Global Development System Notes
System Variables Part-1

1. Introduction

This document describes all the System Variables that are available to 32-bit Global application programmers.

The System Variables are described in a thematic order. An index will be provided to list all the routines in strict alphabetic order.

Important Note: This document is under constant review and the internal structure is subject to change.

2. System Variables in the 16-bit Address Space

2.1 $$TICK ?????????????? ??? PIC 9(4) COMP

2.5.6 The Clock Interrupt Period, $$TICK

```
++++++++++++++++++++++++++++++++++++++++
System variable $$TICK is a PIC 9(4) COMP  field  which  contains
the  clock  interrupt  period  in milliseconds.  It indicates the
frequency at which you will receive control if  you  establish  a
clock  interrupt  intercept  routine,  as  described  in  section
2.3.2.
```

2.2 $$BYTE ?????????????? ??? PIC X

7.1.10 The Tap Selection Byte, $$BYTE

```
++++++++++++++++++++++++++++++++++++++
The system variable $$BYTE is a PIC X field used in creating
TAPs. It is mentioned here only for completeness.
```

2.3 $$11-L ?????????????? ??? PIC 9(4) COMP

LEN OF CURSOR POS'N

2.4 $$AC-5 ?????????????? ??? PIC 9 COMP

7.1.9 The Version 5.0 ACCEPT Simulation Flag, $$AC-5

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++
The system variable $$AC-5 is a PIC 9 COMP field that allows you
to simulate the V5.0 ACCEPT...LINE...[COL...] statement. By
setting $$AC-5 to 1, the ACCEPT...LINE...[COL...] statement will
cause the input area immediately to the right of the colon to be
erased by writing blanks to it. Once set, this condition will
remain in force until the end of job when the field will be set
back to 0.
```

2.5 $$ARCH ?????????????? ??? PIC X

2.5.2 The Architecture Code, $$ARCH

```
++++++++++++++++++++++++++++++++++++
```

System variable $$ARCH is a single-character, PIC X field identifying the architecture of your BOS computer and thus the instruction set used by its assembler language programs. Table 2.5.2 shows those architectures currently supported, together with the character codes assigned to $$ARCH, and the associated address forms.

ARCHITECTURE	$$ARCH	$$ADDR
Intel 80286, 80386 in protected mode	A	2
Intel 8088, 8086, 80186, 80286 and 80386 in real mode	J	2
Motorola 68000, 68010, 68020	K	1
DEC VAX	V	2
PDP-11	Z	2

Table 2.5.2 - Architecture Codes and Address Forms

If you provide functionally identical assembler language programs for a number of different architectures, then you should incorporate the architecture code in their program-ids so that the correct program is readily identifiable and easily loaded. For example, the assembler language components of the BOS Cobol compiler, the TRAMS interpreter, is given a command program name of the form $aTRAM, where a is the architecture code. Thus
/ /
$JTRAM is the version used on any Intel 8086 family processor, $ZTRAM the version for the PDP-11 and so on. The compiler's initiation overlay simply constructs the name using $$ARCH. Then it uses an ENTRY$ call to see if the interpreter has already been placed on the system stack by LOAD customisation. If this is not the case it invokes LOAD$ to load the interpreter as a temporary user stack item.

2.6 $$AREA Application Work Area PIC X(16) R/W

The Application Work Area consists of 16 bytes within the system area which are available for application use. The bytes are set to binary zero at the start of a session. However, once this area has been erased in this way Global System Manager never refers to it again. Thereafter the 16 bytes can be used for any application purpose whatsoever. The work area allows you to pass a small amount of information between programs very conveniently.

To refer to the Application Work Area you simply redefine $$AREA. For example:

```
01    AP REDEFINES $$AREA
  03  APINIT      PIC 9 COMP        * INITIATION FLAG
  03  APPRIV      PIC 9(2) COMP     * OPERATOR PRIVILEGE LEVEL
  03  APNUMB      PIC 9(4) COMP     * OPERATOR NUMBER
```

In practice, of course, you would probably place the entire definition of the Application Work Area in a copy book, so that each programmer need only code, for example:

