```
┌-----+-----┐
| B U G S |
└-----+-----┘
GMS
```

Graphic Monitor System (GMS)

User's Guide

GMS is a software package designed to operate on the Meta 4A with the extensions provided by the Level 0 Extended Machine. GMS provides program loading, supervisory services, I/O and utility program support. A knowledge of the Meta 4 Level 0 Extended Machine is assumed.[1]

February 4, 1977

---

[1]The research for this project was done under a grant from the National Science Foundation (GJ-28401X) and a contract from the Office of Naval Research (n00014-67-A-0191-0023). Reproduction in whole or in part is permitted for any purpose of the United States Government.

TABLE OF CONTENTS

1 INTRODUCTION

The Graphic Monitor System (GMS) provides supervisory services to the BUGS user not prcvided by the Meta 4A Level 0 Extended Machine. These services facilitate a number of commonly required functions including program storage and loading, file manipulaticn, and input/putput to card reader, operator's console and disk.

All readers are assumed to be familiar with the Meta 4A Level 0 extended machine, including data control and extended instructions, excluding extended I/O instructions. Readers planning to perform S/360 or Meta 4B I/O are required to be familiar with Level 0 support for I/O including the event handling mechanisms.

2 GMS

The Graphic Monitor System (GMS) is one of a number of possible supervisory programs which run on the Meta 4A Level 0 Extended Machine. GMS provides a framework within which a user's programs are loaded and executed. In addition, GMS provides a number of utility programs which a user can run either from the console or directly from another program.

2.1 GMS ENVIRONMENT

GMS runs on the Meta 4A Level 0 Extended Machine. This machine has a minimum configuration of a Meta 4A with 16K bytes of storage, Meta 4A Control Panel, Model 1444 disk, a console terminal, and Model 3461 Card Reader. GMS resides on one or more of 4 supervisor program files named GMSNUCn. The version desired is indicated by the Control Panel switch setting (bits 14 and 15) entered during the IPL sequence.

2.2 GMS FILES AND SVC'S

2.2.1 GMS FILES

Data and programs are stored in files on disk. These GMS disk files consist of an arbitrary number of fixed length records. Files can have records of arbitrary length greater than or equal to 16 bytes. Each file has associated with it an eight-character name and a four-character type. Files are required to have unique file name, type pairs. GMS utility names and supervisor program names (GMSNUCn) are reserved for system use. Only one system defined file type is used - module (MODU).

GMS programs residing on disk as MODU files can be invoked via GMS either from the console or directly from a user's program. All GMS utility programs reside on separate disk MODU files under their respective names. All user programs which run under GMS must be contained in such a file. For the creation of such files see the GMSLINK and CHARON manuals or the OFFLINE READ utility in Section 4.

GMS files can be manipulated via GMS SVC's and GMS Utility Programs called either from the console or from the user's program. SVC's are available to read or write individual records of a file. Records may be randomly read or written over the whole extent of the file. In addition, files may be extended one record at a time. Utilities are available to create, merge, erase, examine, and alter files.

2.2.2 GMS SVC'S

Supervisor calls (SVC's) provide the mechanism whereby a user program communicates with GMS. Supervisory services available from GMS fall into three categories: unit record I/O, disk I/O, and program invocation. The unit record equipment supported includes the card reader and the terminal (console). The disk I/O supports GMS files with a single access method for reading and writing random fixed-length records. The third category of SVC allows the user to call any other program which exists as a separate GMS file. The usage of the various GMS SVC's is described in the next section.

2.3 GMS OPERATION

GMS is IPLed from the Meta 4A Control Panel. Once IPLed, GMS will first print out an initial message on the current command source and then enter command mode. The command source is either the console or the Vector General, depending on the setting of the Terminal/VG switch (in the Black Box next to the terminal). If the switch is left, the terminal is used, if right, the Vector General is used. In general, "console" will mean the current command source in the discussion below.

The terminal can also be used as a 1200 baud CP/CMS terminal. The lights on the Black Box indicate if the terminal is in CP/CMS mode or GMS mode. Pressing the white button switches from one mode to the other.

Messages from the 360 will print immediately on the terminal and put the terminal into CP/CMS mode. Messages from GMS are not printed until the terminal is in GMS mode. The GMS light will blink if there is a message pending. (Note that a circumflex, capital N, is the logon character for 1200 baud.)

In command mode, GMS prints out a '*' character and waits for entry of a program request. The program request consists of a file name followed by parameters separated by blanks. A carriage return indicates the end of a request. Null input requests are ignored. When the program request is entered, the file, if found and of type MODU (module), is loaded and called with the entered parameters. When programs are called from the console no data is returned. When called from another program, data may or may not be returned (see description of SVCCA in Section 3).

Program and GMS synchronization is achieved by use of the WAIT and POST instructions provided by the LEVEL0 Extended Machine. GMS initiates the input command by issuing an SVC (SVCCA) and WAITS for notification of completion from the called program. The command mode of GMS is re-entered when the called program indicates completion by POSTing the WCH pointed to by register 2 on entry. The called program is deleted when it issues the final RETurn instruction. The POST must precede the RET. However, the called routine need not immediately issue a RET after POSTing GMS. Such a defered RET would allow the called program to remain in core during the execution of other console-initiated programs.

