PRELIMINARY GSM-3 COMMAND MANUAL

This is a working draft of sections 0 through 5 of a reference manual for the Graphic Sub-Monitor, version 3.5.

G. A. Robinson

June 2, 1971

This is an internal document of the Graphics Laboratory of the Computation Group at SLAC. The Computation Group is part of the Research Division of SLAC and is not affiliated with the SLAC Facility of the Stanford Computation Center.
<table>
<thead>
<tr>
<th>Section</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>6-2-71</td>
</tr>
<tr>
<td>0.1</td>
<td>6-2-71</td>
</tr>
<tr>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0.4.1</td>
<td>6-7-71</td>
</tr>
<tr>
<td>0.4.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>0.4.3</td>
<td>6-7-71</td>
</tr>
<tr>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>6-1-71</td>
</tr>
<tr>
<td>3.1</td>
<td>6-7-71</td>
</tr>
<tr>
<td>3.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>3.3</td>
<td>6-1-71</td>
</tr>
<tr>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>4-5-71</td>
</tr>
<tr>
<td>5.2.1</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.2.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.2.4</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.2.5</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.2.7</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.2.8</td>
<td>4-8-71</td>
</tr>
<tr>
<td>5.2.10</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.2.12</td>
<td>4-9-71</td>
</tr>
<tr>
<td>5.3.1</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.3.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.3.4</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.3.6</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.3.7</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.4.1</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.4.2</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.4.3</td>
<td></td>
</tr>
<tr>
<td>5.4.4</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.5.1</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.5.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.5.2b</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.5.3</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.5.4</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.5.5</td>
<td>3-15-71</td>
</tr>
<tr>
<td>5.5.6</td>
<td>6-7-71</td>
</tr>
<tr>
<td>5.5.7</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.6.1</td>
<td>3-17-71</td>
</tr>
<tr>
<td>5.6.2</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.6.4</td>
<td>6-1-71</td>
</tr>
<tr>
<td>5.6.5</td>
<td>6-2-71</td>
</tr>
<tr>
<td>5.6.7</td>
<td>6-1-71</td>
</tr>
</tbody>
</table>
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 was originally programmed in 360 assembly language and FORTRAN, and has now been reprogrammed entirely in assembly language for reasons of core-space economy and more efficient operation. Originally, its philosophy was to utilize only supported, documented features of OS, rather than attempting to capitalize on specific internal programming features of the implementation of a given version of OS. In later versions, however, some features have come to depend on internal properties of OS as described in various IBM Program Logic Manuals.

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 IDIOM, or the IBM 2260.
### Alphabetical Index to GSM Commands

**Numeric Entries Indicate Page Addresses in this Manual.**

Alphabetical entries indicate commands not of interest to the general user and not listed in this manual.

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.2.5</td>
</tr>
<tr>
<td>ABEND</td>
<td>5.6.2</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>5.2.6</td>
</tr>
<tr>
<td>ALTDD</td>
<td>40</td>
</tr>
<tr>
<td>ALTER</td>
<td>5.2.10</td>
</tr>
<tr>
<td>ALTERD</td>
<td>5.2.10</td>
</tr>
<tr>
<td>ALIST</td>
<td>5.4.4</td>
</tr>
<tr>
<td>AOBJ</td>
<td>5.4.4</td>
</tr>
<tr>
<td>AS</td>
<td>5.2.5</td>
</tr>
<tr>
<td>ASK</td>
<td>05</td>
</tr>
<tr>
<td>ASM</td>
<td>5.4.2</td>
</tr>
<tr>
<td>ASQ</td>
<td>5.4.4</td>
</tr>
<tr>
<td>AUX</td>
<td>TR</td>
</tr>
<tr>
<td>C</td>
<td>5.2.5</td>
</tr>
<tr>
<td>CA</td>
<td>5.6.7</td>
</tr>
<tr>
<td>CANCEL</td>
<td>5.6.2</td>
</tr>
<tr>
<td>CARDS</td>
<td>5.3.4</td>
</tr>
<tr>
<td>CARDSS</td>
<td>5.3.6</td>
</tr>
<tr>
<td>CASE</td>
<td>5.6.7</td>
</tr>
<tr>
<td>CC</td>
<td>5.6.2</td>
</tr>
<tr>
<td>CG</td>
<td>5.2.4</td>
</tr>
<tr>
<td>CGS</td>
<td>5.2.5</td>
</tr>
<tr>
<td>CHANGE</td>
<td>5.2.5</td>
</tr>
<tr>
<td>CHAP</td>
<td>TR</td>
</tr>
<tr>
<td>CHKKN</td>
<td>5.2.2</td>
</tr>
<tr>
<td>CHWORD</td>
<td>SM</td>
</tr>
<tr>
<td>CMNDAC</td>
<td>5.5.2</td>
</tr>
<tr>
<td>CONVSQ</td>
<td>5.3.4</td>
</tr>
<tr>
<td>COPY</td>
<td>5.2.7</td>
</tr>
<tr>
<td>COPYX</td>
<td>5.2.7</td>
</tr>
<tr>
<td>CORE</td>
<td>SM</td>
</tr>
<tr>
<td>CP</td>
<td>5.5.3</td>
</tr>
<tr>
<td>CREATE</td>
<td>5.2.2</td>
</tr>
<tr>
<td>CS</td>
<td>5.2.5</td>
</tr>
<tr>
<td>D</td>
<td>5.2.4</td>
</tr>
<tr>
<td>DA</td>
<td>5.6.5</td>
</tr>
<tr>
<td>DELETE</td>
<td>5.2.8</td>
</tr>
<tr>
<td>DIDN</td>
<td>NU</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>5.2.4</td>
</tr>
<tr>
<td>DLN</td>
<td>5.5.7</td>
</tr>
<tr>
<td>DPC</td>
<td>TU</td>
</tr>
<tr>
<td>DRC</td>
<td>5.5.5</td>
</tr>
<tr>
<td>DUMPOFF</td>
<td>OR</td>
</tr>
<tr>
<td>DUMPON</td>
<td>OR</td>
</tr>
<tr>
<td>DTH</td>
<td>5.5.7</td>
</tr>
<tr>
<td>DTOS</td>
<td>5.3.4</td>
</tr>
<tr>
<td>DUMGME</td>
<td>SP</td>
</tr>
<tr>
<td>DUMP</td>
<td>TU</td>
</tr>
</tbody>
</table>
ONLINE... SP
OPTIONS... NO
P.... T0
PCOFF.... 5.4.5
PCON.... 5.4.5
POS.... 5.4.2, 5.4.4
PHY.... 5.2.7
PLINK.... 5.4.2
PLIST.... 5.4.4
PLOAD.... 5.4.3
PL1.... 5.4.2
POBJ.... 5.4.4
POFF.... 5.5.7
PON.... 5.5.7
PRINT.... 5.3.2
PROCESS... NO
PROT.... TP
PROTECT.... 5.2.12
PSQU.... 5.4.4
PURG.... TP
PURGE.... 5.2.12
QQ.... 5.5.3
R.... (5.2.10)
RCD.... 5.5.6
RCH.... 5.5.6
REPLACE.... 5.2.10
REPLAC.... 5.2.10
REPSEQ.... N1
RETURN.... 5.5.4
REW.... 5.3.2
SAP.... 5.5.6
SETNAME.... 5.2.12
SO.... 5.3.6
STAP.... 5.5.6
START.... OD
STOD.... 5.3.6
STODS.... 5.3.6
STORE.... SM
SRC.... 5.5.5
SUBMIT.... 5.6.4
SUBN.... 5.6.4
T.... T0
UPPER.... 5.6.7
USER.... 5.6.2
USN.... 5.5.7
UTILITY.... 5.4.3
V.... 5.3.2
VUELIST.... 5.3.2
W.... 5.3.4
WAIT.... 5.5.5
W2OP.... 5.3.2
WTO.... 5.6.5
X.... 5.3.7
XSCAPE.... SP
Z.... 5.5.3
EQUIPMENT CONFIGURATION:

GSM is currently implemented on three page-oriented devices: the IBM 2250 model 3, IBM 2260, and the IDIOM. In addition, direct access storage is required on the 360 for the user's large direct data set discussed in Section 3.3 and for the libraries discussed in Section 3.2, as well as temporary direct access space for the sequential data sets described in Section 3.1. The interactive version of GSM currently runs in 150K of core. (The batch version requires 300K.)
ENTERING A GSM COMMAND OR TEXT LINE:

Two basic types of information are entered by the user from the terminal: GSM commands and text. When GSM is expecting a command, it is said to be in "command mode"; when it is expecting text, in "text mode". On the 2250 and 2260, the dollar sign was chosen to indicate the end of a GSM command and the @ sign to indicate the end of a line of text, the former being called the "command delimiter" and the latter the "text delimiter". Throughout this document, these symbols ($) and @) will be used to represent the command and text delimiters respectively, even though different markings appear on the face of the keys used for these purposes on the IDIOM keyboards.