```
COPY AP
```

The 16-bit compatible Application Work Area, $$AREA, is completely separate from the 32-bit only Extended Application Work Area, $$AR32 (see section 3.??). Thus, the total work area available for 32-bit applications is 272 bytes.

2.7 $$ATIM ?????????????? ??? PIC 9 COMP

7.1.13 The Accept Time Out, $$ATIM
++++++++++++++++++++++++++++++++++
This is a PIC 9 (2) COMP field used to cause the subsequent
ACCEPT statement to "Time Out" if no operator input is keyed
within a specific time. It can be set to any period in seconds
between 1 and 60 (if set to 0 then "Time Out" will not occur).
The time-out is cancelled as soon as the operator keys any
character, or if there were any characters in the type-ahead
buffer when the ACCEPT was invoked. A NULL clause must be
coded. A null response is indicated by $$COND = 1. Time-out is
indicated by $$COND = 2. See section 7.8.2 for use with CHAR$.

2.8 $$AUTO ?????????????? ??? PIC 9 COMP

7.1.2 The Auto-input Flag, $$AUTO
+++++++++++++++++++++++++++++++++
Usually each input keyed by the operator must be completed by a
terminator, normally <CR> or a control sequence such as <CTRL A>
-- ------
or <ESCAPE>. However, when the system variable $$AUTO, a PIC 9

COMP field, is set to 0, input will be automatically terminated
when the operator keys the last character of the field, the
length of which is implied by the ACCEPT, ACCEPT...LINE or CALL
ACCE$... statement responsible for the operation. The auto-input
flag only applies to a single accept: you must set it to zero
before each statement to be affected.

Auto-input is useful in creating simple dialogues where, for
example, the operator selects from a menu of actions by keying
just a single character. It also allows for the support of
terminals which cannot transmit <CR> or other control sequences,
for example telephone key pads.

You should note that even when $$AUTO is set to zero the operator is still able to end the input early by keying a terminator, providing the terminal supports the necessary key stroke.

2.9 $$BELL ?????????????????? PIC 9 COMP

BELL SINCE LAST ACCEPT

2.10 $$BIF1 ?????????????????? PIC 9 COMP

1ST NUCLEUS BIT FLAG

2.11 $$BIF2 ?????????????????? PIC 9 COMP

2ND NUCLEUS BIT FLAG

2.12 $$BLNK ?????????????????? PIC 9 COMP

BLANK VIA EXECUTIVE FL

2.13 $$BLWR ?????????????????? PIC 9 COMP

-1=BASE LN WR'TN,0=NOT 1 = BASELINE JUST WRITTEN

2.14 $$CLMM ?????????????????? PIC X(40)

CALCULATOR WORK AREA

2.15 $$CLR ?????????????????? PIC 9 COMP

COLOUR MODE FLAG

2.16 $$CLS ?????????????????? PIC X(6)

CLEAR-SCREEN SEQUENCE

2.17 $$CODE ?????????????????? PIC 9(4) COMP

STOP/EXIT CODE

2.18 $$COND Exception Condition PIC 9(2) COMP R

The $$COND System Variable is set by any statement which can return an exception. When non-zero it is a positive integer used to identify the reason for an exception when the condition arises due to a number of different circumstances. For certain exceptions system variable $$RES will also be established to provide further information about the reason for the exception.

$$COND and $$RES are always used in conjunction with an ON EXCEPTION statement, and are explained in more detail as part of the description of that statement in the Global Development Language Manual.

2.19 $$CRES ??????????????? PIC 9(9) COMP ???

HOST ERROR CONDITION

2.20 $$DATE Today's Date PIC DATE R

Today's date is established in $$DATE from information supplied or confirmed by the operator at the start of a session. The date is stored as a computational number equal to:

10000*(year - 1900) + 100*(month of year) + day number

in order that dates can be compared using the standard arithmetic conditional statements. The date conversion routines allow you to convert a date in this PIC 9(6) COMP internal format to an 8 character external form, suitable for printing or displaying, and the DT-DY$ routine enables you to calculate the day number since 1900 from an internal format date field.

2.21 $$DECP ??????????????? PIC X ???

7.1.6 The Decimal Point Character, $$DECP
++
System variable $$DECP is a PIC X field which holds the character
currently being used as the decimal point. This is the character
set using $CUS, and may not be modified by the program.

2.22 $$DELA ??????????????? PIC 9(2) COMP ???

DELAY AFTER CLEAR

2.23 $$DOWK Day of the Week PIC 9(2) COMP R

This field contains the day number of the current system date (with 1 representing Sunday, 2 representing Monday etc.).

2.24 $$EOF ??????????????? PIC X ???

7.1.8 The End of Field Code, $$EOF
++++++++++++++++++++++++++++++++++
System variable $$EOF is a PIC X field which indicates which
terminator the operator keyed in order to complete a field input
by an ACCEPT, ACCEPT...LINE, CALL ACCE$ or ACDEF$... or CALL
PASS$... statement. The byte value of each terminator is in the
range #01 to #1F, and in consequence #40 is added to it in cal-
culating $$EOF, so that the relationship shown in table 4.1.8A is
established:-

| | | |

| | | |
| TERMINATOR | BYTE VALUE | $$EOF |
++++++++++ ++++++++++ +++++
| | | |

<CTRL A>	#01	#41 or "A"
<CTRL B>	#02	#42 or "B"
<CTRL C>	#03	#43 or "C"
<LINE FEED>	#0A	#4A or "J"
<CR>	#0D	#4D or "M"
<ESCAPE>	#1B	#5B or "["

Table 7.1.8A - Special Terminators
++++++++++++++++++++++++++++++++++

2.25 $$EPTO ??????????????? PIC SPT ???

6.7 Current Program Information, $$EPT, $$PGM and $$RUN

$$EPT addresses the entry point of the program last loaded, $$PGM contains its program-id, and $$RUN indicates whether the program has been loaded as a result of an operator request ($$RUN = 1) or because of a LOAD, CHAIN or EXEC statement executed by another program ($$RUN = 0). Use of these system variables is explained in more detail in chapter 4.