2.3.1 COMMAND INPUT FROM THE CONSOLE

When entering input lines from the typewriter, the console edit characters (RUBOUT, LINEFEED) can be used to backspace one character or delete the whole request line respectively. The BREAK character can be used to delete and restore console output. All console output will be suppressed after the use of BREAK until the next TYPEIN SVC is processed, or BREAK is depressed again.

2.3.2 COMMAND INPUT FROM THE VECTOR GENERAL

When entering input from the Vector General, the cursor forward and cursor backspace provide character editting facilities, and the home key acts as a line delete. Function keys 0, 1, 8, 9, 10, and 11 are used to control the operation of the Vector General for input:

0 is used to erase the screen.

1 is used as a toggle. When it is lit, lines will be rolled off the screen when more room is needed; when it is not lit, the user must press key 0 to continue when the screen gets full.

8 through 11 set the character size to the smallest through the largest, respectively.

3 GMS SVC'S

GMS provides a number of services to the user. These services are requested by the user program by means of supervisor calls. A single Meta 4A instruction, SuperVisor Call Single-register (SVCS), performs all the setup required to issue an SVC. The instruction loads a pointer to the SVC parameter list into register 2 and issues the SVC type indicated by the second argument:

```
          SVCS     PLIST,SVCNAME
```

where PLIST is the label of the SVC parameter list and SVCNAME is equated to the SVC number. The macro GMSVCS expands into a set of equates for the various SVC numbers supported by GMS.

The parameter list for the SVC provides a communication area for passing parameters to the SVC and storing the return code from the SVC and asynchronous control mechanism. The first word of the parameter list is always a Wait Control Halfword (WCH). The WCH must be zeroed before issuing an SVC pointing at the parameter list containing the WCH. When the SVC completes, it posts the WCH and inserts the return code in the second byte of the WCH. In order to insure that the SVC has completed, the user must issue a WAIT instruction with the WCH as the operand. The user will not execute the next instruction until the SVC has completed and the return code inserted.

The calling sequences, parameter lists, function, and return codes of the GMS SVC's are described in the following sections. Field lengths are in bytes unless otherwise specified.

FINIS

Function:

To close a file which was manipulated using either RDBUF and/or WRBUF SVC's.

Format:

```
                SVCS PLIST,FINIS
                .
                .
                .
     PLIST      DC    H'0'           WCH
                DC    A(FNFT)        POINTER     TO     FILENAME
                                     FILETYPE
                .
                .
                .
     FNFT       DC    CL8'filename'
                DC    CL4'filetype'
```

Usage:

The subject file is removed from the list of active files. Any unfinished buffers are written to disk. The directory entry for the file is updated if it has been changed. A file with pending I/O requests cannot be the subject of a FINIS SVC.

Return Code:

Value	Meaning
0	Normal.
2	File was not active.
4	File was in use--no action taken.

PARSESVC

__Function__:

To parse a command line.

__Format__:

```
          SVCD INPUT,L'INPUT,PARSESVC
          BNS  NULLINPUT

            or

          SVCD INPUT,0,PARSESVC
          BNS  NULLINPUT
```

__Usage__:

The input line is parsed into eight byte fields. The address of a getmained area containing the parsed line is returned in R3; the user must explicitly free this area when no longer needed. If the length is specified as zero, PARSESVC will process up to the first carriage return (X'15') in the input line; otherwise it will process the length specified. If the input line was completely blank, no parsed line will be returned, and the condition code will be set to 'NO SUCCESS'. If successful, the returned line will be in the format:

```
    HEADER     DC    H'Length including header'
               DC    H'Number of eight byte fields'
    TEXT       DC    CL8'Command'
               DC    CL8'1st operand'
               •
               •
               •
               DC    8X'FF'
```

RDBUF

Function:

To read a record from the disk.

Format:

```
              SVCS PLIST,RDBUF
              .
              .
              .
    PLIST     DC   H'0'              WCH
              DC   A(FNFT)           POINTER    TO    FILENAME
                                     FILETYPE
              DC   H'record number'  DESIRED RECORD NUMBER ≥
                                     1
              DC   A(input buffer)   POINTER TO INPUT BUFFER
              DC   H'record-length'  LENGTH OF DATA FIELD TO
                                     BE READ
              DS   2H                RESERVED FOR SYSTEM USE
              .
              .
              .
    FNFT      DC   CL8'filename'     NAME OF FILE TO BE READ
              DC   CL4'filetype'     TYPE OF FILE TO BE READ
```

Usage:

The input buffer is filled with data from the named disk file starting at the indicated record and continuing until the length has been satisfied. Both partial and multiple records can be read in this manner. It should be noted, however, that only the starting record number is checked for validity. Therefore, users employing multiple record reads must insure that all records included in the read exist in the file. The STATE SVC can be used to determine the record count of a file on disk. The data read from records beyond the end of the file are indeterminate.

Return Code:

Value	Meaning
0	Normal return.
2	File not found.
4	Invalid record number. Either equals zero or exceeds records in file.