Several different GSM commands place GSM in text mode, some such as ACCEPT leave that mode in effect indefinitely, other such as CHANGE leave it in effect only for the number of lines to be changed. In either case, text mode may be terminated by the "escape command" E$. The following chart indicates the key label information for each of the four types of keyboard:

<table>
<thead>
<tr>
<th>2250</th>
<th>2260</th>
<th>IDIOM KB</th>
<th>Teletype KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>text delimiter</td>
<td>@ plus END</td>
<td>@ plus ENTER</td>
<td>@ rubout</td>
</tr>
<tr>
<td>command delimiter</td>
<td>$ plus END</td>
<td>$ plus ENTER</td>
<td>;</td>
</tr>
<tr>
<td>escape (col 1-2)</td>
<td>FS plus END</td>
<td>FS plus ENTER</td>
<td>F;</td>
</tr>
</tbody>
</table>

For use in a GSM control procedure, the command delimiter may be entered as a dollar sign within a line of text.

As each line of text is entered, it is assigned a four digit decimal integer as a sequence number. This integer is inserted in columns 77-80 of the card image.

For the 2741, the text delimiter is the at sign plus CR, the command delimiter is a dollar sign plus CR, and the escape is number sign plus CR.
SEQUENTIAL DATA SETS IN GSM:

When an OS processor or a user program requires data in the form of a sequentially ordered data set, GSM can manipulate such data by means of sequential data sets numbered (in the current version) from 1 to 50 inclusive. Although other logical record lengths can, in principle be used, the most commonly used features of GSM are oriented toward the use of card images (80 bytes per logical record) and line images (120 bytes per logical record). The most commonly used sequential sets are:

<table>
<thead>
<tr>
<th>unit #</th>
<th>RECFTM</th>
<th>LRECL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>FRA</td>
<td>80</td>
<td>usually SYSIN</td>
</tr>
<tr>
<td>6</td>
<td>FBA</td>
<td>80</td>
<td>usually SYSOUT=A</td>
</tr>
<tr>
<td>7</td>
<td>FB</td>
<td>80</td>
<td>usually SYSOUT=B</td>
</tr>
<tr>
<td>10</td>
<td>FR</td>
<td>80</td>
<td>source input to OS processor</td>
</tr>
<tr>
<td>12</td>
<td>FBM</td>
<td>121</td>
<td>assembly/linkage editor listings</td>
</tr>
<tr>
<td>15</td>
<td>FBA</td>
<td>120</td>
<td>FORTRAN listings</td>
</tr>
<tr>
<td>17</td>
<td>VBA</td>
<td></td>
<td>PL/I listings</td>
</tr>
<tr>
<td>21</td>
<td>FB</td>
<td>80</td>
<td>OS processor object output, linkage editor object input</td>
</tr>
<tr>
<td>33-38</td>
<td>M</td>
<td></td>
<td>Wylbur edit format files</td>
</tr>
<tr>
<td>39</td>
<td>P</td>
<td>88</td>
<td>input to HASP reader</td>
</tr>
<tr>
<td>42-59</td>
<td>P</td>
<td></td>
<td>User-defined units</td>
</tr>
<tr>
<td>80</td>
<td>VBA</td>
<td></td>
<td>SYSUDUMP</td>
</tr>
</tbody>
</table>
PARTITIONED DATA SETS OF LOAD MODULES:

Load modules to be executed during a GSM session (including the load modules used to implement GSM itself) are contained in one of two "libraries", LINKLIB or JOBLIB. No JCL DD statements are necessary for LINKLIB, but JOBLIB DD statements are required. JOBLIB is usually a concatenation of several partitioned data sets, including the P.D.S. which contains the load modules implementing GSM. Other libraries containing the load modules which the individual user wishes to use may also be concatenated with JOBLIB by writing appropriate JCL. The SYSMOD DD statement (and possibly other DD statements -- see the PDS, <dname>* command) specifies a P.D.S. which is to receive load modules created by the linkage editor during the GSM session. That P.D.S. may also be concatenated with JOBLIB.

Load modules from LINKLIB and/or JOBLIB can be invoked explicitly by means of the GSM C or X commands or in certain special cases the GSM UTILITY command, or implicitly by commands such as ASMG, PL1, FTNG, LINK, etc.
DIRECT SETS:

Permanent and working storage of data (text, source programs, object programs, etc.) is in one large direct access data set managed by GSM. This direct data set consists of a large number of blocks of 25 card images (2000+ bytes), numbered consecutively beginning with block 0001. Consecutive blocks may be (logically) organized into virtual data sets called "direct sets" in GSM. Each (virtual) direct set consists of a "header" block followed by one or more "text" blocks. A (virtual) direct set is created by the CREATE,<blocknum> command, which initializes the header block. The order in which the card images of a direct set are stored on the disk in one or more consecutively numbered text blocks is called the "physical order" of the lines (card images). As new lines are added to the set, they are placed at the end of the last lines added previously, regardless of where the user intends them to go in his scheme of things within the set. Each line is assigned for a sequence number the next consecutive integer, and this sequence number is stored in columns 77-80 of the card image. Subsequent references to the line are made by referring to this sequence number.

It is by means of pointers in the header block that the "logical order" intended to the user is imposed on the lines of text. If the user indicates, for example, that he wishes to interleave some new lines of text between lines 0042 and 0043 of an existing direct set, and if that set consists, say of 462 lines, then the new lines are assigned sequence numbers 463, 464, ... and stored physically following line 462. They are treated, however, as if they fell logically between lines 42 and 43 for the purposes of just about all GSM commands. It is this "logical order" that the user is almost always interested in rather than actual physical order. For example, when a page of text is displayed, the lines appear on the screen in logical order, with their (not necessarily consecutive) sequence numbers appearing on the left of the screen.

In order to avoid accidental destruction of data, each direct set may be protected from the most common sources of disaster by means of an optional protection key system, in which each block of a (virtual) direct set points to its header block and the header block contains a password which must be known to overwrite the data. The only usual means of changing a direct set without using the password is by means of an EDIT command explicitly referring to the origin of the set. While this does not prevent against malicious destruction of the set it has proved nearly 100 per cent effective against accidental overwriting of a set.

In order to facilitate reference to direct sets, a number of the GSM commands have a symbolic address option that allows reference to the set by a symbolic name rather than by the numerical value of the header block address, although the numerical reference technique is also allowed. Symbolic direct set names must begin with a non-digit and be from one to eight characters long. It is recommended that sets containing object code be named by strings beginning with a slash, and that only object code be so named.
COMMAND SEQUENCING:

Much of the time, the user will be interacting with GSM on a command-by-command basis, typing in a command, observing it being obeyed, and then typing in another command. By the use of GSM monitor control procedures, however, it is possible to start a chain of prestored commands in action and yield control to GSM until all have been executed. See section 5.5 for details concerning monitor control procedures and section 7 for examples of how they may be used.

Still another possibility involves a batch-oriented version of GSM. Although GSM is primarily intended for interactive use, a batch-oriented version has been developed for use in 300K of core. In this version, commands are included as card images with the JCL, loaded into a large core buffer, and executed from the buffer one at a time as if they were being typed in from a terminal. This version has seen very little use and is not as well debugged as the regular interactive version.
NOTATION:

Among the conventions used in specifying the structure of the individual commands in this section are the following:

Capital letters and special characters stand for themselves and are to be written as shown. Exceptions to this rule are parentheses, brackets, braces, less than, greater than, and stroke.