```
a).from the READY prompt: $$RUN=1
b).from keying <F1> to the menu: $$RUN=1
c).from a menu line: $$RUN=1
d).by an explicit EXEC from another frame: $$RUN=0
e).from the new "super system request" Run menu: $$RUN=??
f).as a Speedbase forward frame: $$RUN=0
```

2.26 $$ESC ??????????????? PIC 9 COMP ???

7.1.3 The Escape Key Suppress Flag, $$ESC
++

Normally, when the operator keys ESCAPE key in response to a prompt, control is immediately returned to the monitor or to a menu. You may, however, wish to prevent this happening in certain applications where an inadvertent return to the monitor could cause difficulties, such as files becoming inconsistent, or updates being lost. The System Manager provides the system variable $$ESC to allow you to inhibit the normal ESCAPE key handling.

$$ESC is a PIC 9 COMP field which controls the way in which the ESCAPE key is handled. On entry to an application $$ESC is zero,
++++
and this setting causes <ESCAPE> to return control to the monitor

------
in the normal way.

You may set $$ESC to 1 to cause the normal processing to be
suppressed. In this case when <ESCAPE> is keyed it is treated
------
exactly as though the operator had keyed <CR>. Thus, when $$ESC
--
is 1 it is no longer possible for the operator to obtain a ready
prompt by keying <ESCAPE> in response to an application prompt.

Note that $$ESC is automatically reset to zero at end of job in
order to restore normal ESCAPE key handling.

2.27 $$FED ?????????????? ??? PIC 9 COMP

7.1.11 The Field Editing Flag, $$FED
++++++++++++++++++++++++++++++++++++
The system variable $$FED is a PIC 9 COMP field used to indicate
whether field editing is enabled. It is set to 1 if field
editing is enabled and to 0 if not. It is always set to 0 in
pre-V6.0 systems.

2.28 $$FESC ?????????????? ??? PIC 9 COMP

7.1.12 The Field Editing Start Character Flag, $$FESC
++
The Field Editing Start Character Flag is a PIC 9(2) COMP field
used to control the enabling of field editing in ACCEPT
statements. The following values are permitted:-

0 Field editing is not entered automatically, but may be
enabled during the ACCEPT as described in the Global
Operating manual (section 3.1).

-1 Field editing is not permitted. (Cursor editing keys
are returned as termination characters to ACCEPT.)

n (Where n is greater than 0) This causes field editing
/ /
to be enabled automatically, and the cursor to be
positioned under the nth character of the ACCEPT
/
field. Setting n to 1 means that the ACCEPT starts in
/
field editing mode.

Special Value
+++++++++++++

-2 This causes the cursor to be placed at the first
trailing space in the field.

On completion of the ACCEPT the flag is reset to 0.

2.29 $$FID ??????????????? ??? PIC X(8)

LAST FILE ACCESSED

2.30 $$FINX ??????????????? ??? PIC X(8)

LAST INDEX ACCESSED

2.31 $$FOP ??????????????? ??? PIC 9 COMP

LAST FILE OPERATION

2.32 $$FI Facility Code I PIC 9 COMP

This field is reserved for internal use by Global System Manager.

2.33 $$FJ Facility Code J PIC 9 COMP

This field is reserved for internal use by Global System Manager.

2.34 $$FK Facility Code K PIC 9 COMP

This field is reserved for internal use by Global System Manager.

2.35 $$FL Facility Code L PIC 9 COMP

This field is reserved for internal use by Global System Manager.

2.36 $$FLSH ??????????????? ??? PIC 9(2) COMP

7.1.4 The Type-ahead Buffer Flush Flag, $$FLSH
+++
Characters keyed at the console are accumulated in a type-ahead
buffer until requested by an ACCEPT statement or equivalent. An
accept operation normally inputs characters from the buffer and
then, if a terminator is not present, takes them directly from
the terminal as the operator keys them. By setting the PIC 9
COMP system variable $$FLSH to 1 before an accept takes place you
can cause any information currently in the buffer to be "flushed"
so that the new input comes directly from the keyboard. The
flush flag only applies to a single accept operation: if you wish
to clear the buffer again you must reset the flag to 1.

The BELL statement, which you are recommended to use when an
error is detected, sets the buffer flush flag to 1. The assump-
tion is that any information in the buffer is useless, since you
would not be expecting the error. Note that the System Manager
issues a BELL statement, and hence clears the buffer, whenever it

detects an error, such as an invalid format number being keyed,
or an irrecoverable I/O error.

2.37 $$GUI ??????????????? PIC 9 COMP ???
GUI FLAG

2.38 $$GUI2 ??????????????? PIC 9 COMP ???
GUI 2 FLAG

2.39 $$GWW ??????????????? PIC 9 COMP ???
GWW FLAG

2.40 $$GX ??????????????? PIC 9 COMP ???
GUI 2 FLAG

2.41 $$HBK ??????????????? PIC X(2) ???
START BOOK FOR HELP

2.42 $$HCLR ??????????????? PIC 9 COMP ???
CLEAR HELP TEXT FLAG

2.43 $$HFIL ??????????????? PIC X(8) ???
HELP FILE NAME

2.44 $$HPTR ??????????????? PIC SPT ???
SPECIAL HELP REQUEST P

2.45 $$HUN ??????????????? PIC X(3) ???
HELP FILE UNIT

2.46 $$INDE ??????????????? PIC SPT ???

6.32 Library Index Pointer, $$INDE

The library index pointer, $$INDE, is a PIC PTR system variable, used in storage management (see chapter 8), which addresses the first byte to be occupied by the 1134-byte library index record when an overlay is next loaded from a program library. At the beginning of each job Global System Manager will have set $$INDE to address the top 1134 bytes of the user area. A job which acquires a work space and then continues to require overlays, or which uses an unconventional overlay scheme, should set $$INDE to address a buffer within the transient area to be used for its overlays in order to prevent the index record corrupting any other part of the occupied user area as explained in 8.1.4.