RDCARD - OBSOLETE

Function:

Provide card input facilities to requester.

Format:

```
               SVCS PLIST,RDCARD
               •
               •
               •
               WAIT PLIST
               •
               •
               •
     PLIST     DC    H'0'           WCH
               DC    A(BUFFER)      POINTER TO INPUT BUFFER
               DC    A(LENGTH)      LENGTH OF INPUT
               DS    H              RESERVED FOR SYSTEM USE
```

Usage:

The RDCARD SVC is used to read a card from the card reader. Requests are processed in the order in which they are received. If an input length of greater than 80 is specified, a read of length 80 is assumed.

Return Code:

Value	Meaning
0	Normal read completed.
2	End of file encountered.
4	Hardware read error.

SEGLOAD

Function:

Dynamic loading of programs.

Format:

```
               SVCS PLIST,SEGLOAD
               .
               .
               .
    PLIST      DC    V(PROGRAM NAME)  EXTERNAL NAME OF PROGRAM
                                         WITHIN A SEGMENT
```

Usage:

The SEGLOAD SVC is used to dynamically load the segment which contains the given program. The segment can be deleted with the SEGDELET SVC. The GMSLINK manual describes how to set up program segments.

Condition Code:

C2 is 1 load successful or program already in core.

C2 is 0 load failed; return code will be in R2.

Return Code:

Value	Meaning
2	Program not found.
4	Insufficient core to load.
6	Invalid format.
8	Loader table overflow; insufficient core.
10	Invalid parameter.

SEGDELET

Function:

Dynamic deleting of programs.

Format:

```
        SVCS PLIST,SEGDELET
        •
        •
        •
PLIST   DC   V(PROGRAM NAME)  EXTERNAL NAME OF PROGRAM
                              WITHIN A SEGMENT
```

Usage:

The SEGDELET SVC is used to dynamically delete the segment which contains the given program. The GMSLINK manual describes how to set up program segments.

Condition Code:

C2 is 1 delete successful or program not in core.
C2 is 0 delete failed; a return code will be in R2.

Return Code:

Value	_Meaning_
10	Invalid parameter.

STATE

Function:

Provide information about a disk file to the requester.

Format:

```
              SVCS PLIST,STATE
              .
              .
              .
    PLIST     DC   H'0'           WCH
              DC   A(FNFT)        POINTER TO FILE
              DC   A(WORKAREA)    POINTER TO 26-BYTE WORK
                                  AREA
              .
              .
              .
    FNFT      DC   CL8'filename',CL4'filetype'
              .
              .
              .
    WORKAREA  DS   0H
    AFTFN     DS   CL8            FILE NAME
    AFTFT     DS   CL4            FILE TYPE
    AFTSECAD  DS   H              DISK SECTOR ADDRESS OF FILE
    AFTRECLN  DS   H              FILE RECORD LENGTH
    AFTRECCT  DS   H              FILE RECORD COUNT
    AFTSECCT  DS   H              FILE SECTOR COUNT
    AFTDATE   DS   CL6            DATE LAST CHANGED
```

Usage:

The STATE SVC returns to the requestor 26 bytes of information about the indicated file. This information represents the present status of the file on disk. For files which are open, this information may not reflect the present RECORD and SECTOR counts and date last changed. To insure that the information is correct the user must first FINIS an open file before issuing the STATE SVC. If a WORKAREA pointer of ZERO is used, no data is returned.

Return Code:

Value	Meaning
0	File found.
2	File not found.

SVCCA

Function:

To allow the user to invoke GMS utility programs or other program files directly from a program

Format:

```
               SVCS PLIST,SVCCA
               •
               •
               •
     PLIST     DC    H'0'             WCH
               DC    A(PROGRAM)       POINTER    TO   PROGRAM   CALL
                                      AREA
               DC    A(RETURN)        POINTER    TO    RETURN   DATA
                                      STORAGE AREA OR ZERO
               •
               •
               •
     PROGRAM   DC    CL8'program file name'  PROGRAM FILE NAME
                                      OR GMS UTILITY NAME
               DC    CL8'argument1'
               DC    CL8'argument2'
               •
               •
               •
               DC    CL8'argumentn'
               DC    8X'FF'           FENCE    INDICATING   END   OF
                                      PARAMETERS
```

Usage:

The SVCCA SVC is used to invoke external programs which reside on disk as separate files. GMS conventions exist for calling GMS utilities and receiving data returned from them. Calling another program (including GMS utility programs) using a return pointer of zero causes the program to operate as if it had been initiated via the console with the same arguments. Should a non-zero return area pointer be supplied, data is returned in the return area. In the case of GMS utility programs the data returnd includes data that would otherwise have been printed on the console. The user should consult the subject program source code to determine the order and format of the data returned.

The format example demonstrates the format and manner in which parameters are passed to programs invoked from the console. When GMS utility programs are called frcm a user's program, the same parameter passing conventions are required. If SVCCA was unable to load the routine in question, the WCH will be posted with the return code from LOADMOD plus X'100'.

TIMEWAIT

__Function__:

Provide time interval wait facilities fcr the user.

