CGTM # 84

COMMAND STRUCTURE OF A SUB-MONITOR FOR THE SLAC GRAPHIC INTERPRETATION FACILITY

G. A. Robinson

November 1969

INTRODUCTION:

GSM stands for "Graphic Sub-Monitor". As the name suggests, it is a sub-monitor (or second-level operating system) that operates under OS/360 and is oriented toward graphic display devices. It is programmed in 360 assembly language and FORTRAN. From the outset, its philosophy has been to utilize only supported, documented features of OS, rather than attempting to capitalize on specific internal programming features of the implementation of a given release of OS.

The purpose of GSM is to provide a relatively JCL-free on-line environment in which a user can (in real time) edit, reassemble (or recompile), execute, and debug his program from a suitable page-oriented graphic terminal such as the IBM 2250, the IDIIOM, or the IBM 2260.

The version described herein is GSM-2. It was designed to operate under release 16 of OS/MVT on the IBM 360/91 at SLAC. It replaces GSM-1, the original system designed to operate under release 13 of OS/MFT. With the implementation of release 14, GSM-1 became inoperable due to a number of serious bugs in release 14 and unidentified, undocumented external performance changes in release 14, notably in the release 14 F-level assembler and the release 14 linkage editor.

GSM-2 can be used, but with considerably degraded efficiency, under HASP with OS release 16. Keyboard response, inter-job timing, and most input-output operations are considerably slower when running under HASP than when running directly under OS/MVT. At the time of this writing, it has not yet been determined whether GSM

can be supported under OS release 17 with HASP. That will depend on the severity of the new system program bugs introduced in release 17, on the attention given to correcting bugs and malfunctions in release 17 by SFSCC, and upon the resources available to those members of the SLAC Computation Group who have developed and maintained the 360 and 620-I programs embodied in GSM.

DATA STRUCTURE:

Three classes of data storage are provided under GSM-2:

A - A partitioned data set (DSN=LMODS), composed of load modules created and/or executed under GSM.

B - A number of temporary sequential data sets (DDNAMEs= FTnF001) holding principally card images or line images during or between process steps requested by the user.

C - One large direct data set (DSN=CG01DA) used by GSM to provide a number of virtual data sets to various users of GSM. These virtual data sets will be called "direct data sets" to distinguish them from sequential data sets in class B.

GSM COMMANDS:

Creating, editing, and copying ~~direct~~ data sets:

CREATE,n$ — Creates an (empty) direct data set beginning at disk block n. Leaves GSM in state ready to edit that set.

EDIT,n$ — Prepares to edit the existing data set beginning at block n.

ENDEDIT$ — Transfers all intermediate information from core to the data set currently being edited so that it will not be lost during subsequent operations. Leaves GSM still in a state ready to edit the set.

ACCEPT,n$
A,n$ — Assumes that a previous EDIT or CREATE has specified which direct set we are editing. This command tells GSM that the text about to be entered is to be put in logical sequence just before statement number n in the data set. To add to the end of a data set or to start from scratch, n=30000.

CHANGE,n$
C,n$ — Specifies that statement number n in the current data set is to be changed. The text of the existing statement will be displayed, ready for changing.

CARDS,n$ — Specifies that card images from sequential unit n are to be placed at the location specified by the last ACCEPT command.

INSERT,n1,n2,n3$ — Specifies that statements numbered n1 through n2 in the current direct set are to be placed in logical sequence just before the first occurrence of statement number n3 in the current direct set. This does not affect any other occurrences of n1...n2 in the set.

DELETE,n1,n2$
DELETE,n1$ — Specifies that all of the text in the current direct set logically between the first occurrence of n1 and the next subsequent occurrence of n2 inclusive is to be logically deleted. Other occurrences of n1, n2, etc. are unaffected. If n1=n2, only one statement will be deleted, and the specification of n2 may be omitted.

PROTECT,n,name,k$ — Assumes that a direct data set exists beginning at block n. k blocks of storage will be protected against clobbering by anything except an EDIT,n$. To remove the protection, one must know the name used. BLOCKS 6000-6999 AND 4000-4999 CANNOT BE SECURELY PROTECTED.

PURGE,n,name$ — Remove the protection from the data set beginning at block n if the name specified in the original PROTECT agrees with the name specified in this PURGE. Otherwise, punt.

STOD,u,b$ Creates a direct data set beginning at block b and copies into it the information (card images) on sequential unit u, beginning at the current position of unit u and continuing until an end-of-file or pseudo-delimiter (columns 1-4 = blank //*). The pseudo-delimiter is not copied.

DTOS,b,u$ Copies the card images comprising the direct data set beginning at block b onto sequential unit u beginning at the current position of unit u. No end-of-file is written and the sequential unit is not rewound before or after copying the information.

