fcbs) Starting from memory location (octal) 17 trace the linked list of current file control blocks.

list PL/1 stack frames: (lss | list_stacks) Starting from memory location (octal) 37 trace the linked list of PL/1 (automatic storage) stack frames. Note: gdb is currently oriented only to the Toshiba PL/1 compilers implementation of stack frames, this is not compatible with the GCOS PL/2 utilization.

lower: (l | low | lower) Qualify that the "lower" (right 18 bits) part of the "a" or "q" register contents are to be used.

no operation: (n | nop) "Do nothing" debugger command. Provides for resetting the current address values without designating any overt action.

octal: (o | oct | octal) Print selected memory words in octal (12 octal digits apiece). Note that octal is the default command.

offset: (off | offset) Reset the address offset to the value of the first address.

pointer: (p | ptr | pointer) Print selected memory words as PL/1 "pointer" values.

prefix: (pre | prefix) Print information from the SSA prefix.

quit: (qit | quit) Exit (return) from the gcos debugger. After this command is executed the remainder of the GCOS simulator execution will proceed.

registers: (reg | regs | registers) Print the contents (in octal) of all register contents.

stack: (stk | stack) Print memory selected by the first address as a PL/1 (Toshiba) stack frame.

upper: (u | upr | upper) Qualify that the upper half (left 18 bits) of the "a" or "q" register contents is to be used as the value.

help: (?) Typing a question mark as a command ("?") calls the Multics "help" command with the info file for gdb. If the question mark is followed by one or more space delimited keywords then the help is called for those specific entries. Responding with "quit" or "no" to a help request returns you to the debugger. Remember, that if in doubt you can always interrupt and type "pi" to return to debugger command level.

Name: gtss_attributes_mgr_

This subroutine is used to maintain a subset of the GCOS file attributes for files used by the GCOS environment simulators. It does this by using added names on the branch entry to save each of the required attributes. See below for a list of the attributes accommodated and the specific formats involved.

Entry: gtss_attributes_mgr_$set

This entrypoint is used to set initial attribute values and also modify existing attributes.

Usage

```
dcl  gtss_attributes_mgr_$set entry (ptr, fixed bin (35));

call gtss_attributes_mgr_$set (attrib_struc_ptr, code);
```

where:

1. attrib_struc_ptr (Input)
   points to the control structure to be used for setting and resetting the attributes. See below for the structure declaration.

2. code (Output)
   is a standard status return or a gcos_et_ error return.

Notes

None.

Entry: gtss_attributes_mgr_$get

This entrypoint is called to obtain existing GCOS attribute information about a file.

Usage

```
     dcl  gtss_attributes_mgr_$get entry (ptr, fixed bin (35));

     call gtss_attributes_mgr_$get (attrib_struc_ptr, code);
```

where:

1.   attrib_struc_ptr   (Input)
            is as described above.

2.   code   (Output)
            is as described above.

Notes

None.

Attributes_Structure_Declaration

```
/* BEGIN INCLUDE FILE gtss_file_values.incl.pl1
   (Wardd Multics)  08/30/78 1208.1 mst Wed */

/* The gtss_file_values structure provides parameters to the
   gtss_attributes_mgr_ subroutine.

   The caller must provide space for this structure, fill in
   the version with 1, the dname and ename with the file directory
   and entry name, and for calls to gtss_attributes_mgr_$set,
   fill in values to be reset and set the corresponding set_switch
   to "1"b.
*/

dcl  attr_name                  (0:5) char(4) static int
     options(constant)
     init("mode","maxl","curl","busy","attr","null");

dcl 1 gtss_file_values         aligned based(file_values_ptr)
,      3 version               fixed bin(17)
         /* Current version is 1. */
,      3 dname                 char(168)
         /* Directory name. */
,      3 ename                 char( 32)
         /* Entry name. */
,      3 set_switch
         /* "1"b => Set corresponding value. */
,        4 set_ranseq          bit( 1)unal
         /* 0. Set the random/sequential(linked) field. */
```

```
,       4 set_max                bit( 1)unal
        /* 1. Set max size value. */
,       4 set_current            bit( 1)unal
        /* 2. Set current size value. */
,       4 set_busy               bit( 1)unal
        /* 3. Set file as busy. */
,       4 set_attr               bit( 1)unal
        /* 4. Set user attributes value. */
,       4 set_null               bit( 1)unal
        /* 5. Set null file value. */
,       4 not_in_use             bit(30)unal

        /* The above set_ variables should be declared in an order
   corresponding to the value in the attr_name array. */

,     3 data_flags
,       4 mode_random            bit( 1)unal
        /* "1"b => random. */
,       4 busy                   bit( 1)unal
        /* "1"b => file is busy. */
,       4 not_null_file          bit( 1)unal
        /* "1"b => file NOT null. */
,       4 not_in_use2            bit(33)unal
,     3 data_fields
,       4 curll                  fixed bin(35)
        /* Current length in llinks (>=0). */
,       4 maxll                  fixed bin(35)
        /* Maximum length in llinks (>=0). */
,     3 attributes
,       4 not_in_use3            bit( 1)unal
,       4 attr                   bit(35)unal
        /* User specified file attribute value. */
;

/*   END INCLUDE FILE gtss_file_values.incl.pl1 */
```

Name: gtss_expand_pathname_

This subroutine is used to map GCOS-format catalog/file strings into their corresponding Multics pathnames. Various algorithms are used, depending on the current mode setting. The default mode is the home_dir mode. The default mode may be restored with the reset_mode entrypoint.