__Format__:

```
                SVCS PLIST,TIMEWAIT
                .
                .
                .
    PLIST       DC    H'0'              WCH
                DC    H'Time Interval in 1/100 Seconds'
                DS    4H                RESERVED FOR SYSTEM USE
```

__Usage__:

The TIMEWAIT SVC is used to set a time interval. The WCH will be posted with a code of 0 when the interval is expired.

TCANCEL

Function:

Provide time interval control facilities for the user.

Format:

```
              SVCS PLIST,TCANCEL
              •
              •
              •
    PLIST     DC   H'0'           WCH
              DC   H'Interval'
              DS   H              TIME  REMAINING IN INTERVAL
                                  WHEN CANCELED
              DS   3H             RESERVED FOR SYSTEM USE
```

Usage:

The TCANCEL supervisor call is used tc cancel an active time interval. PLIST must be te parameter list passed to the TIMEWAIT supervisor. The third word of the parameter list will be set to the time remaining in the interval when it was canceled, and the WCH posted with a completion code of 2. If the interval was not active, the time will not be returned, and the WCH will not be posted.

TYPEIN

Function:

Provide console input facilities to the user.

Format:

```
                SVCS PLIST,TYPEIN
                .
                .
                .
                WAIT PLIST
                .
                .
                .
      PLIST     DC    H'0'            WCH
                DC    A(BUFFER)       POINTER TO INPUT BUFFER
                DC    H'LENGTH'       MAXIMUM LENGTH OF INPUT
                DC    H'count'        NUMBER  OF  CHARACTERS READ
                                      EXCLUDING RETURN
                DS    CL1             START OF LINE CHARCTER
                DC    CL1'E or N'     E FOR EDIT, N FOR NO EDIT
                DS    2H              RESERVED FOR SYSTEM USE
```

Usage:

The TYPEIN SVC is used to read one logical record from the console. This record begins with the first character the user enters after the start of line character has been printed and ends with the return character. The start of line character is used to identify the task requesting input. The GMS command dispatcher uses the * character to indicate that a command input is requested. Logical backspace and line delete characters indicate editing functions in the EDIT mode and are treated as regular input charcters in the NO-EDIT mode. TYPEIN and TYPEOUT SVC's are performed in the order received. The execution of a particular TYPEIN request immediately following a TYPEOUT request can be achieved by using the REPLY option of the TYPEOUT SVC.

TYPEOUT

Function:

Provide console output facilities to the user.

Format:

```
                 SVCS PLIST,TYPEOUT
                 •
                 •
                 •
                 WAIT PLIST
                 •
                 •
                 •
     PLIST       DC    H'0'            WCH
                 DC    A(BUFFER)       POINTER TO OUTPUT BUFFER
                 DC    H'LENGTH'       LENGTH OF OUTPUT BUFFER
                 DC    CL1'E or N'     E FOR EDIT, N FOR NO EDIT
                 DC    CL1'R or N'     R FOR REPLY, N FOR NO REPLY
                 DC    A(TYPEIN)       POINTER  TO  TYPEIN REQUEST
                                       IF REPLY, UNUSED OTHERWISE
                 DS    2H              RESERVED FOR SYSTEM USE
```

Usage:

The TYPEOUT SVC is used to write one logical record to the console. The record contained in the output buffer of the indicated length is output to the console. In the EDIT mode trailing blanks are omitted and a carriage return automatically issued. In the No-Edit mode the buffer is output to the console as is. A single output request is issued in the NO-REPLY mode. To cause a given TYPEIN SVC to follow immediately a particular TYPEOUT SVC, the REPLY option of the TYPEOUT SVC is used. When specified, the TYPEIN pointer points to a TYPEIN PLIST. The input requested by the TYPEIN PLIST immediately follows the output requested by the TYPEOUT PLIST.

WRBUF

Function:

To write a record to a file on disk.

Format:

```
                SVCS PLIST,WRBUF
                .
                .
                .
     PLIST      DC    H'0'             WCH
                DC    A(FNFT)          POINTER     TO      FILENAME
                                       FILETYPE
                DC    H'record number'  RECORD NUMBER TO BE
                                       WRITTEN ≥1
                DC    A(output buffer)  POINTER TO OUTPUT
                                       RECORD
                DS    3H               RESERVED FOR SYSTEM USE
                .
                .
                .
     FNFT       DC    CL8'filename'    NAME OF FILE
                DC    CL4'filetype'    TYPE OF FILE
                DC    H'record length'  RECORD LENGTH USED ONLY
                                       WHEN   CREATING   FILE   BY
                                       WRITING   1ST   RECORD  1ST
                                       TIME.
```

Usage:

The indicated record of the named file is written out. If the indicated record exists, it is replaced by the new copy. If the record number does not exist, and it exceeds by 1 the present record count, the file is extended by this one record. When the file is created by the first WRBUF SVC, the record number specified must be 1. During the creation of the file, the record length field must be valid and exceed 15 bytes in length. The record length specified is used as the record length for all subsequent WRBUF's to this file.

Return Code:

Value	Meaning
0	Normal return.
2	File not found.
4	Invalid record number - either 0 or too large.
6	Insufficient contiguous disk space - see PACK utility.