Note that the LOAD$ and UNLO$ system routines modify $$INDE if they are used to introduce or remove a relocatable program.

To allow you to overcome this type of problem Global System Manager provides the system variable, $$INDE, a pointer that you can modify in order to control where the library index record is to be placed (see section 6.2.9). You only require to use $$INDE in this way when the normal handling might cause the program area or the work space to be corrupted. You set the pointer to address a buffer within the area to which your overlays are to be loaded. Then the record is simply overwritten by an incoming module and it does not corrupt any part of the remainder of the program. (You may also require to set $$INDE to direct the library index record into a data area within your program, so that you can inspect it to see which programs are actually present in a library: this use of $$INDE is described in 7.1.)

You should note that whenever a program acquires the whole of memory, and then requires to load an overlay, normal index record handling will **always** corrupt part of the work space, and so it is essential to set $$INDE to address the overlay area itself. Programs that do not acquire work space, but adopt unconventional overlay strategies not conforming to the deepest-level, highest-address rule, may also need to manipulate $$INDE. In general, such programs should be avoided.

There is an example of the use of $$INDE, in conjunction with some of the other storage management functions provided by FREE$, in section 8.2. In addition, the programming notes on the use of the sort in the Global Development File Management Manual illustrate how $$INDE should be used when employing input and output processing overlays.

2.47 $$INT R/W Disable System Request Flag PIC 9 COMP

This field that can be used to disable system requests during crucial code paths within a program. A value of 1 disables system requests and a value of 0 re-enables them. This flag is used by $CUS to disable system requests, so its value must be preserved by your program.

2.48 $$ISOG ??? ??????????????? PIC 9 COMP

EXTRA ISO CHARS

2.49 $$JCL ??? ??????????????? PIC 9 COMP

7.1.6 The Job Management Flag, $$JCL
++++++++++++++++++++++++++++++++++++
When an application program runs under job management, input for
ACCEPT, ACCEPT...LINE, CALL ACCE$ or ACDEF$, CALL PASS$ and MAPIN
statements is normally supplied from a previously prepared dia-
logue table held in memory. The operator is no longer prompted,
and instead the necessary responses are supplied from the table.
There is also an option of job management which allows you to

suppress the output of information by DISPLAY, DISPLAY...LINE, CALL VIDEO$, and CALL DISP$, so that programs can be run in a "quiet" mode, with no console displays appearing.

System variable $$JCL is a PIC 9(2) COMP field which you can set to 1 to temporarily bypass job management, so that the next accept operation takes input directly from the operator, ignoring the stored dialogue. Any displays that take place while $$JCL is 1 are always output to the screen, irrespective of whether the general dialogue has been suppressed. $$JCL is reset to a value other than 1 by any accept operation, so that it only modifies one group of displays terminated by an accept at a time.

You can set $$JCL to 0 to cause the program to be terminated with an error if it is running under job management. Any displays that take place after setting $$JCL to 0 are always output to the screen, and the program will be terminated on the next accept operation. In practice, you will normally use the BELL statement which has the same effect, since it causes $$JCL to be set to zero.

This description of $$JCL is included in this chapter on advanced console management for completeness only, because the system variable affects the operation of a number of console I/O statements. You should refer to the Global Cobol Job Management Manual for a fuller explanation of the coding of application programs to run under job management.

2.50 $$L ??? ??????????????? PIC 9(4) COMP

LENGTH OF CURSOR POS'N

2.51 $$LANG ??? ??????????????? PIC X(3)

LANGUAGE IDENTIFIER

2.52 $$LENG ??? ??????????????? PIC 9(2) COMP

7.1.7 The Input Field Length, $$LENG
++++++++++++++++++++++++++++++++++++

Following an ACCEPT statement, or equivalent, the PIC 9(2) COMP system variable $$LENG is set to contain the length in bytes of the input field actually keyed by the operator. This may be smaller, of course, than the input variable supplied by your program. the System Manager automatically packs a short character input with rightmost blanks, so you can use $$LENG to determine the significant part of the input without the need to write special code to detect and count any trailing blanks.

The following statements set $$LENG as described above:-

ACCEPT CALL ACCE$, ACDEF$...
ACCEPT...LINE CALL PASS$...

2.53 $$LEVN ??????????????? PIC X ???

6.17 The Global System Manager Level Number Indicator, $$LEVN

$$LEVN is a PIC X field which holds the level number of Global System Manager being used. The field is only available in the V5.2 compiler, and only exists in V5.1, or later, Global System Manager. The value held in the field will be one of the following:

1	Not used at present
2	SBOS
3	CBOS
4	MBOS/PC
5	MBOS
6	BOS/NET (obsolete)
7	BOS/LAN
8	BOS/LAN+
9	BOS/XLAN

2.54 $$LIB Attached Library Name PIC X(8) R/W

The $$LIB System Variable field contains the library-id of the currently attached user library. If there is no library attached it will contain "P.".

This variable allows a subroutine which needs to attach a library to save the name of the caller's library on entry, and re-attach this library on exit.

For example, such a subroutine might contain the following statements:

```
MOVE $$LIB TO SAVE-LIB           * SAVE CALLER'S LIBRARY
LOAD "P.SA"                      * ATTACH OWN LIBRARY
.........
......... (other processing)
........
LOAD SAVE-LIB                    * RESTORE CALLER'S LIBRARY
EXIT
```

Note that if there were no library attached on entry, $$LIB would contain "P.", and hence the final LOAD statement would simply detach the subroutine's library, restoring the initial status.

2.55 $$LINE ??????????????? PIC 9(4) COMP ???

6.8 The Console Dimensions, $$LINE, $$WIDE and $$RWID

The system variables $$LINE and $$WIDE are two PIC 9(4) COMP fields which indicate, respectively, the depth of the console in lines and its width in characters. Typical values with current technology are $$LINE = 24 and $$WIDE = 80. However, Global System Manager supports some consoles with much larger dimensions, for example $$LINE = 55 and $$WIDE = 132.

Global System Manager will operate successfully on consoles with $$WIDE = 40 or greater, and $$LINE = 20 or greater. However most Global Software requires a console at least 79 characters wide and 24 lines deep. (Note that a number of consoles which are nominally 80 characters wide handle the 80th character position specially, and hence must be treated as 79 characters wide by Global System Manager.)

Portable programs should always check the dimensions of the console to ensure that it is sufficiently large, particularly if they use formatted screens. A program may be able to adjust the depth or width of its displays to suit the console. For example, programs writing long reports to the console in teletype mode should use $$LINE to determine when to output a paging prompt such as:

Key P to Page, A to Accept and continue:

Normally ($$LINE)-1 lines of report will be output, followed by the prompt, which will then appear on the base line to give the operator the chance to read the information that has just been displayed. Usually, as in the example, you will allow the operator the option of either asking for the next "page" of output (with automatic "wrap around" to the first page following the last), or requesting that the program continue once enough of the report has been read.

Note that even if $$WIDE is greater than 80 the maximum amount of information you can input or output using a single ACCEPT or DISPLAY statement is still restricted to 80 bytes.

$$RWID, the physical terminal's maximum width (i.e. the widest available screen display), is a PIC 9(4) COMP field which indicates the **physical** terminal's maximum width (as opposed to $$WIDE, which indicates the current **logical** width).

2.56 $$LNID R Local Node Indicator PIC X

$$LNID, the local node indicator, contains the computer (or system) identification letter on a networked or separated system. If your computer (or system) is a file server it will be in the range A - Z. For non file server computers (or systems) it can take other values (not necessarily printable). Values in the range #40 - #FF correspond to node numbers 0 - 191, and #01 - #3F correspond to 193 - 255. For non-network systems $$LNID is set to #00.

2.57 $$MACH ??? ?????????????? PIC X(15)

2.5.4 The Machine Name, $$MACH
++++++++++++++++++++++++++++++
System variable $$MACH is a PIC X(15) field containing the
machine name of your BOS computer. This is displayed together
with the architecture code on the system information report
output by the $S command

2.58 $$MCOD ??? ?????????????? PIC X(2)

2.5.3 The Machine Family Code, $$MCOD
+++++++++++++++++++++++++++++++++++++
System variable $$MCOD is a PIC X(2) field containing a two

character code identifying the family of machines on which the software is running. If you provide, for a variety of BOS computers, functionally identical machine code routines which access peripherals, and can hence differ between two machines with the same architecture code, then you should incorporate the machine family code in the file name so that the correct program can be identified and loaded. For example the serial port handler used by $REMOTE is file %.amR where am is the machine family code. The code is displayed on the system information report produced by $S.

2.59 $$MEDF ?????????????? PIC 9 COMP ???

6.33 Default Menu Entry, $$MEDF

The default menu entry, $$MEDF, is PIC 9 COMP system variable. Its use is described in section 7.5.6.
The system variable $$MEDF contains the menu entry selected by the Global System Manager menu if the menu is CHAINed to. It is used by $AUTH and may be useful in other authorization programs.

2.60 $$MNID Master Node Indicator PIC X R

$$MNID, the master node indicator, identifies the master computer (or system) on a network or separated system configuration by means of its computer or system identification letter, A - Z. It is set to #00 for other systems, and can thus be used as a test for whether you are on a network configuration.

2.61 $$MU Multi-User Flag PIC 9 COMP R

System variable $$MU is set to zero under SBOS, and set non-zero in all other levels of Global System Manager. Applications can test $$MU in order to bypass special shared file handling procedures when operating in a single-user environment. For example, there is no need for a single-user program to LOCK the files it requires to update.

The following table shows how $$MU indicates the exact type of system under which your program is running:

$$MU	*Type of system*
0	Single-user, single processor (SBOS)
1	Multi-user, single processor (CBOS or MBOS)
2	Single-user processor in a networking environment
3	**Multi-user processor in a networking environment**

This field is mainly for historical use as it will always contain 3 for all 32-bit configurations.

2.62 $$NCYR Century Start Year PIC 9 COMP R

This field contains the year from which the century is said to start for short date purposes on the system. The century start year can be customised using $CUS (see the Global Utilities Manual).

For example, if the century start year is set to 50 then the short date, 01/01/11 will be set to refer to 01/01/2011 and 01/01/60 to 101/01/1960 by the date conversion routines.

2.63 $$NULL ??????????????? PIC SPT ???

NULL POINTER

2.64 $$PACC ??????????????? PIC SPT ???

POINTER TO ACCEPT AREA

2.65 $$PAGE Standard Printer Page Size PIC 9(4) COMP R

$$PAGE contains the number of lines per page for the **standard** printers used by a configuration. If printers with two or more different page sizes are supported a decision must be made at installation time as to which size is to be the standard. Normally 66 lines per page is adopted. If any other value is to be established it must be set up by running $CUS, selecting Configuration customisation, and altering Standard Page Size.

Portable application programs writing reports to standard stationery should refer to $$PAGE to determine when it is necessary to advance the stationery to a new page. The page size of special stationery is under program control, as explained in the section on print files in section 1.5 of the Global Development File Management Manual.

2.66 $$PAR Partition Number Indicator PIC 9 COMP R

This field contains the partition number currently being used. It is set to 1 or 2 for foreground or background, or 1 - 9 (usually 1 - 4) for concurrent partitions. On single-user processors it is set to 1.