Lower case words or phrases that are underlined or enclosed in <...> signs are to be replaced by appropriate expressions as indicated in the description accompanying the command.

Square brackets indicate an optional parameter or sequence of parameters. If two or more alternatives are stacked vertically, one may be chosen from the alternatives or the parameter may be omitted.

Options stacked vertically within braces indicate a mandatory choice between the indicated alternatives.

An alternative notation to square brackets and to braces will frequently be used. (] will be used for left square bracket and ]) for right square bracket. Where no confusion will result, ordinary parentheses may be used for braces in this description. As an alternative to stacking options vertically, they will sometimes be written horizontally and separated by the stroke symbol |.

When no confusion with actual text strings can result, internal flags and other internal parameters in GSM are sometimes referred to in capital letters.
5.2 GSM EDITING AND DISPLAY COMMANDS:

These commands allow the GSM user to create, edit, and examine (virtual) direct sets.

page command

5.2.2 CREATE Creates a (virtual) direct set by initializing the header block for that set.

CHPNY Changes block number in NAMES directory.

EDIT Prepares to edit an existing direct set.

EDX Variant of EDIT.

5.2.4 DISPLAY Displays lines of text from a direct set.

GPAGF Displays portion of GPAGE buffer.

LIN Move text to GPAGE buffer.

5.2.5 ACCEPT Prepares to add new lines of text to a direct set.

CHANGE Changes one or more lines of text in a direct set.

5.2.7 COPY Copies lines of text within a direct set or from one direct set to another.

MOVE Moves lines of text within a direct set.

PHF Creates a copy of a direct set, retaining same physical sequence.

5.2.8 DELETE Logically deletes lines of text from a direct set.

INSERT Logically inserts lines of text that are already physically present in the direct set.

5.2.9 ESCAPE Escape from text mode.

5.2.10 ALTER Change lines of text containing occurrence of string.

REPLACE Replace occurrences of string within specified lines.

5.2.12 PROTECT Establishes GSM protection over a direct set.

PURGE Removes protection from a direct set.

ENDEDIT Forces rewriting of header block of active set.

SFTNAME Aids in changing name of direct set.

EDMAP Displays pointers for active set.

For reasons of internal logic, the VUELIST (display information from sequential unit) and V (display listable information from a direct set) have been included in the section on Data Transmission rather than in this section.
CREATE A DIRECT SET:

\[ \text{CREATE,} \langle \text{blknum} \rangle(1, \langle \text{setname} \rangle) \]

This command creates an empty direct set beginning at disk block \(\langle \text{blknum} \rangle\) which may be an integer between 1 and 8998. If the optional \(\langle \text{setname} \rangle\) parameter is specified, \(\langle \text{blknum} \rangle\) must be exactly four digits in length (including leading zeroes if necessary). In this case, \(\langle \text{setname} \rangle\) and \(\langle \text{blknum} \rangle\) are entered into the directory contained in the direct set NAMES so that future references to the new set may be made by the symbolic name \(\langle \text{setname} \rangle\) if desired. The string \(\langle \text{setname} \rangle\) must begin with a non-digit and be from one to eight characters long. For additional suggestions concerning \(\langle \text{setname} \rangle\), see the discussion of symbolic set names. CREATE leaves GSM in a state ready to edit the set just created (i.e., an immediately subsequent EDIT serves no essential purpose).

CHANGE BLOCK NUMBER FOR SYMBOLIC NAME:

\[ \text{CHBN,} \langle \text{setname} \rangle, \langle \text{blocknum} \rangle \]

Assumes that \(\langle \text{setname} \rangle\) appears in the directory in the set NAMES. Changes the block number for \(\langle \text{setname} \rangle\) to \(\langle \text{blocknum} \rangle\).

EDIT AN EXISTING DIRECT SET:

\[ \text{EDIT,} (\langle \text{blocknum} \rangle \mid \langle \text{setname} \rangle) \]

Prepares to edit the specified direct set by bringing into core the necessary information from the header block for that set. The first DISPB lines of the set will be displayed, where DISPB is initially 40 and is reset by the \(\langle \text{rlines} \rangle\) parameter of the DISPLAY command. The bottom line of the display will show the first and last block numbers of the set, the number+1 of logical records (card images) currently in the set, the number of pointers currently used in the header block, and the user name and set name found in the header. The set being edited (if any) will be called the "active set". In the current version, the line displayed at the bottom of the screen is also printed on PT06P001 to aid in identifying the set edited.

\[ \text{EDT,} (\langle \text{blocknum} \rangle \mid \langle \text{setname} \rangle) \]

Same as EDIT, except that the printing of the bottom line is suppressed.
DISPLAY A DIRECT SET (OF CARD IMAGES)

(DISPLAY | D) (| <firstsn> (| ,<nlines> |))\%

Displays the first <nlines> in logical sequence beginning
with the logically first occurrence of sn <firstsn>.
If <nlines> is omitted, the value left in DISPB by the last
nonnull <nlines> parameter is used. The initial default is 40.
For the 2260, the user should specify <nlines> as 11 if he
intends to display successive portions of the set. If both
parameters are omitted, the value used in place of <firstsn>
is the sn of the last line displayed by the previous DISPLAY
command. Thus D* will simply go on to the next page-full of
display (repeating the last line of the previous page as the
first line of the new page.) This feature should not be used
to display past the end of the direct set, as unpredictable
results will occur.

DISPLAY GPAGE BUFFER:

GPAGE,<numlines>\%

The first <numlines> 80 byte lines of the GPAGE buffer are
displayed.

MOVE TEXT TO GPAGE BUFFER:

LTV,<linenum>,<text>\%

Where <linenum> is a two digit decimal integer, the <text>
between the second comma and the dollar sign is moved to line
number <linenum> of the GPAGE buffer.

DISPLAY ONE LINE:

GL,<text>\%

Displays a line at the top of the graphic display page. At
present, not only <text> but also the $ and most of whatever follows
will be displayed.
(ACCEPT | A | AS) (|<beforesn>|)

Assumes that some direct set is active. Sets the text acceptance pointer ACCEPTNO to <beforesn>. If <beforesn> is omitted, the default is 30000, which indicates the end of the set. The initial default value for ACCEPTNO is also 30000 if no ACCEPT command has been given. This command leaves 55M in text mode, ready to accept a sequence of text lines typed in. It will remain in text mode until it is told to escape from text mode (e.g., by the ES command). A is the short form of ACCEPT. AS causes the text to be processed in parameter substitute mode. Otherwise the text is processed in opaque mode (no parameter substitution).

CHANGE ONE OR MORE LINES OF TEXT:

(CHANGE | C | CS | CGS | CG),<firstsn>|(1,<lastsn>|

Specifies that lines in logical sequence from that numbered <firstsn> through that numbered <lastsn> are to be presented one at a time to be changed by the user. If <lastsn> is omitted or is equal to <firstsn>, only one line is so presented. The CHANGE command puts 55M into text mode until all of the lines in the specified range have been processed or until the escape text mode command is entered (ES in columns 1-2). Each line in logical sequence is presented at the bottom of the display with the cursor positioned at column 1. The line is then modified by the user and the text delimiter entered by the user to terminate processing of the line. Then the next line is presented, etc.

The command CHANGE causes the text line as modified by the user type-in to replace the existing line. C is the short form of CHANGE. CG is a variant that changes only those characters which have non-blank characters in the line as returned by the user. In the CG command, characters in columns for which a blank is returned by the user remain unchanged and columns for which a slash is returned by the user become blanks, all other columns are changed to the characters returned by the user.