4 GMS UTILITIES

GMS provides a number of utility programs. These utility programs, invokable from the operator's console or directly from a user's program, provide commonly used functions including file creation, manipulation, status monitoring, and dumping as well as system housekeeping and monitoring. These utilities can be invoked directly from the console by typing in the utility name followed by the required parameters separated by one or more blanks. These utilities also can be invoked directly from a user's program via the SVCCA SVC. The calling conventions for use of SVCCA are described in Section 3. The operation, parameters, and error messages of the GMS utilities are described in the following sections.

The general form of the console input for GMS utilities is as follows:

```
┌----------------------------┐
| OPERATION  <OPERANDS>  |
└----------------------------┘
```

The symbols used to represent console inputs in this document are described below:

UPPERCASE	Information given in capitals must be typed as shown although it may be entered in either upper or lower case.
lower case	Lowercase information designates the contents of a field and does not in itself constitute meaningful input.
( )	Parentheses must be typed as shown when any of the information appearing within them is specified.
< >	Brackets indicate information which may be omitted.
<< >>	Nested brackets indicate items which, if specified, must appear in the order shown.

...	Ellipsis indicates that the preceeding item may be repeated more than once.
__________	Underlining indicates the value which is assumed if none is specified.

ALTER

Function:

The ALTER utility is used to modify the filename and/or filetype of an indicated file.

Format:

```
┌──────────────────────────────────────────────┐
| ALTER    oldfn  oldft  newfn  <newft>        |
└──────────────────────────────────────────────┘
```

oldfn	present name of file
oldft	present file type
newfn	name to which file is to be changed
newft	type to which file is to be changed. If not specified, the old filetype is assumed.

Usage:

The name and or type of a file may be changed. This utility is especially helpful in maintaining programs by allowing new versions of programs to be manipulated and tested while old versions exist under another name.

Error Messages:

FILE NOT FOUND. The specified file was not found in the directory.

NEW NAME ALREADY EXISTS. A file by the new name already exists. The utility will not run until the desired new name does not exist in the directory.

INVALID ARGUMENT(S). Improper input.

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	File nct found.
6	New name already exists.

COMBINE

Function:

The COMBINE utility allows the user to combine 'n' old files into one new one.

Format:

```
┌──────────────────────────────────────────────────────┐
| COMBINE  newfn  newft  <oldfn  oldft> ...            |
└──────────────────────────────────────────────────────┘
```

newfn	name to be given to file created by the COMBINE directive
newft	type of file created by the COMBINE directive
oldfn	each file specified by the filename filetype pair must have equal record lengths.

Usage:

The COMBINE directive creates a new file by concatenating the old files specified in the order specified. The new file must not already exist as a disk file.

Error Messages:

INVALID ARGUMENTS(S). Improper number of arguments specified.

NEW FILE ALREADY EXISTS. The new file specified already exists. No action taken.

RECORD LENGTH INCOMPATIBLE. The old files have incompatible record lengths.

NOT FOUND. Old file not found. New file closed containing old files in list to this point.

DISK FULL - PACK! Insufficient contiguous disk storage. See PACK directive.

DELETE

Function:

The DELETE utility allows the user to remove from core a program previously loaded with LOADMOD or SEGLOAD.

Format:

```
┌─────────────────────┐
| DELETE  filename |
└─────────────────────┘
```

filename	name of program to be deleted.

Usage:

The program and all segments associated with it (see the GMSLINK manual for a description of segments) will be deleted from the load list and its program memory freed if its load count goes to zero.

ERASE

Function:

The ERASE utility is used to delete a given file from the disk.

Format:

```
┌──────────────────────────────────┐
| ERASE  filename  filetype        |
└──────────────────────────────────┘
```

filename	name of file to be deleted.
filetype	type of file to be deleted.

Usage:

The named file is deleted from the disk.

Error Messages:

FILE IN USE. The indicated file is in use presently and therefore cannot be deleted.

FILE NOT FOUND. The indicated file was not found or was found to be the object of an ERASE already in progress.

Return Code:

Value	Meaning
0	Normal
2	File in use.
4	File nct found.

EXEC

Function:

The EXEC command is used to stack a command list from disk for execution by GMS.

Format:

```
┌────────────────────────────────────────────────────────────┐
| EXEC filename <operand ...>                                |
└────────────────────────────────────────────────────────────┘
```

filename	name of EXEC file (filetype EXEC)
operand ...	symbolic operands

Usage:

The EXEC file is read into core, with symbolic substitutions made for operands. The substituted command list is then queued for execution by GMS, with prompting done for prompt operands at execution time.

Error Messages:

filename EXEC NOT FOUND. The specified EXEC file was not found in the directory.

NO NAME SPECIFIED. There was no EXEC file specified.

filename EXEC I/O ERROR. There was an I/O error reading the EXEC file.

filename EXEC INVALID SYMBOL CRD no. An invalid symbol was encountered in card "no." of the specified EXEC file.

filename EXEC INVALID IF STATE CRD no. An invalid if statement was encountered in the specified card of the specified file.

execname EXEC INVALID OPERATOR CRD no. An invalid logical operator was encountered in an if statement.

execname EXEC INVALID OPERAND CRD   no. The value to check against the return code was not rumeric.