2.67 $$PAUS ??????????????? PIC 9 COMP ???

1 = PAUSE REQUIRED * IE BASELINE , PROMPT AT BASE LIN

2.68 $$PGM ??????????????? PIC X(8) ???

6.7 Current Program Information, $$EPT, $$PGM and $$RUN

$$EPT addresses the entry point of the program last loaded, $$PGM contains its program-id, and $$RUN indicates whether the program has been loaded as a result of an operator request ($$RUN = 1) or because of a LOAD, CHAIN or EXEC statement executed by another program ($$RUN = 0). Use of these system variables is explained in more detail in chapter 4.

2.69 $$PM ??????????????? PIC 9 COMP ???

6.14 The Presentation Manager Flag, $$PM

This is a PIC 9(2) COMP field which is set to 1 if the system the program is running on is a Global System Manager - Presentation Manager system and set to 0 if the system is just a Global System Manager system. This flag is not available on pre-V8.1 systems.

2.70 $$PRIN ??? ??????????????? PIC 9(4) COMP

6.10 The Standard Printer Line Width, $$PRIN

$$PRIN contains the line width in characters of the **standard** printers used by a configuration. If two or more different line widths are in use a decision must be made at installation time as to which is to be the standard. Normally 132 characters per line is adopted. If any other value is to be established it must be set up by running $CUS, selecting Printer customisation, and altering Printer Page Size.

By using $$PRIN together with $$PAGE it is possible to develop applications which are able to adapt themselves to a variety of different paper sizes.

2.71 $$PROM ??? ??????????????? PIC X

7.1.5 The Prompt Character, $$PROM
+++++++++++++++++++++++++++++++++++
System variable $$PROM is a PIC X field which you may set to
change the prompt character used in the next accept operation.
If you set it to LOW-VALUES no prompt character at all is dis-
played. Normally $$PROM contains a colon character, and it will
be restored to this value following the accept. Thus you must
reset $$PROM every time you wish to produce a special prompt not
identified by the usual colon character.

2.72 $$PVOL Program Volume-id PIC X(6) W

When programs are loaded from exchangeable disks or diskettes you can request Global System Manager to check that the correct volume is online, and to prompt the operator to mount it if it is not. To do this you set $$PVOL to the six character volume-id used to uniquely identity the program volume just before executing the LOAD, EXEC, CHAIN or RUN statement responsible for bringing the program into memory. Once the statement returns control $$PVOL will be reset automatically to its initial state, low-values, to prevent Global System Manager making further checks until you set up another volume-id.

2.73 $$REV ??? ??????????????? PIC X

GSM REVISION LETTER

2.74 $$REL ??? ??????????????? PIC 9 COMP

6.31 The Release Program Area Flag, $$REL

The size of the program area is normally only ever increased by the loader. By setting the PIC 9 COMP system variable $$REL to 1 before executing a LOAD, CHAIN or EXEC statement you can cause the loader to set the program area size to the end of the program it is loading, provided the work space is not in use. If the work space is in use setting $$REL has no effect: you must use FREE$ instead.

2.75 $$RESR Result Code PIC X

For certain exceptions the $$RES System Variable will be established to provide further information about the reason for the exception.

$$COND and $$RES are always used in conjunction with an ON EXCEPTION statement, and are explained in more detail as part of the description of that statement in the Global Development Language Manual.

$$RES is also used for special I/O error handling, which is described in the Global Development File Management Manual.

2.76 $$RUN ??????????????? PIC 9 COMP ???

6.7 Current Program Information, $$EPT, $$PGM and $$RUN

$$EPT addresses the entry point of the program last loaded, $$PGM contains its program-id, and $$RUN indicates whether the program has been loaded as a result of an operator request ($$RUN = 1) or because of a LOAD, CHAIN or EXEC statement executed by another program ($$RUN = 0). Use of these system variables is explained in more detail in chapter 4.

2.77 $$RWID ??????????????? PIC 9(4) COMP ???

6.8 The Console Dimensions, $$LINE, $$WIDE and $$RWID

The system variables $$LINE and $$WIDE are two PIC 9(4) COMP fields which indicate, respectively, the depth of the console in lines and its width in characters. Typical values with current technology are $$LINE = 24 and $$WIDE = 80. However, Global System Manager supports some consoles with much larger dimensions, for example $$LINE = 55 and $$WIDE = 132.

Global System Manager will operate successfully on consoles with $$WIDE = 40 or greater, and $$LINE = 20 or greater. However most Global Software requires a console at least 79 characters wide and 24 lines deep. (Note that a number of consoles which are nominally 80 characters wide handle the 80th character position specially, and hence must be treated as 79 characters wide by Global System Manager.)

Portable programs should always check the dimensions of the console to ensure that it is sufficiently large, particularly if they use formatted screens. A program may be able to adjust the depth or width of its displays to suit the console. For example, programs writing long reports to the console in teletype mode should use $$LINE to determine when to output a paging prompt such as:

    Key P to Page, A to Accept and continue:

Normally ($$LINE)-1 lines of report will be output, followed by the prompt, which will then appear on the base line to give the operator the chance to read the information that has just been displayed. Usually, as in the example, you will allow the operator the option of either asking for the next "page" of output (with automatic "wrap around" to the first page following the last), or requesting that the program continue once enough of the report has been read.

Note that even if $$WIDE is greater than 80 the maximum amount of information you can input or output using a single ACCEPT or DISPLAY statement is still restricted to 80 bytes.

$$RWID, the physical terminal's maximum width (i.e. the widest available screen display), is a PIC 9(4) COMP field which indicates the **physical** terminal's maximum width (as opposed to $$WIDE, which indicates the current **logical** width).

2.78 $$SEED ?????????????? ??? PIC 9(9) COMP

6.11 The Seed for the Random Number Generator, $$SEED

System variable $$SEED is a PIC 9(9) COMP item that you may set to cause the RAND$ system routine to generate a repeatable sequence of pseudo-random numbers, as explained in section 3.1.

2.79 $$SER ?????????????? ??? PIC 9(6) COMP

SERIAL NUMBER

2.80 $$SNAM System Name PIC X(30) R

This field contains the system name set up using $CUS and is displayed on subsequent Global System Manager menus.

2.81 $$SRKT ?????????????? ??? PIC X(10)

SYSTEM REQUEST KEY

2.82 $$SVSR ?????????????? ??? PIC X(8)

6.30 The Supervisor Program Name, $$SVSR

The system variable $$SVSR is a PIC X(8) field in which you can establish the name of a supervisor program. Its use is described in section 7.6.