CS and CGS cause the line(s) to be changed to be processed in parameter substitute mode. Otherwise opaque mode is used.
COPY LINES OF TEXT

(COPY | COPYS),<firstsourcesn>),(<lastsourcesn>),
<lastbeforesn>(1,<sourceblocknum> | ,<sourcesetname>))$}

A copy of the lines in logical sequence from <firstsourcesn>
through <lastsourcesn> in the source block is inserted at a
position in the active set logically just before <destbeforesn>.
The new lines are physically located just after the current
physically last line of the active set and are assigned new
numbers accordingly (which are thus necessarily distinct
from any sequence numbers yet existing in the active set).
The source lines are unaffected. If the source blocknum/
setname parameter is omitted, the default is the active set itself.
If <lastsourcesn> is omitted or is equal to <firstsourcesn>, only
one line is copied. In version 3.0 and until further notice,
this command disturbs sequential unit 21.

MOVE LINES OF TEXT

(MOVE | MOVES),<firstsourcesn>),(<lastsourcesn>),
<destbeforesn>$

This command is effectively a COPY,<firstsourcesn>,
<lastsourcesn>,<destbeforesn>$ followed by a DELETE,<firstsourcesn>,
<lastsourcesn>$$. Note that a new physical copy of the lines exists
just as in COPY, and that the old lines are dropped logically but
not physically from the active set.

NOTE: MOVES and COPYS cause the text to be processed in parameter
substitute mode. MOVE and COPY cause the text to be processed
in opaque mode.

COPY DIRECT SET RETAINING SAME PHYSICAL ORDER:

PHY,<blocknum>$

Creates a new direct set at <blocknum> that has the same
pointers, sequence numbers, and physical text sequence as the active
set. WILL NOT GIVE COMPLETELY SATISFACTORY RESULTS ON PROTECTED SETS.
LOGICALLY DELETE LINES FROM ACTIVE SET

DELETE, <snfirst> (1, <snlast>)

Deletes from the active set the logical (not physical) sequence of lines beginning at the logically first occurrence of <snfirst> and ending with the logically next subsequent occurrence of <snlast>. The lines are still physically present in the set, taking up space, but unless the logical and physical sequence of the deleted lines is the same, there is no non-tedious way to get them back logically into the set. (Default for <snlast> is <snfirst>.)

LOGICALLY INSERT LINES IN ACTIVE SET

INSERT, <insertfirstsn>, <insertlastsn>, <beforeasn>

Logically inserts into the active set just before the logically first occurrence of <beforeasn> the lines in physical sequence from <insertfirstsn> through <insertlastsn>. This command assumes that the lines are already physically present in the active set. It does not result in adding any new lines physically to the set. By clever or careless use of this command, it is possible for a single physical copy of a line to occur logically in more than one place in the set. But then if a CHANGE command is performed for that line, all logical occurrences change together. This command is not particularly frequently used from the keyboard, but the internal mechanism is used extensively by the system in carrying out other commands. As an emergency measure, an INSERT can be done for lines not technically within the boundary of the set as determined by the set origin and number of physical lines in the set. Occasionally, this can be used to recover portions of an accidentally clobbered data set. For routine insertion of existing text, the reader is directed to the discussion of the COPY and MOVE commands, which are easier to use but more wasteful of time and disk space.

ESCAPE TEXT MODE

(ESCAPE S P3)

This "command" is an anomaly, since it is effective in text mode rather than command mode. It causes GSM to return to command mode. If used when already in command mode, it may result in a COMMAND REJECTED message.
(ALTER | ALTIRS), <firstsn>, <lastsn>, '<string>''$'

Same as the CHANGE command, except that only those lines in which
the character string <string> occurs will be presented to the user,
one line at a time for optional alteration. Typing $s bypasses the
current line without change (and is to be preferred to hitting the
F6N TO 71 key on the IMPAC). Typing $5 will terminate the command
To change a line, make the desired change in the line as displayed
and hit the text delimiter. ALTIRS causes the lines being changed
to be processed in parameter substitution mode. Otherwise opaque
mode is used. Imbedded apostrophes in <string> are represented by
two consecutive apostrophes for each imbedded apostrophe.

(REPLACE | REPLACS), <firstsn>, <lastsn>, '<oldstring>', '<newstring>''$'

Similar to ALTER command, except that as each occurrence of
<oldstring> is encountered, the user is presented with a display of
the column number of the occurrence and the line of text in which it
occurs and is told to reply $s to replace the occurrence or N$ to skip
over the occurrence without replacement. If $s is typed, an
occurrence of <newstring> replaces the occurrence of <oldstring>.
REPLACS causes processing of lines to be changed in parameter
substitution mode; otherwise opaque mode is used. Imbedded apostrophes
in <oldstring> and <newstring> are handled as in the <string>
parameter of the ALTER command.
PROTECT A DIRECT SET

PROTECT,<blocknum>,<password>,<numblocks>$

Assumes that a direct set exists beginning at <blocknum> (which
must be a numerical value, not a symbolic name). It attempts to protect
<numblocks> of disk storage against clobbering by anything other than
an EDIT,<blocknum>$ for the same <blocknum> (or equivalent symbolic
name), or a system blunder. To remove the protection, the <password>
(1-8 ASCII characters not including a comma or a dollar sign) must be
known. Blocks 6000-6999 are not considered subject to secure protection
and are usually used for temporary storage. (Even these may be
mistakenly protected by a user error.)

REMOVE PROTECTION FROM A DIRECT SET

PURGE,<blocknum>,<password>$

The <password> used in the last successful PROTECT for the area
involved must be supplied here. The PURGE command removes the
protection from all of the blocks protected by the PROTECT command.

FORCE REWRITE OF HEADER BLOCK

ENDEDITs

Assumes that an active set exists. The pointers and other
header information for that set are rewritten into the appropriate
header block.

ASSIGN SET NAME

SETNAME,<setname>$

Stores as the name of the active set, the name <setname> in the
core image of the header information. If an ENDEDIT command or any
command changing the contents of the active set is given, the name
will be written into the header block on disk for the active set.

DISPLAY POINTER MAP

EOMAP$

Displays the pointers showing the logical sequence of text lines
in the active set. The first and last pointers are dummies. This
command is also useful when you have forgotten what set is active,
since it displays that information. The number of text lines and
the last text block number may not be accurate if additions have
been made to the set since the last EDIT command.
5.3 GSM DATA TRANSMISSION COMMANDS:

These commands operate either on a single sequential unit or act to transmit data between a sequential unit and a direct set. All operations assume card images unless otherwise specified.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEW</td>
<td>Rewinds (closes) a sequential unit.</td>
</tr>
<tr>
<td>ZOOF</td>
<td>Writes end-of-file on a sequential unit.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Prints on FT06F001 (usually SYSOUT=A) the entire contents of a sequential unit.</td>
</tr>
<tr>
<td>VUFLIST</td>
<td>Displays a portion of a sequential unit. (Card images or line images)</td>
</tr>
<tr>
<td>TOS</td>
<td>Transmits from specified direct set to seq. unit.</td>
</tr>
<tr>
<td>CONVSQ</td>
<td>Transmits from active set to seq. unit 10.</td>
</tr>
<tr>
<td>NAMER</td>
<td>Writes a NAME &lt;name&gt;(R) card on seq. unit 21 for input to linkage editor.</td>
</tr>
<tr>
<td>W</td>
<td>Writes a card image on specified seq. unit.</td>
</tr>
<tr>
<td>STOP</td>
<td>Transmits from sequential unit to a direct set which it creates.</td>
</tr>
<tr>
<td>SQ</td>
<td>Variant of STOP.</td>
</tr>
<tr>
<td>CARDS</td>
<td>Inserts data from sequential unit into active set.</td>
</tr>
<tr>
<td>FWD</td>
<td>Searches forward on save tape.</td>
</tr>
<tr>
<td>LTOD</td>
<td>Transmits line images from seq. unit to direct set.</td>
</tr>
<tr>
<td>LDS</td>
<td>Inverse of LTOD.</td>
</tr>
<tr>
<td>X</td>
<td>Displays line images from direct.</td>
</tr>
</tbody>
</table>
REWIND (CLOSE) A SEQUENTIAL UNIT:

REW, <unit>s

Closes (rewinds) sequential data set <unit>. (FT<unit>7001)
Note: If you intermix operations (includingREW, etc., as well as data transmission commands) on different sequential units, you may get unexpected and undesirable results and surprise rewrites. So do all operations on one sequential unit together if at all possible.