Return Code:

Value	Meaning
0	Normal
12	Abnormal

Command Statement Syntax:

```
Terminals:
  <number> =: Number between 1 and 12
  <text7> =: 1 to 7 characters
  <text8> =: 1 to 8 characters
  <label> =: -<text7>
  <prompt> =: ?<text7> | <text7>?<text7>
  <operator> =: EQ | NE | GT | LT | GE | LE
  <symbolic> =: &<number> | <text7>&<number>
  <op1> =: RC | <symbolic> | <text8>
  <op2> =: <symbolic> | <text8>
  <freeop> =: <symbolic> | <text8> | <prompt>

Statements:
  <ifstate> =: &IF <op1> <operator> <op2> <label>
  <goto> =: &GOTO <label>
  <print> =: &PRINT <op2> | <print> <op2>
  <typeout> =: &TYPEOUT OFF
  <command> =: <command> <freeop>
  <rcommand> =: <ifstate> | <goto> | <print> | <typeout>
  | <command>
  <lcommand> =: <rcommand> | <label> <fcommand>
```

FUDD

<u>Function</u>:

The FUDD command provides the user with an online debugging capability.

<u>Format</u>:

```
┌───────┐
| FUDD  |
└───────┘
```

<u>Usage</u>:

The debugger will respond with the message 'FUDD HERE ...' and request a command. The FUDD subsystem is described in detail in the <u>FUDD User's Guide</u>.

HERCULES

Function:

The HERCULES utility allows a user to validate and correct the disk directory. It should only be invoked from the console after IPL'ing to avoid conflict with presently open files.

Format:

```
┌──────────────┐
| HERCULES     |
└──────────────┘
```

Usage:

The HERCULES utility reads the entire disk directory into core. Any file with leading or inbedded blanks or unprintable characters in its name is deleted. The directory is then scanned for invalid or conflicting entries. Invalid files are deleted and a message printed to that effect. If the allocation for two files conflicts, the user will be requested to decide which file should be deleted. The updated directory is then written back to the disk.

Informational Messages:

PHASE x COMPLETE: i PROCESSED, j KEPT, k DELETED. One or more files have been deleted by HERCULES.

filename filetype DELETED. This message tells the user which file has been deleted.

Action Messages:

```
FILES CONFLICT:
  FILENAME TYPE START LENGTH  END
1-filename type  i      j      k
2-filename type  i'     j'     k'
```
DELETE 1 OR 2? The user must enter '1' or '2' on the keyboard and the indicated file will be deleted.

LISTF

Function:

The LISTF utility allows the user to list on the console information about the files on the disk.

Format:

```
┌─────────────────────────────────────────────────────┐
| LISTF  <filename  filetype   <(keywords)>>  |
|        <file*     file*      <(keywords)>>  |
|        <   *          *      <(keywords)>>  |
|            *          *                     |
└─────────────────────────────────────────────────────┘
```

keywords

NAME -- print only file name
EXEC -- create an exec file with filename GMS and filetype EXEC
EXEC=fn -- create exec file with filename fn and filetype EXEC
INFO --print size of file and date last modified
HEAD -- print file header
SORT=sortid -- sort files in ascending order as specified, sortid is any combination of the following:

N -- sort by filename and type
T -- sort by filetype
S -- sort by filesize
P -- sort by position on disk.

Usage:

When a LISTF utility request is entered, the filename and filetype of all files meeting the request are printed. The specification of the filename and type can take three alternate forms. The first is the complete specification of the indicated field. The second is the partial specification of the field immediately followed by a '*' character. The field is matched by all disk directory entries which begin with the given partial specification. The third alternate form is the '*' character along which is matched by any directory entry. A null LISTF request causes a list of all files to be printed.

Error Messages:

FILE NOT FOUND. The utility request file specification was not matched by any file in the disk directory.

INVALID ARGUMENT. An incorrect option was specified.

Return Code:

Value	Meaning
0	Normal
2	Invalid argument.
4	File not found.

LOADMOD

Function:

The LOADMOD utility allows a user to load a program from disk into core.

Format:

```
┌──────────────────┐
| LOADMOD filename |
└──────────────────┘
```

filename	name of MODU file containing the program to be loaded.

Usage:

The root segment of specified program will be loaded into core. That copy will remain until a DELETE command is issued specifying the same name. Other segments may be loaded by the SEGLOAD SVC or they may be loaded automatically when a routine in a segment not currently in core is called. (See the GMSLINK manual for a description of segments.)

Return Code:

Value	Meaning
0	Normal
2	MODU file not found.
4	Insufficient core to load.
6	Invalid MODU format.
8	Loader table overflow; insufficient core.

MODMAP

Function:

The MODMAP utility allows a user to print a map of a program.

Format:

```
MODMAP  filename  <start>
```

filename	filename of the program to be mapped.
start	address to be used as the initial load point.

Usage:

The MODMAP prints out the names of all the external symbols and their address relative to the start point. Non-CSECT entries will be indented one space.

Error Messages:

FILE NOT FOUND. The MODU file was not found in the disk directory.

Return Code:

Value	Meaning
0	Normal
2	File not found.