Usage

```
     dcl  gtss_expand_pathname_ entry (ptr, fixed bin, char (*),
          char (*), fixed bin (35));

     call gtss_expand_pathname_$gtss_expand_pathname_
          (ascii_name_struc_ptr, name_count, dname, ename, code);
```

where:

1. ascii_name_struc_ptr (Input)
   is a pointer to the structure that contains the eight ASCII character strings that specify the GCOS catalog/file path to be mapped into the corresponding Multics pathname.

2. name_count
   is the number of names in the structure that are actually used.

3. dname (Output)
   is the directory portion of the resulting Multics pathname.

4. ename (Output)
   is the entryname portion of the resulting Multics pathname.

5. code (Output)
   is a standard status return or a gcos_et_ status return.

Notes

The Multics pathname generated from the GCOS catalog/file string is dependent on the current mode setting for the procedure. See below for a discussion

Entry: gtss_expand_pathname_$set_home_dir_mode

This entrypoint sets the home_dir mode. A GCOS catalog/file string that does not begin with a UMC name will be mapped to the user's current working directory. A catalog/file string that does begin with a UMC name will be mapped to a Multics pathname where the UMC name is replaced by >udd>Project_id>Person_id, where Project_id and Person_id are the user's login Person_id and Project_id.

Usage

```
     dcl  gtss_expand_pathname_$set_home_dir_mode entry
          (fixed bin (35));

     call gtss_expand_pathname_$set_home_dir_mode (code);
```

where:

code (Output)
is the same as above.

Notes

See the table below for examples of pathname mappings using the various modes.

Entry: gtss_expand_pathname_$set_working_dir_mode

This entrypoint sets the working_dir mode. A GCOS catalog/file string that does not begin with a UMC name will be mapped to the user's current working directory. A catalog/file string that does begin with a UMC name will be mapped to a Multics pathname where the UMC name is replaced by the path for the user's current working directory.

Usage

```
     dcl  gtss_expand_pathname_$set_working_dir_mode entry
          (fixed bin (35));

     call gtss_expand_pathname_$set_working_dir_mode (code);
```

where:

code (Output)
is the same as above.

Notes

See the table below for examples of pathname mappings using the various modes.

Entry: gtss_expand_pathname_$set_smc_dir_mode

This entrypoint sets the smc_dir mode. A GCOS catalog/file string that does not begin with a UMC name will be mapped to the user's current working directory. A catalog/file string that does begin with a UMC name will be mapped to a Multics pathname where the SMC directory path is prepended to the catalog/file string, including the UMC name.

Usage

```
     dcl  gtss_expand_pathname_$set_smc_dir_mode entry (char (*),
          fixed bin (35));

     call gtss_expand_pathname_$set_smc_dir_mode (smc_path,
          code);
```

where:

1. smc_path (Input)
   is the Multics directory pathname to be used as the SMC (root) catalog.

2. code (Output)
   is the same as above.

Notes

See the table below for examples of pathname mappings using the various modes.

Entry: gtss_expand_pathname_$set_umc_dir_mode

This entrypoint sets the umc_dir mode. A GCOS catalog/file string that does not begin with a UMC name will be mapped to the user's current working directory. A catalog/file string that does begin with a UMC name will be mapped to a Multics pathname where the UMC name is replaced by >udd>umc_name>umc_name. The umc_name is in lowercase.

Usage

```
     dcl  gtss_expand_pathname_$set_umc_dir_mode entry
          (fixed bin (35));

     call gtss_expand_pathname_$set_umc_dir_mode (code);
```

where:

code (Output)
is the same as above.

Notes

See the table below for examples of pathname mappings using the various modes.

Entry: gtss_expand_pathname_$reset_mode

This entrypoint causes the mapping mode to be reset to the default mode. The default mode is the home_dir mode.

Usage

```
     dcl  gtss_expand_pathname_$reset_mode entry
          (fixed bin (35));

     call gtss_expand_pathname_$reset_mode (code);
```

where:

code (Output)
is the same as above.

MAINTENANCE OF GTSS INSTALLATION VALUES

The gcos_tss (gtss) facility provides a data base of values sensitive to each installation of gtss. These values are kept in the Multics object segment "gtss_install_values_" (referenced as unbound external variables, gtss_install_values_$varb). This object is created by calling the Multics command create_data_segment (cds). The source gtss_install_values.cds is provided for this call. The include file, gtss_install_values_.incl.pl1, contains a data structure declaration whose initialization values designate the current set of installation values. To provide a change in these values the following steps are taken:

1. Change any existing initialization values appropriate in the segment
   gtss_install_values_.incl.pl1.

2) Execute the Multics command:
   create_data_segment gtss_install_values_ -list

Note that new variables can not be introduced into the the gtss_install_values_ structure (this implies new facilities that must be coded into gtss), nor can the attributes be changed. The order of the level 2 variables can be changed.