WRITE END OF FILE:

WEOF, <unit>s

Writes an end of file on sequential unit <unit>. In version 3.0 and until further notice, WEOF is of interest only in the sequence REW, <unit>s WEOF, <unit>s REW, <unit>s to write an EOF at the very beginning of a sequential unit. This is subject to change in future versions.

PRINT A SEQUENTIAL UNIT:

PRINT, <unit>s

Prints on unit FT06F001 the entire contents of sequential <unit>. The logical record length on <unit> may be 80, 120, 121, or any of a variety of values not greater than 121 (133??).

DISPLAY A SEQUENTIAL DATA SET:

(VUELST | V) ((<unit>(1,<linesperpage> 1)))s

If <unit> is specified, that sequential unit is rewound and the first <linesperpage> (or to a carriage restore control character) are displayed. If <linesperpage> is omitted, the last value supplied for that parameter is used. The original default value is 40. (Better use 11 for the 2260). If both arguments are omitted, subsequent display pages are presented. When the end of file is reached, --END OF DATA-- is displayed.
TRANSMIT DATA IN LOGICAL SEQUENCE FROM A DIRECT SET TO SEQUENTIAL UNIT:

DTOS,(<setname>,<blknum>)(<unit>,((<firstsn>,<lastsn>)))$%

WITHOUT BEGINNING THE SEQUENTIAL UNIT <unit>, successive card images in logical sequence in the direct set specified by <setname> or <blknum> are transmitted to the sequential <unit>, beginning with the logically first occurrence of <firstsn> and ending with the logically next subsequent occurrence of <lastsn>. If <lastsn> = <firstsn>, only one line is transmitted. In both <firstsn> and <lastsn> are omitted, as is most commonly the case in present usage, the entire direct set is transmitted in logical sequence. If <unit> is omitted, the last value used in a similar command is used.

CONVSQ$

This command has the effect of REW,10$ DTOS,<activeset>,10$
REW,10$, where <activeset> is the block number of the active set.$%

WRITE LINKAGE EDITOR NAME CONTROL CARD:

NAME,<name>?

Where <name> is a string of from 1 to 8 characters not including any delimiters, a card image of the form

NAME <name>(?)

is written on sequential unit 21. This is most commonly used to inform the linkage editor that the preceding object code written by some means onto unit 21 is to go into a load module to be called <name>.

WRITE TO SEQUENTIAL UNIT:

W,<unit>,<text>$%

Were text contains no dollar sign, this command writes on sequential unit <unit> a card image with text copied from columns 6-72 of the W command in columns 1-67 of the card image, except that the command delimiter is replaced by a blank. Columns 73-80 of the written card are blank. The W command should, for compatibility with possible future changes, contain no non-blank characters between the command delimiter and the end of the command line. This will normally be the case when the W command is entered directly from the keyboard, but care must be exercised in using it in a procedure. To avoid error, <unit> should be written as a two-digit number.
TRANSMIT CARD IMAGES FROM SEQUENTIAL UNIT TO DIRECT SET:

(STOD | STODS),<unit>,(<setname> | <blocknum>)$3

Without rewinding sequential unit <unit>, this command transmits the card images from <unit> to the direct set specified by <setname> or <blocknum>, beginning with the first physical line of the direct set and continuing with physical and logical sequence coinciding. Data transmission is terminated by an end of file on <unit> or a card image on <unit> with a blank in column 1 and /* in columns 2-4. Sequence numbers are inserted in columns 77-80.

SD,(<setname> | <blknum>)$3

Most commonly used to store object code just produced by a language processor, this command has the same effect as PER,21$ followed by STOD,21,<setname or blknum>$3, with the restriction that if the <setname> form is used, the first character of the symbolic set name must be a slash.

(CARDS | CARDSS),<unit>$3

Without rewinding sequential <unit>, card images are transmitted in the same manner as in STOD, except that they are logically inserted at a point just before the most recent ACCEPT NO value supplied by the ACCEPT command. Note: Since ACCEPT is usually used just before this command, it is necessary to escape back from text mode into command mode before giving this command. NOTE: STODS and CARDSS differ from STOD and CARDS in that lines of text are processed in parameter substitution mode rather than in opaque mode.
SEARCH FORWARD ON SAVE TAPE:

FWD,<name>$

Searches forward on unit 21 (presumably a GSW save tape) until the set <name> is found, leaving unit 21 positioned just before the set and displaying a message to the effect that the set was found. If not found, a message to that effect is displayed.

TRANSMIT LISTABLE DATA FROM SEQUENTIAL TO DIRECT:

LTOD,<unit>,<blknum>$

Without rewinding <unit>, this command writes the print line images (assuming no more than 121 characters per line) from <unit> into consecutive direct blocks beginning with <blknum> + 1, sixteen lines per block up to a maximum of 298 blocks (4768 lines) or until an end of file is encountered on <unit>. Such a set cannot be edited.

TRANSMIT LISTABLE DATA FROM DIRECT TO SEQUENTIAL:

LDI,<blocknum>,<unit>$

This is essentially the inverse of LTOD, but infrequently used and probably not completely debugged.

DISPLAY LISTABLE FROM DIRECT:

X,([<blocknum>,<linesperpage>])$

This command is similar to VUELIST except that the print lines are obtained for display from direct set blocks (written by LTOD) and can be accessed randomly. The S-type carriage control character interpretation facilities of this command are not as sophisticated as VUELIST, so some left margin irregularities are to be expected when displaying assembler output.
5.4 INVOKING OS PROCESSORS, USER PROGRAMS, AND GSM SERVICE ROUTINES:

The commands in this section enable the GSM user to invoke dynamically the various OS language processors (assembler, compilers, linkage editor, utilities, etc.) in order to perform functions such as assembling or compiling his program, linkage editing it, etc. They also allow the user to execute his own program, or a program from a library developed in support of GSM or a program from a library developed by the user and his coworkers.

5.4.2 ASMG  Invoke the assembler.
FTNG  Invoke the FORTRAN compiler.
PLI  Invoke the PL/I compiler.
LINK  Invoke the linkage editor.
PLINK  Variant of LINK for PL/I.
PGS  Change library to receive new load modules.

5.4.3 LOAD  Invoke the loader.
PLOAD  Variant of LOAD for PL/I.
G  Execute user program.
K  Variant of G.
UTILITY  Invoke an OS utility program.

5.4.4 ALIST  Change alternate DNAME for assembly listing output
ARBJ  
ASOU  
FJST  
FORJ  
PSOU  
LIST  
LOBJ  
PLIST  
PORJ  
PSOU
INVOKE OS/360 ASSEMBLER:

ASMG#

Invokes the G-level assembler. Source input is assumed to be on sequential unit 10. Assembly listings will go to unit 12. Object program card images are written on unit 21.

INVOKE FORTRAN COMPILER:

FNG#

Invokes the G-level FORTRAN compiler. Source input is assumed to be on sequential unit 10. FORTRAN listings will go to unit 15. Object program card images are written on unit 21.

INVOKE PL/I COMPILER:

PLIS

Invokes the PL/I compiler. Source input is assumed to be on sequential unit 10. PL/I listings are written on unit 17. Object program card images are written on unit 21.

INVOKE LINKAGE EDITOR:

LINK#

Invokes the Linkage Editor. Object program card images are assumed to be on unit 21. The linkage edit map is written on unit 12. The load module(s) created by the linkage editor are placed in the partitioned data set specified by the SYSLMOD DD statement in the call deck, unless the PDS,<ddname> command has been given to specify a different DD statement. The FORTRAN subroutine library is used.

PLINK#

Same as LINK#, except that the PL/I subroutine library is used in place of the FORTRAN library.