MODZAP

Function:

The MODZAP utility allows a user to permanently change the contents of a program file on disk.

Format:

```
MODZAP  filename  <csect>
```

followed by any number of lines of

```
address  old-contents  new-contents
```

or

```
address  P  <hex count>
                 2
```

filename	name of the MODU file to be modified.
csect	CSECT name, if multiple CSECT assembly. If not specified, filename is assumed to be CSECT name.
address	offset from the start of the CSECT to be modified.
old-contents	hex string used to verify old contents of file at location to be modified. If the verification fails, the update is not performed. This field must have an even number of characters. Odd length fields are zero padded right by one.
new-contents	hex string used to replace the old contents of the file at the indicated address. This field must have an even number of characters. Odd

length fields are zero padded right by one. The replacement and verification fields must be equal in length and not greater than 32 characters long.

P	indicates printout of hex contents at the indicated address is desired.
hex count	count of hex digits to be printed. Two is assumed if not specified. A value of 0 causes a nop. A value exceeding 64 is invalid.

Usage:

The file and CSECT thereof to be modified are specified in the MODZAP utility request. MODZAP processes all subsequent console requests directly. Two MODZAP requests are available: one to display the hex contents at a given location and one to verify and replace the contents of a given location. Any number of requests will be honored once the original MODZAP utility is entered. The utility is exited when a null line is input.

Error Messages:

FILE NOT FOUND. The filename in the MODZAP utility directive does not exist on disk.

CSECT NOT FOUND. The csect field specifies a csect name not found in the indicated file.

INCOMPLETE. A modify or print request was not completed due to non-existant text.

INVALID ARGUMENT(S). Either improper specification of one of the modify or print fields or a verification failure.

ADDRESS NOT IN CSECT. Address given was not in current csect.

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	File not found.
6	CSECT not found.

OFFLINE LIST

Function:

The OFFLINE LIST utility allows the user to list on the console fixed columns of cards read from the card reader.

Format:

```
┌──────────────────────────────┐
| O LIST  <start  <end>>       |
|            1     80          |
└──────────────────────────────┘
```

start	column starting the field to be printed on the console. Column 1 is assumed if not otherwise specifed.
end	column ending the field to be printed on the console. Column 80 is assumed if not otherwise specified.

Usage:

The card reader is prepared by pushing the start and read buttons. The utility is then initiated from the console. Cards will be read and listed until either the reader is stopped manually, or the hopper is empty.

Error Messages:

INVALID ARGUMENT(S). Invalid specification of utility name or column parameters.

CARD READ ERROR. Hardware indication of card reader error.

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	Card read error.

OFFLINE READ - OBSOLETE

Function:

The OFFLINE Read utility provides for the creation of a disk file of card images from the card reader.

Format:

```
| O READ  filename  filetype
| O READ  *
```

filename	filename of the disk file to be created.
filetype	file type of the created file.

Usage:

The utility is initiated from the console. The card reader is then started and will continue to read until stopped or until the hopper is empty. The file specified is created on disk. If a file by the same name already exists, it is deleted and the new file created. The record length for the file is determined by the filetype. Standard GMS filetypes assume their default lengths. Non-standard filetypes cause files with record lengths of 80 to be created.If '*' is specified, OFFLINE expects the cards in the stacker to have OFFLINE READ commands on the front of each file to be read. The format of these cards must be:

```
OFFLINE READ filename filetype
```

with exactly one space between each parameter. Each OFFLINE READ card will be printed on the console as it is read.

Error Messages:

DISK FULL - PACK! Insufficient disk space is available. Use PACK utility to recover unavailable space.

INVALID ARGUMENT(S). Utility name, filename or filetype in error.

CARD READ ERROR. Card reader error occured. File may exist partially completed.

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	Card read error.
6	Disk full.

PACK

Function:

The PACK utility moves disk files to available sectors closer to the lower sector number portion of the disk. The result is an unfragmented disk with all the free sectors available in one contiguous area at the end of the disk.

Format:

```
┌──────────┐
| P_A_C_K  |
└──────────┘
```

Usage:

This utility should only be invoked from the console after IPL'ing to avoid conflict with presently open files.

Error Messages:

SPURIOUS DIRECTORY ENTRIES:
FILENAME TYPE ADDRESS.

Entries listed exhibit inconsistencies between disk file directory and disk allocation record; run HERCULES.

Return Code:

Value	Meaning
0	Normal
2	One or more spurious directory entries encountered.

PRINTF

Function:

The PRINTF utility dumps a file cr a portion thereof on the console.

Format:

```
┌──────────────────────────────────────────────────┐
| PRINTF  filename  filetype  <start  <end>>       |
|                              1       last        |
└──────────────────────────────────────────────────┘
```

filename	name of file to be dumped.
filetype	type of file to be dumped.
start	starting record number of dump. If not specified, 1 is assumed.
end	ending record number of dump. If not specified the last record is assumed. If specified, the ending record number must not be less than starting record number.

Usage:

The PRINTF utility will dump consecutive records starting from the indicated record until the ending record is specified or an end of file is encountered.

Error Messages:

FILE NOT FOUND. The indicated file does not exist.