The GTSS implementation utilizes the declarations in the segment gtss_install_values_.incl.pl1 for access to the actual external variables in the object at runtime.

The object produced from the create_data_segment execution must be "found" when gtss is executed (i.e., it is not bound into gtss). This is usually accomplished by ensuring the object segment is in the same directory as gtss. With gtss_install_values_ not bound into gtss, there is the opportunity for many versions to be available. The version desired can be initiated before gtss is called and terminated with the corresponding Multics commands.

The cds execution will produce the segment gtss_install_values_.list, reflecting the cds execution. This segment can be dprinted to provide a record of the alteration of the installation values. The execution will reflect if the data structure for gtss_install_values_ is acceptable. Either PL/I syntax errors will be reflected (on the error_output switch) or a message indicating the number of words in the object data structure (on the user_output switch).

APPENDIX A

BELL CANADA SPECIFIC REQUIREMENTS

The following (edited) text is from Sandy Bartlett of Bell Canada. It is not an official memorandum but it does contain certain requirements as seen by one of the system programmers at Bell. It is included here to indicate the level of compatibility that is desired by at least some of the users of native GCOS TSS.

Note that the MTB states that item one is not planned. Also, GTSS will always return to Multics command level when the user types BYE.

1. The GCOS erase kill proposed will not meet our needs. In fact we feel it would cause confusion as the line kill character is a non-printable character whereas on Multics you can see it! We feel that if this approach were used it would be better to use standard Multics erase kill.

   For this reason we will require the standard GCOS erase character "@" to erase only the preceding character and the standard GCOS kill line ctrl'x' to be treated as a carriage return and followed by a "DEL carriage return-line feed".

   We feel that the erase could probably be fairly easily handled in a tty_ module and the kill should not be difficult to implement in the fnp (maybe as an extra option to set_tty such as delecho). Even with these changes type-ahead will probably still be possible.

2. Papertape should be simple to implement. There are two modes of papertape input, file and command.

   To input a papertape file the user types TAPE and the system responds with READY followed by "carriage-return, line-feed, x-on". The user's tape will then start and continue until it transmits "x-off" which will terminate the file input mode. The actual input will consist of lines of data terminated by "carriage-return line-feed". Data lines will possibly be preceded by rubouts which are ignored.

To input commands from papertape the system must have "x-on" as the last prompt character (GCOS prompts with "carriage-return, line-feed, asterisk, x-on"). This could be done simply with the general_ready command for command mode. DRL KOUTN could add "x-on" to the output text so programs could receive their input from papertape also. Each line of the user's tape would consist of data followed by "carriage-return x-off" possibly followed by rubouts which are ignored.

3. File system permissions required are read, execute, and write for files as append is not implemented under GCOS. GCOS catalog permissions CREATE, PURGE, and MODIFY should map to Multics append, modify-append, and status-modify-append, respectively.

   We use all access permissions except TEST and RECOVERY (APPEND results in RW). We do not use GCOS permissions RECOVERY or LOCK and we do not allow DEVICE specification. We do not use ABORT, VERIFY, AUDIT, INCRSAVE, PAGESIZE, RDERR, and ACCESS/RWW/ or ACCESS/MONITOR/.

   We allow ACCESS/CONCURRENT/ which is really only a flag indicating that the file can be opened with multiple writers and readers. Concurrent access is handled entirely by the program. The file is created with this attribute and is handled normally except when accessed with CHANGING permission in which case multiple writers and readers are allowed. A file accessed with WRITE permission MUST NOT be available to anyone else (i.e. a file can have many readers and no writers or only one writer and no readers [excluding the writer] unless CONCURRENT and CHANGING are used).

4. We will require the full CARDIN system but on a lower priority than FORTRAN and BASIC. We will not require system ALGOL, JOVIAL, or DATABASIC as we do not use them. We also do not use any database managers (e.g. IDSQ).

5. A separate GTSS logon would not be required if all users of GTSS were logged on the same project. For this purpose Multics logon would suffice. All Person_ids registered on this common project would be synonymous with the GCOS user master catalog (UMC). The GCOS simulator should be changed to reflect the GTSS file mapping.

   We feel that a GTSS logon facility would be useful for an identical TSS. This could be done by having the initializer recognize a special logon command such as "gtss". In this manner the initial_command in CMF would be "gtss" and the user who did "logout -hd" would enter "gtss" instead of "login". This "gtss" command would only be recognized by the initializer and would cause a gtss process to be created which would perform the GCOS TSS logon sequence. A set of lines could be assigned to the GTSS process so that a user who dialed one of these special numbers was logged directly on to GTSS. From the user's point of view it would look as if he logged on to GCOS TSS.

A feature that would be useful in GTSS would be the ability to return to Multics. This could be done with a command such as "Multics" which would be synonymous with "logout -hd".