CHANGE LIBRARY TO RECEIVE LOAD MODULES:

PDS,<ddname>3

Where <ddname> is the Ddbname of a DD statement in the call deck, this command causes that DD statement to be used in future linkage edits in place of SYSLMOD. PDS,SYSLMOD3 restores the original state.
EXECUTE USER PROGRAM ON CMS SERVICE ROUTINE:

G,<modulename>*
K,<modulename>‡

The load module whose entry point is specified by <modulename> is executed. If the G command is used, the partitioned data sets specified in the SYSLMOD and SYSLMAD DD statements will be searched for the desired module and no attempt will be made to execute the module if it is not found there. Instead, a message "MEMBER NOT FOUND" will be displayed. If the K command is used, no such preliminary search will be performed. Instead, the module will be executed (or an 806 ABEND occur if the module is not present in JCLLIB or LINKLIB) without checking. Thus, when it is known that the module is not in SYSLMOD or SYSLMAD, one should use the K command. Otherwise, the G command provides some useful checking against misspelling, etc.

EXECUTE AN OS UTILITY:

UTILITY,<utilityname>‡

The OS/360 utility with entry point <utilityname> is invoked from LINKLIB. Input card images are assumed to be on unit 21, output listings go to unit 12.
CHANGE DONAMES FOR INVOKING OS PROCESSORS:

These commands, all of the form

<command>, <ddname>

change the DONAME specified for one of the input/output units associated with invoking an OS processor. The following table indicates what processors and units are involved:

<table>
<thead>
<tr>
<th>command</th>
<th>processor</th>
<th>type of data</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOBJ</td>
<td>ASMG</td>
<td>source listing output</td>
<td>FT12P001</td>
</tr>
<tr>
<td>ASOJ</td>
<td>ASMG</td>
<td>object module output</td>
<td>FT21P001</td>
</tr>
<tr>
<td>AOBJ</td>
<td>ASMG</td>
<td>source card input</td>
<td>FT10P001</td>
</tr>
<tr>
<td>PLIST</td>
<td>FTNG</td>
<td>source listing output</td>
<td>FT15P001</td>
</tr>
<tr>
<td>FSJO</td>
<td>FTNG</td>
<td>object module output</td>
<td>FT21P001</td>
</tr>
<tr>
<td>FSOU</td>
<td>FTNG</td>
<td>source card input</td>
<td>FT10P001</td>
</tr>
<tr>
<td>LLIST</td>
<td>LINK</td>
<td>linkedit map, etc.</td>
<td>FT12P001</td>
</tr>
<tr>
<td>LOBJ</td>
<td>LINK</td>
<td>object module input/control</td>
<td>FT21P001</td>
</tr>
<tr>
<td>PDS</td>
<td>LINK</td>
<td>load module output</td>
<td>SYSLMOD</td>
</tr>
<tr>
<td>PLIST</td>
<td>PL1</td>
<td>source listing output</td>
<td>FT17P001</td>
</tr>
<tr>
<td>POBJ</td>
<td>PL1</td>
<td>object module output</td>
<td>FT21P001</td>
</tr>
<tr>
<td>PSOJ</td>
<td>PL1</td>
<td>source card input</td>
<td>FT10P001</td>
</tr>
</tbody>
</table>
GSM CONTROL PROCEDURES AND RELATED COMMANDS:

A GSM control procedure is a sequence of GSM commands stored in a (virtual) direct set for use as a "GSM-language subroutine." The INVOKE (blockname) command causes the specified block of GSM commands to be executed. Since a GSM control procedure may itself issue an INVOKE, procedures may be nested indefinitely (but not recursively).

These procedures are most convenient when carrying out a frequently used sequence of commands, but where considerations of flexibility or complexity preclude building the whole process into the GSM command analyzer as a new command. By combining the use of the short form of the INVOKE command (I), and the format parameter substitution option and the symbolic naming technique for procedure blocks, the user has all of the convenience of "do-it-yourself" command definition at the expense of typing only two extra characters "I, ".

Procedures may be created and edited just as other text, with the following important exception: PROCEDURE STEPS ARE EXECUTED IN PHYSICAL RATHER THAN LOGICAL SEQUENCE, so that when inserting and/or deleting lines in a procedure, it is often necessary to resequence the text in the procedure block.

One should also note that it is not possible to use protected procedures, since the INVOKE command modifies the procedure block to specify the return linkage for nesting.

page command

5.5.3 INVOKE Invoke a GSM control procedure.
5.5.4 JP Jump to a specified statement in the current procedure block or another procedure block.
JR Jump return from procedure.
RETURN Exit from procedure to terminal control.
5.5.5 JLE Conditional jump on return code.
JGE "" "" "" ""
SRC Set return code.
IRC Increment return code.
DRC Decrement return code.
WAIT Wait for specified period of time.
5.5.6 LAP Load actual parameter from text line.
STAP Store actual parameter in text line.
SAP Set actual parameter.
RCD Convert return code to decimal.
RCH Convert return code to hexadecimal.
5.5.7 PON Enter Command substitute mode (substitute for formal parameters in GSM commands).
POPF Leave command substitute mode.
USN Set parameter from user name.
JBN Set parameter from job name.
EDS Set parameter from edit set block number.
DLM Set parameter from display line number.
DTM Convert parameter decimal to hexadecimal.
PARAMETER SUBSTITUTION IN GSM COMMANDS AND TEXT:

In GSM 3.0 there are eight formal parameter symbols:

:1 :2 :3 :4 :5 :6 :7 :8

At any given time, with each of these formal parameters there is associated a value—the corresponding actual parameter—which is a string of from 0 to 18 EBCDIC characters none of which is a GSM delimiter. These actual parameter values may be established explicitly by means of GSM commands such as Set Actual Parameter (SAP), which has the form

SAP,<pnum>,<value>

and assigns a string determined by <value> to the formal parameter :<pnum>. For example, SAP,5,APC5 assigns the three character string ABC as the actual parameter corresponding to the formal parameter :5. More commonly, however, the actual parameter values will be established implicitly by occurring positionally as arguments of a procedure call. For example, INVOKE,PROC1,ABC,635 assigns the three character EBCDIC string ABC to :1 and the two character EBCDIC string 63 to :2, just before it passes control to the GSM procedure named PROC1. Within PROC1, occurrences of :1 will be replaced by ABC and of :2 by 63 (provided that certain conditions discussed below in connection with substitution modes obtain.) Parameter substitution potentially applies to all but a very few cases where a line of information is being processed by the GSM command analyzer CMDA (either as a command or as a line of text) on the way from an interactive terminal or a sequential data set to a destination in a GSM direct set. It applies only in substitute mode (defined below), and it gives exceptional treatment to consecutive character pairs composed of a colon and a digit of a colon and a letter of the alphabet. (The latter case, that of the alphabetic character following a colon, is not yet defined in GSM 3.0, but the reserved nature of the character pair should be observed for compatibility with future versions.) Parameter substitution consists of replacing each occurrence of a formal parameter by the corresponding actual parameter string (even when the latter is of a different length than the two character formal parameter—the formal parameter may even be of zero length, in which case the effect is merely to omit the occurrence of the formal parameter).

Upon entry to CMDA (subject to possible change in a later version so that popping up to the GSM driver will not affect this mode), the system is in command opaque mode, so that no parameter substitution is done while analyzing GSM commands for execution. The command PCN switches to command substitution mode, in which parameter substitution takes place before the command or its arguments are analyzed prior to execution. POPF switches back to
command opaque mode. There is one notable exception to the statement made above about command opaque mode: In order to achieve maximum forward compatibility (maximum likelihood that something that could legally be done in an earlier version will still give the same results), if a formal parameter occurs by itself in an argument field of a GMS command, parameter substitution will be carried out as in previous versions of GMS.

For lines being processed as text destined for a direct set (e.g., the processing performed by the commands ACCEPT, CHANGE, MOVE, COPY, STOP, etc.) the situation is independent of whether or not command substitution mode is in effect. To maintain maximum forward compatibility, the text processing of these commands is precisely the same as in earlier versions of GMS--no parameter substitution takes place. Instead, an additional set of commands has been added to GMS paralleling the function of these commands:

<table>
<thead>
<tr>
<th>text opaque</th>
<th>text substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AS</td>
</tr>
<tr>
<td>C</td>
<td>CS</td>
</tr>
<tr>
<td>STOD</td>
<td>STODS</td>
</tr>
<tr>
<td>CARDS</td>
<td>CARDS</td>
</tr>
<tr>
<td>MOVE</td>
<td>MOVES</td>
</tr>
<tr>
<td>COPY</td>
<td>COPYS</td>
</tr>
</tbody>
</table>

where, of course, A is the short form of ACCEPT and C is the short form of CHANGE. (No long forms have been provided for AS or CS, since experience has shown that users tend to avoid the long forms in favor of the short forms.)