INVALID ARGUMENT(S). One of the optional paramenters is in error or one of the required parameters is not supplied.

FILE HAS N RECORDS. The start record number exceeds the number of records in the file (N).

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	File nct found.

PRINTFX

Function:

The PRINTFX utility dumps a portion or all of a given file in hexadecimal and the equivalent EBCDIC.

Format:

```
┌──────────────────────────────────────────────────┐
| PRINTFX  filename  filetype  <start  <end>>      |
|                                1      last       |
└──────────────────────────────────────────────────┘
```

filename	name of the file to be dumped.
filetype	type of the file to be dumped.
start	beginning record number to be dumped. If not supplied, 1 is assumed.
end	ending record number to be dumped. If not supplied, the last record in the file is assumed. The end record number may not be less than the start record number.

Usage:

The PRINTFX utility will dump records from a file as directed until all the indicated records have been dumped or until an end of file is encountered.

Error Messages:

FILE NOT FOUND. The named file does not exist on disk.

INVALID ARGUMENT(S). Either one of the first two parameters was omitted or one of the optional parameters was incorrectly specified.

FILE HAS N RECORDS. The starting record number exceeds the number of records in the file (N).

Return Code:

Value	Meaning
0	Normal
2	Invalid argument(s).
4	File nct found.

QUERY

Function:

The QUERY utility allows the user to obtain the date, the number or names of parallel tasks executing, modules loaded, files active, or the free memory available.

Format:

```
┌────────────────────────────────────┐
|          DATE                      |
|          FILES                     |
| QUERY    PARALLEL  <mode>          |
|          MEMORY                    |
|          MODULES                   |
└────────────────────────────────────┘
```

DATE	indicates the date is desired
FILES	indicates that the number of open files is desired..
PARALLEL	indicates that the number of parallel tasks executing is desired.
MEMORY	indicates that the free memory space available is desired.
MODULES	indicates that the number of modules currently loaded is desired.
mode	if specified, indicates that the names rather than the numbers are required for FILES, PARALLEL, and MODULES.

Usage:

The response to the request is output on the console.

Error Messages:

INVALID ARGUMENT. The subfunction was specified incorrectly.

Return Code:

Value	Meaning
0	Normal.
2	Invalid argument.

SET

Function:

The SET utility allows the user to change the date, logical backspace and logical delete characters.

Format:

```
┌──────────────────────────┐
| SET DATE  mm/dd/yy       |
└──────────────────────────┘

┌───────────────────────────┐
| SET BACKSPACE  char       |
└───────────────────────────┘

┌────────────────────────┐
| SET DELETE  char       |
└────────────────────────┘
```

mm/dd/yy	standard representation of date.
char	a single console character to be used as specified.

Error Messages:

INVALID ARGUMENT(S). Error in subfunction specification, date specification or logical character not specified.

Return Code:

Value	Meaning
0	Normal.
2	Invalid argument.

SLOAD

Function:

The SLOAD utility is used to load SIMALE text pages from a disk file into main memory, build a page table for the SIMALE, and copy the page table and one selected text record into M4B local storage.

Format:

```
┌─────────────────────────────┐
| SLOAD filename label        |
|                pagenumber   |
└─────────────────────────────┘
```

filename	name of the SIMO file to be loaded.
label	label or entry point name in text record to be copied to M4B local storage.
pagenumber	decimal number between 0 and 31 inclusive indicating page number to be copied to M4B local storage.

Usage:

When an SLOAD utility is entered, the SIMALE text pages in the filename with filetype of SIMO are loaded into main memory allocated in one page blocks (256 bytes). Once all text pages are loaded, the selection field is used to select the text page to be copied to M4B local storage (128 halfwords starting at local storage address 128). If a label is specified, the symbol table is accessed to find the corresponding page number. If a decimal number is specified, it is used as the page number. Note that by GMSconvention the label field can be at most 8 characters long.

Error Messages:

Improper parameter specification. Too few parameters were specified.

filename SIMO NOT FOUND. Specified file was not found.

filename SIMO NOT WELL FORMED. Specified file is internally inconsistant.

filename SIMO - INSUFFICIENT FREE STORAGE. There were no 256 byte blocks left in main memory.

INVALID LOCAL STORAGE PAGE SELECTION. If a label was specified, it was not found in the symbol table or it was neither a label nor an entry point. If a page number was specified, it was either out of range (0 ≤ page number ≤ 31) or the indicated page did not have a corresponding text page in the file.

DISK I/O ERROR. Disk i/o error occurred during the reading of the specified file.

MULTIPAC NOT AVAILABLE. Multipac not available.

Return Code:

Value	Meaning
0	Normal
2	Normal
4	Improper parameter specification
6	File not found
8	Nsufficient free storage
10	Invalid local storage page selection
12	Disk i/o error
14	Multipac not available

STATISTICS

Function:

The STATISTICS utility provides the user with a report of the present status of the disk.

Format:

```
┌─────────┐
| STATS   |
└─────────┘
```

Usage:

When invoked, the statistics utility produces the following output:

```
i FILES OCCUPYING j SECTORS; k SECTORS FREE (l% FULL)
```