CONVSQ$ Rewinds sequential units 12,13,15, and 10. Then does the equivalent of DTOS,current data set,10. Then an end-of-file is written on 10 and 10 is rewound. This command is most frequently used to prepare the source input for the assembler or compiler.

W,u,message$ u must be a two-digit unit number. This command then writes a card image with message copied into it, beginning with column 1 and continuing until the end of message as indicated by the dollar sign.

NAMER,name$ A NAME control card for the linkage editor, indicating the (R) option for replace, is written on unit 20.

Displaying Data Sets:

DISPLAY,n1,n2$ D,n1,n2$ The first n2 lines of text of the current direct data set, beginning with statement number n1, are displayed. If n2 is omitted (as is usually the case) the default is the last value of n2 given for a DISPLAY command or the number assembled in as a default. If both n1 and n2 are omitted, n1 is taken to be the last statement number from the previous display.

VUELIST,n1,n2$ V,n1,n2$ The first n2 lines of text from sequential unit n1 are displayed. If n2 is omitted, n1 is taken from the last vueing command or from an assembled default value. If n1 and n2 are both omitted, the display begins with the first line after the last frame displayed.

PRINT,n$ Not really a display command, but it is in this context that it most frequently comes up. This command causes the entire contents of sequential unit n to be printed on SYSOUT.

Executing Load Modules:

G,name$ — Causes a LINK to the load module (from LMODS) that has name as an entry point. (Note the distinction between entry points for load modules and entry points for object programs. Most commonly, a load module you have just created will have only one entry point--the name that appeared in the NAME name(R) statement used to control the linkage editor. If you aren't clear on this point, look it up in the linkage editor manual or ask someone who knows.)

EXECUTE,name$ — Same as G, but the load module is obtained from JOBLIB.

Invoking System/360 processors:

ASM$ — Invokes the F-level assembler. Source input is from sequential unit 10, listings to to unit 12, object output goes to unit 20.

ASMG$ — Same as ASM, but the G-level assembler is invoked.

FTNG$ — Invokes the Fortran (G) compiler. Input from unit 10, listing tounit 15, object programs to unit 20.

PL1$ — Invokes the PL/I compiler. Input from unit 10, listings to unit 17, object programs to unit 20.

LINK$ — Invokes the linkage editor. Object programs are input from unit 20, linkage map to unit 13, load module written into the P.D.S. LMODS.

PLINK$ — Same as LINK, but for use with PL/I programs.

UTILITY,name$ — Invokes the OS/360 Utility program specified by name. Input is from unit 21 (unblocked). Output goes to unit 13 (unblocked).

Inspecting core memory:

DUMP,h$ — Displays a hex and EBCDIC dump of a portion of core beginning at hex address h.

PDUMP,h$ — Same as DUMP, but displays the contents of core at the time of the return from the last object program executed by an EXECUTE or G command. Assumes that DUMPON was issued prior to the EXECUTE or G.

DUMPON$ — Causes the state of core to be saved upon each return from a G or EXECUTE command.

DUMPOFF$ — Nullifies DUMPON$

DPC$ — Continues displaying the contents of core, beginning just after the end of the previous display. Used following DUMP, PDUMP, or DPC.

Miscellaneous Input, Output, and Control commands:

REW,n$	Rewinds sequential unit n.
WEOF,n$	Writes an end-of-file on sequential unit n.
GETOFF$	Causes a normal termination of the GSM task, returning control to the GSM supervisor.
ABEND$	Same as GETOFF, but the termination is abnormal.
WTO,message$	Displays message on the 91 operator's console.
SUBMIT,n$	Assumes that the direct data set at block n contains the card images for a job to be submitted to the 360. Submits that job.
PCON$	Causes the commands processed by CMNDA to be printed on the standard output.
PCOFF$	Nullifies PCON$
CMNDAC,name$	Causes an alternative version of CMNDA to be selected from LMODS and used to process subsequent commands. This change remains in effect until the job terminates or control is returned to the GSM supervisor, or another CMNDAC is issued.
DUMGME,n$	Causes CMNDA to take its commands from unit n until it encounters an ONLINE command on that unit.
ONLINE$	See DUMGME.

Commands used primarily in writing and invoking canned procedures:

INVOKE,n,pl,...,pn$ I,n,pl,...,pn$	Invoke the canned procedure at block n with actual parameters pl,...pn.
RETURN$	Returns control from a procedure to the keyboard.
N$	No-operation. Used primarily to delete temporarily steps of a procedure when these steps are not desired at present.
WAIT,n$	Waits for n tenths of a second before returning control to the next step of the procedure or to the keyboard. Used primarily to keep the display from one procedure step visible on the scope long enough to be seen by the user.