Note that the processing of the arguments of these commands, (and indeed of the characters in the command field as well), as the commands are being analyzed prior to execution, is governed by whether or not command substitute mode is in effect (both for the text opaque and the text substitution commands), while the processing of the text data by the routine carrying out the command within CMNDA is governed by the presence or absence of the S on the command and is independent of command substitute mode.
INVOKING A SUB-MONITOR CONTROL PROCEDURE:

(INVOKE | I | CP | Z), (⟨proclname⟩ | ⟨blocknum⟩), ⟨par1⟩,..., ⟨parN⟩
(n LP 8)

This command invokes a procedure composed of user-written and edited GSM commands. The procedure may be referenced either by a symbolic name ⟨proclname⟩ or by the actual block number ⟨blocknum⟩ of the direct set. It may be given directly from the interactive terminal or may itself occur within a procedure. Thus procedure calls may be nested arbitrarily deep (but recursion is not permitted). If we let ⟨new⟩ denote the direct set containing the procedure about to be invoked and ⟨old⟩ denote the direct set containing the procedure currently in control (⟨old⟩ = 0 if the new procedure is being invoked directly from the terminal), then the linkage housekeeping internal to GSM may be described as follows: The actual parameter values in force at the time of the new call are stored in SN 23-24 of ⟨new⟩ and a JP, ⟨snold⟩+1, ⟨old⟩5 (where ⟨snold⟩ is the SN of the INVOKE statement) is stored in SN 25 of ⟨new⟩, providing a return jump to the caller. This return jump may be reached from within the ⟨new⟩ procedure by a JP, 25 and from any other point by a JP, 25, ⟨new⟩. After the old parameter values have been saved and the link planted, ⟨par1⟩,..., ⟨parN⟩ are loaded as the new actual parameters (with null strings loaded for any missing parameters), and control is passed to SN 1 of ⟨new⟩; that is, the next GSM command is fetched from the first 80 bytes of the data in the direct set ⟨new⟩ and subsequent commands are fetched from subsequent card images in that set until something happens to alter the path of control.
JUMPING WITHIN OR BETWEEN PROCEDURES WITHOUT CHANGING PARAMETERS:

JP,<sn>)(,<proclname> | ,<blocknum> | )$

This command passes control to another statement within a procedure block or to a statement in another procedure block. It may also be used to execute a procedure without any linkage or parameter manipulation operations. The optional second argument specifies the block in which the new procedure is located; if omitted, the current procedure block is assumed. The statement within the block is specified by <sn> (where <sn> is between 1 and 25 inclusive).

EXITING FROM A PROCEDURE (JUMP RESTORE):

JR,<sn>,<blocknum>$

The current actual parameter values are discarded and replaced by those stored in SN 23-24 of the procedure block containing the JR command. Control is then passed to the procedure in the direct sat <blocknum> at statement number <sn>. This is a "subroutine return" type exit to the caller of the procedure block. Most commonly it is not written by the user but planted by the system in SN 25 when the procedure is invoked.

EXITING FROM A PROCEDURE (RETURN TO TERMINAL CONTROL):

RETURNS

This command does not affect actual parameter values. It returns control to the terminal keyboard, bypassing any intervening calling procedures (if any) that may have led to the procedure giving the RETURN statement.
JUMP IF RETURN CODE LESS THAN OR EQUAL TO GIVEN VALUE:

JLE,<hexval>,<sn>$

Control is passed to statement <sn> in the current procedure block if RETC (the return code from the last OS processor or load module executed, unless altered by intervening GSM commands) is less than or equal to the value of <hexval> interpreted as a hex integer (composed of EBCDIC characters 0 through F).

JUMP IF RETURN CODE GREATER THAN OR EQUAL TO GIVEN VALUE:

JGE,<hexval>,<sn>$

Same as JLE except for reversing the inequality.

SET RETURN CODE:

SRC,<hexval>$

The return code RETC is set to the (binary value of) the unsigned hex integer <hexval>.

INCREMENT RETURN CODE:

IRC,<hexval>$

The binary value of RETC is increased by the (binary value of) the unsigned hex integer <hexval>.

DECREMENT RETURN CODE:

DRC,<hexval>$

The binary value of RETC is decreased by the (binary value of) the unsigned hex integer <hexval>.

PUT COMMAND TASK IN WAIT STATE FOR SPECIFIED TIME INTERVAL:

WAIT,<tenthsec>$

The command analyzer task is put in wait state for <tenthsec> tenths of a second.
LOAD ACTUAL PARAMETER:

\[ \text{LAP,} \langle \text{parnum} \rangle, \langle \text{bytenum} \rangle(1, \langle \text{blocknum} \rangle, \langle \text{setname} \rangle) 3 (1 \text{.LE.} \langle \text{parnum} \rangle \text{.LE.} 3) \]

Actual parameter number \( \langle \text{parnum} \rangle \) (i.e., the value corresponding to formal parameter \( \langle \text{parnum} \rangle \)) is loaded from the first 20 bytes beginning with \( \langle \text{bytenum} \rangle \) (.LE. 1980) of text in the specified direct set. The format of actual parameter values is such that the first two of the twenty bytes specify the length-1 of the string and the leftmost of the remaining eighteen specify the string itself. If the block parameter is omitted, the first text block of the current procedure is used as default.

STORE ACTUAL PARAMETER:

\[ \text{STAP,} \langle \text{parnum} \rangle, \langle \text{bytenum} \rangle(1, \langle \text{blocknum} \rangle, \langle \text{setname} \rangle) 5 \]

This command is the inverse of the LAP command.

SET ACTUAL PARAMETER:

\[ \text{SAP,} \langle \text{parnum} \rangle, \langle \text{string} \rangle 3 (1 \text{.LE.} \langle \text{parnum} \rangle \text{.LE.} 3) \]

Where the EBCDIC string \( \langle \text{string} \rangle \) contains from 0 through 18 characters, that string is used to set the actual parameter number \( \langle \text{parnum} \rangle \) (the one corresponding the the formal parameter \( \langle \text{parnum} \rangle \)).

CONVERT RETURN CODE TO DECIMAL:

\[ \text{RCD,} \langle \text{parnum} \rangle 5 \]

The binary value of PETC is converted to the four digit EBCDIC representation of its decimal value, which is assumed to be between 0 and 9999, inclusive. The resulting four character string is assigned as the value of actual parameter number \( \langle \text{parnum} \rangle \).

CONVERT RETURN CODE TO HEXADECIMAL:

\[ \text{RCH,} \langle \text{parnum} \rangle 5 \]

The binary value of PETC is converted to the 3-digit EBCDIC representation of its hexadecimal value. The resulting eight character string is assigned as the value of actual parameter number \( \langle \text{parnum} \rangle \).
CHANGE COMMAND SUBSTITUTE/COMMAND OPAQUE MODE:

PON$

This command puts GSM into command substitute mode so that before each command (not line of text) is analyzed and executed, all occurrences in the command and its arguments of formal parameters are replaced by the corresponding actual parameter values. This does not affect in any way the substitution mode applied to lines entered or processed as text rather than as commands. That mode is determined by the form of the text processing command (usually an "S" suffix).

POFF$

This is the inverse of PON. It puts GSM in command opaque mode so that formal parameters occurring in command lines (not text lines) are treated naively as the two character strings that they appear to the naked eye to be. Exception: if a formal parameter occurs alone in an argument field, it will, for forward compatibility purposes, be processed as in older versions of GSM—it will be replaced by the corresponding actual parameter. The POFF command does not affect the substitution mode applied to lines being processed as text.

SET ACTUAL PARAMETER FROM VARIOUS SOURCES:

USN,$\langle\text{parnum}\rangle$

Sets actual parameter number $\langle\text{parnum}\rangle$ to the current username (a four character string left adjusted with blank fill).

JBN,$\langle\text{parnum}\rangle$

Same as USN except that the current eight character jobname is used.

EDS,$\langle\text{parnum}\rangle$

Same as USN except that the four digit decimal (EBCDIC) integer representation of the current active set block number is used.

DLN,$\langle\text{parnum}\rangle$

Same as USN except that the four digit decimal (EBCDIC) integer representation of the line number of the top of the current active set page display is used.

DTH,$\langle\text{parnum}\rangle$,$\langle\text{decimalvalue}\rangle$

Sets actual parameter number $\langle\text{parnum}\rangle$ to the 8-digit hexadecimal (EBCDIC) integer representation of the decimal (EBCDIC) second argument.
### 5.6 Control and Status Commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>Identify the current user.</td>
</tr>
<tr>
<td>GETOFF</td>
<td>Return control to the GSM supervisor.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>Cancel current job (supervisor command).</td>
</tr>
<tr>
<td>CMNDAC</td>
<td>Use a different version of the GSM command analyzer.</td>
</tr>
<tr>
<td>ABEND</td>
<td>Abnormally terminate with dump and return to GSM supervisor.</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>Submit JCL for a JOB to HASP.</td>
</tr>
<tr>
<td>JQ</td>
<td>Inquire as to status of all class G jobs.</td>
</tr>
<tr>
<td>JQX</td>
<td>Inquire as to status of all jobs in specified queue.</td>
</tr>
<tr>
<td>JN</td>
<td>Inquire as to status of single job.</td>
</tr>
<tr>
<td>N</td>
<td>No-operation.</td>
</tr>
<tr>
<td>WTO</td>
<td>Write to 360 operator.</td>
</tr>
<tr>
<td>DA</td>
<td>Display status of all jobs in execution.</td>
</tr>
<tr>
<td>PCON</td>
<td>Begin logging all GSM commands on printer.</td>
</tr>
<tr>
<td>PCOFF</td>
<td>Nullify PCON.</td>
</tr>
<tr>
<td>LISTUN</td>
<td>Divert command logging to specified unit.</td>
</tr>
<tr>
<td>GM27</td>
<td>Accept input from 2741.</td>
</tr>
<tr>
<td>GMID</td>
<td>Nullify GM27.</td>
</tr>
<tr>
<td>CASE</td>
<td>Accept upper/lower case text.</td>
</tr>
<tr>
<td>UPER</td>
<td>Accept only upper case text.</td>
</tr>
<tr>
<td>GC</td>
<td>Special upper/lower display mode.</td>
</tr>
<tr>
<td>GU</td>
<td>Nullify GC.</td>
</tr>
<tr>
<td>CA</td>
<td>Present lines for case change of individual characters.</td>
</tr>
<tr>
<td>NOD</td>
<td>Allow omission of delimiter.</td>
</tr>
</tbody>
</table>
IDENTIFY USER:

USER, <username> ;

Where <username> is a three or four byte user identification, this command identifies the current user to the GSM system. Once given, it will suppress the request to IDENTIFY YOURSELF which appears on the screen following each return to the GSM command analyzer CMNDA from an OS processor or load module being executed.

RETURN TO GSM SUPERVISOR:

(GETOFF, QQ) ;

Passes 360 control back to the GSM supervisor. QQ is effective even in text mode.

CANCEL JOB:

CANCELS (on the 2741 CANCEL call return)

Strictly speaking not a GSM CMNDA command, this causes the GSM supervisor (when it is in control, e.g., after GETOFF or after certain CMNDA and OS processor ABENDS) to return control to the operating system, terminating the job step.

LOAD NEW COPY OF COMMAND ANALYZER:

(CMNDA | CC), <name> ;

This relatively infrequently used command causes the GSM driver to replace the current CMNDA load module by one with the name <name>. This allows the use of experimental versions of CMNDA, in which case the experimental version continues to be used until control is returned to the GSM supervisor. It can also be used to refresh CMNDA when that module has been paralyzed (for disk protection reasons) by a disk protect error.

ABEND TO GSM SUPERVISOR OR DRIVER WITH DUMP:

ABENDS

Causes an abnormal termination of CMNDA with a dump to SYSUDUMP or SYSABFND (depending on your JCL) and returns control either to the GSM supervisor or to the GSM driver depending on the version of the system in use at the time. Completion codes in the dump are SYSTEM = 000 USER = 0222.
SUBMIT JCL FOR A JOB:

(SUBMIT | SUBN), (<setname> | <blocknum>)

Assumes that the specified direct set contains JCL (and possibly a limited amount of SYSIN data) for a job to be submitted, and that the JOB card in the JCL is such that JOB does not occur before column 12. The third and fourth characters of the job name are modified by adding 1 to their value interpreted as a decimal integer (if either character is not already a decimal digit it is treated as a zero).

INQUIRE AS TO STATUS OF ALL CLASS G JOBS:

JQ$

Interrogates HASP to get the current status of each job appearing in the class G queue and displays the information.

INQUIRE AS TO STATUS OF JOBS IN SPECIFIED QUEUE:

JQX,<qnum>$

Similar to JQ, except that <qnum> specifies the queue to be examined:

0 - all jobs in system
90 - all jobs in execution
2 - all jobs in output queue
45 - all jobs in E queue
42 - all jobs in B queue (B = hex 42 + hex 80)
56 - all jobs in O queue (O = hex 56 + hex 80)

INQUIRE AS TO STATUS OF SINGLE JOB:

JW,<jobname>$

Without regard to choice of queue, interrogate HASP as to status of job <jobname>.
NO-OPERATION:

Does nothing. Especially convenient for deleting steps from procedures without resequencing.

WRITE TO 360 OPERATOR:

WTO, <message>$ (no delims within <message>)

Does a WTO macro to send <message> to the 360 operator console.

DISPLAY STATUS OF ALL JOBS IN 360 EXECUTION OR INITIALIZATION:

DA$

Displays, two jobs per line, the jobname and stepname of each job in the os system. For each job, the format is <jobname>$<stepname>$ (with no intervening blank unless <jobname> happens to be less than eight characters in length.) If a job isn't really going yet (may be held up for core or equipment allocation) the display HASPRINT$<jobname> will appear. If there are many of these, particularly if the display doesn't change when the command is repeated over a substantial period of time, think twice before cancelling your job and assuming that the next G job will come up promptly.

COMMAND LOGGING CONTROL:

PCON$

Causes all commands to be printed on <listunit> (initial default is <listunit> = 6).

PCOFF$

Nullifies PCON.

LISTUN,<unit>3

Causes the command printing referred to in PCON to go to sequential unit <unit> instead of to the default. The record length for <unit> must be at least 120. If you are trying to use this information by printing and/or viewing it, do a LISTUN,63 before V or PRINT or all you may see is the V or PRINT command. A PCOFF may also prevent this problem.
CHANGE KEYBOARD INPUT SOURCE:

GM27$

Causes the 2741 to replace the graphic device (IODIOM, etc.) as the source of keyboard input. Lines to be changed by the C, CS, CG, or CFS commands are typed out on the 2741 to prompt the user. All graphic displays remain essentially unaffected.

GM10$

Nullifies GM27$

UPPER/LOWER CASE CONTROL:

CASE$

Causes text (not commands) from the 2741 to be treated as upper and lower case characters instead of being forced to upper case as in the default mode.

UPPER$

Cancels the effect of the CASE command.

GC$

Causes the graphic display of pages from a direct set to be modified so that each line of text is replaced by two consecutive lines, the second of which has asterisks under upper case letters and blanks elsewhere.

G1$

Nullifies GC.

CA,<firstsn>,<lastsn>$$

A variant of the CG command, this command causes each line in logical sequence from <firstsn> through <lastsn> to be presented to the user for possible change in upper/lower case of individual letters in the line. Typing a slash forces to lower case, typing a period forces to upper case, and a blank leaves the character alone. Use the slash and period only on letters.

ALLOW DELIMITER OMISSION:

NOD$

Causes the delimiter appropriate to the mode (text/command, etc.) to be supplied if the user types in the wrong delimiter or no delimiter at all. Very convenient for the 2741, but use with care, since some of the error checking features of GSM are lost.