EXAMPLES OF USE OF GSM COMMANDS:

The following examples are intended to illustrate the use of some of the GSM commands. They are by no means exhaustive.

To create a new 30 block (750 card images) virtual data set beginning with block 2100, protect it with the key SOMENAME, and enter date into it from the keyboard:

```
CREATE,2100$
PROTECT,2100,SOMENAME,30$
ACCEPT,30000$          Specifies that the following data is to be
                       placed into the data set logically just
                       before the end (30000) of the data set.

---text---

#                      Escapes from text mode.
ENDEDIT$               Completes the preservation of the data on
                       the disk.
```

To assemble a source program beginning at block 2600, print the assembly listings on SYSOUT, save the object output of the assembler in a new data set beginning at 2700, and linkedit it into the P.D.S. LMODS under the name XYZ:

```
EDIT,2600$             Select the source data set.
CONVSQ$                Prepare a sequential data set for input to
                       the assembler (unit 10).
ASM$                   Invoke the F-level assembler.
PRINT,12$              Print the assembly listings on SYSOUT.  (To
                       view the listings on-line, see the VUELIST command.)
REW,20$
STOD,20,2700$          Save the object program.
REW,20$                   Prepare
DTOS,2700,20$             the input
W,20, NAME XYZ(R)$        to the
WEOF,20$                  linkage
REW,20$                   editor.
LINK$                  Invoke the linkage editor.
```

To execute the load module produced above, one would type-in:

```
G,XYZ$
```

To add three lines of text at a point logically just before statement number 21 in the direct data set at block 2300ff:

```
EDIT,2300$             Select the data set to be edited.
DISPLAY,1$             Display a page of card images beginning at statemer
ACCEPT,21$             Specify that the text goes logically before st. 21.
---text---
#                      ENDEDIT$
```

CANNED PROCEDURES:

When using JCL under OS/360, one frequently employs catalogued procedures for sequences of commands that follow a commonly-used stereotype. A similar, although less elaborate, mechanizm is provided in GSM in the form of "canned procedures".

In GSM-2 a canned procedure is invoked by typing INVOKE,n$ (or, more briefly, I,n$) where n is the block location of the procedure. If the procedure was written using formal parameters, actual parameters may be used to replace the formal parameters by writing the actual parameters (preceded by a comma and separated by commas) between the block number and the dollar sign in the INVOKE command (or the I command). For example, INVOKE,2570,X,YZ,62$ would specify X, YZ, and 62 as the actual parameters to replace the formal parameters :1, :2, and :3 in the procedure at block 2570.

A procedure is written as a sequence of commands, each of which is contained in a card image of text. The logical sequence of the commands must be the same as the physical, that is, if lines are inserted into a procedure, the data set must be resequences before it can be used as a procedure. Formal parameters are composed of a colon and a decimal digit. The first is :1, the second :2, etc. By first, we mean the one that is to be replaced by the first actual parameter. The procedure should end with a RETURN$ command.

Restrictions on canned procedures in GSM-2:

1. No more than 25 commands are permitted in a single procedure.
2. Actual parameters are limited to 18 characters.
3. Formal parameters may not be concatenated with each other or with anything else. They must be complete arguments within commands.

The following is an example of a canned procedure that will if located at block n and invoked by INVOKE,n,nl,n2$ assemble the source program at block nl ff and save the resulting object code beginning at block n2 after printing the assembly listings on SYSOUT.

EDIT,:1$	Select the data set to be assembled (:1 is the first formal parameter).
CONVS9$	Prepare a sequential data set as required for input to the assembler (on unit 10)
ASM$	Invoke the F-level assembler .
PRINT,12$	Print the assembly listings on SYSOUT.
REW,20$ STOD,20,:2$	Save the object code produced by the assembler.
RETURN$	Return control to the keyboard.

The following is an example of a canned procedure that will if located at block n and invoked by INVOKE,n,nl,<u>name</u>$, where <u>name</u> is an eight-character alphameric name, cause the object program at block nl to be linkedited with other object programs from blocks 2500ff and 2650ff, producing a load module called <u>name</u>, and then execute that load module after printing the linkage map.

REW,20$	
DTOS,:1,20$ DTOS,2500,20$ DTOS,2650,20$	Retrieve the given object program and write it on unit 20 along with the object programs from blocks 2500ff and 2650ff.
NAMER,:2$	Write a NAME <u>name</u>(R) control card for the linkage editor on unit 20.
WEOF,20$	
REW,20$	
REW,13$	
LINK$	Invoke the linkage editor.
PRINT,13$	Print the linkage map.
G,:2$	Execute the load module just produced.
RETURN$	Return control to the keyboard.