7.6.1 The Supervisor Program Name, $$SVSR

The system variable $$SVSR is a PIC X(8) field in which you can establish the name of a supervisor program. If the field is blank, as it is initially unless customised as explained below, then there is no supervisor program, and the READY prompt will be displayed. $$SVSR is reset to spaces whenever the supervisor program is invoked, so the supervisor program must re-establish its program name in $$SVSR each time it is executed, unless it wishes to relinquish control, when it should execute a STOP RUN statement with $$SVSR set to spaces, to cause the READY prompt to be displayed.

When the supervisor program is invoked by Global System Manager it is invoked with logging off allowed. If you do not wish the user to be allowed to log off while in the supervisor program, the supervisor program must immediately call the NLOGF$ routine.

2.83 $$SYSM ?????????????? ??? PIC 9 COMP

6.13 The Global System Manager operating System Flag, $$SYSM

This PIC 9 COMP flag identifies the host operating system on which Global System Manager is running (0 for Global System Manager (BOS); 1 for Global System Manager (Unix); 2 for Global

System Manager (MS-DOS and Windows); 3 for Global System Manager (Novell NetWare); 4 for Global System Manager (Windows) other values are reserved for future use). This flag is not available on pre-V8.0 Global System Manager.

2.84 $$TER ????????????????? PIC S9(2) COMP

TERMINATING CHARACTER

2.85 $$TERM ????????????????? PIC X(6)

7.1.1 The Terminal Code, $$TERM
++++++++++++++++++++++++++++++
System variable $$TERM is a PIC X(6) field containing a unique
code identifying the terminal. This is the value which the
operator keys in response to the terminal prompt during sign-on.
It identifies the terminal attribute program (TAP) used to supply
the System Manager with detailed information about the device:
the TAP program-id is of the form:-

$.code
////

where code is the contents of $$TERM.
////

Applications which use hardware-dependent features of certain
terminals can examine $$TERM to determine which, if any, of the
range of terminals they support is currently being employed as
the console, and adapt themselves accordingly. $$TERM should
not, of course, be used in this way by portable applications
which require to run on a wide variety of different devices.

2.86 $$TRAN ????????????????? PIC 9 COMP

TRANSLATION FLAG

2.87 $$TYPE ????????????????? PIC 9(2) COMP

TERMINAL TYPE

2.88 $$ULEV Maximum Number of Users PIC 9(4) COMP R

This field contains the maximum number of users for which this system can be configured. The maximum number of users depends on the level number of the system ordered.

2.89 $$USA ????????????????? PIC 9 COMP

6.18 The American Processing Flag, $$USA

$$USA determines the way in which the Global System Manager command programs, together with the DATE$ system routine, construct the external form of the date to be used on listings

and displays. When Global System Manager is distributed $$USA is set to 0, indicating that European processing is in force, so the external date appears as the 8 characters:

dd/mm/yy

where *dd*, *mm* and *yy* are the day number, month number, and year of century, with leading zeros as necessary.

When Global System Manager is installed you can choose either European or American date format, thus setting the value in $$USA to 0 or 1 respectively. If the American format is chosen the external form of the date appears as:

mm/dd/yy

The format used by an installed system can be altered at any time by running $CUS, selecting Configuration customisation, and altering Date Format.

$$USA can also be used to select other application-dependent American or European options at run-time, thus avoiding the need for two versions of a program.

2.?0 $$VERS Version Number Indicator PIC 9(2) COMP R

The $$VERS System Variable indicates the version of Global System Manager being used. It is set to 0 for V5.0; 1 for V5.1; 2 for V5.2; 3 for V6.0; 4 for V6.1; 5 for V6.2; 6 for V7.0; 7 for V8.0; 8 for V8.1 and, potentially, 9 onwards for future releases. It should be used first to determine what further checks on the task environment need to be made by your program.

2.91 $$WIDE ?????????????? PIC 9(4) COMP ???

6.8 The Console Dimensions, $$LINE, $$WIDE and $$RWID

The system variables $$LINE and $$WIDE are two PIC 9(4) COMP fields which indicate, respectively, the depth of the console in lines and its width in characters. Typical values with current technology are $$LINE = 24 and $$WIDE = 80. However, Global System Manager supports some consoles with much larger dimensions, for example $$LINE = 55 and $$WIDE = 132.

Global System Manager will operate successfully on consoles with $$WIDE = 40 or greater, and $$LINE = 20 or greater. However most Global Software requires a console at least 79 characters wide and 24 lines deep. (Note that a number of consoles which are nominally 80 characters wide handle the 80th character position specially, and hence must be treated as 79 characters wide by Global System Manager.)

Portable programs should always check the dimensions of the console to ensure that it is sufficiently large, particularly if they use formatted screens. A program may be able to adjust the depth or width of its displays to suit the console. For example, programs writing long reports to the console in teletype mode should use $$LINE to determine when to output a paging prompt such as:

Key P to Page, A to Accept and continue:

Normally ($$LINE)-1 lines of report will be output, followed by the prompt, which will then appear on the base line to give the operator the chance to read the information that has just been displayed. Usually, as in the example, you will allow the operator the option of either asking for the next "page" of output (with automatic "wrap around" to the first page following the last), or requesting that the program continue once enough of the report has been read.

Note that even if $$WIDE is greater than 80 the maximum amount of information you can input or output using a single ACCEPT or DISPLAY statement is still restricted to 80 bytes.

$$RWID, the physical terminal's maximum width (i.e. the widest available screen display), is a PIC 9(4) COMP field which indicates the **physical** terminal's maximum width (as opposed to $$WIDE, which indicates the current **logical** width).

2.92 $$X ?????????????? ??? PIC 9(4) COMP

X COORDINATE (COL#)

2.93 $$XY ?????????????? ??? PIC X(4)

XY SEQUENCE

2.94 $$Y ?????????????? ??? PIC 9(4) COMP

Y COORD (LINE#)

2.95 $$ZERO ?????????????? ??? PIC SPT

PTR TO ABSOLUTE ZERO

2.96 $$AUTH ?????????????? ??? PIC X

6.29 The Authorization Code, $$AUTH

The system variable $$AUTH is a PIC X field containing an alphabetic value in the range A to Z, inclusive. It is used when creating an optional authorization vetting program to establish the code for each operation allowed to access your system, as described in section 7.5. of this manual.

7.5.4 The Authorization Routine, AUTH$

The authorization routine must be invoked once, and once only, for each operator processed by your vetting program. You code a CALL of the form:

```
        CALL AUTH$ USING code
```

The *code* is the name of a PIC X field, or a character literal. If you have decided to grant access you must supply an alphabetic character, between A and Z inclusive, which will be established in the system variable $$AUTH as a result of the call. If you wish to refuse access, simply provide a code which is less than ASCII A in collating sequence. For example:

```
        CALL AUTH$ USING "?"                    * REFUSE ACCESS
```

2.97 $$OPID Operator-id PIC X(4) R

The $$OPID System Variable contains the operator-id supplied at the start of a Global session. Global System Manager checks that the operator-id you supply to the prompt is not the same as any other operator-id currently signed on. The same operator-id is associated with all jobs initiated by an operator during a session, and therefore competing foreground, background and concurrent jobs started at the same screens have the same operator-id.

2.98 $$SCNN ?????????????????? PIC 9(2) COMP

6.26 The Screen Number, $$SCNN

$$SCNN is a PIC 9(2) COMP field identifying the partition by a unique screen number (between 1 and 255). This variable is used in various System Routines such as MSG$.

2.99 $$SUSP ?????????????????? PIC 9 COMP

2.5.5 The Suspend Flag, $$SUSP

++++++++++++++++++++++++++++++

System variable $$SUSP is a PIC S9 COMP field which serves as a flag indicating whether or not the current program has suspended itself for a time period by executing a statement such as:-

```
SUSPEND 3600            * SUSPEND FOR ONE HOUR
```

$$SUSP will be negative if the program has suspended itself in this way, positive or zero otherwise. An interrupt service routine on the system stack can end a user program's suspension by setting $$SUSP to zero.

To take a concrete example, suppose a task is used to control a communications link in conjunction with an interrupt service program, INTSER. The task then loads INTSER onto the system stack, and passes it the address of its suspend flag, by executing statements such as:-

```
LOAD "INTSER  " "S"
ON EXCEPTION STOP WITH 1
CALL $$EPT USING $$SUSP
```

INTSER will use this initial invocation to remember the machine address of the $$SUSP byte and activate its interrupt service routine by overwriting the appropriate vector or interrupt address field. It will then return control to the BOS Cobol task. As soon as this finds it has no work to do the task will suspend itself for a long period, say one hour, by a:-

```
SUSPEND 3600
```

The task will then be "awoken" before the end of the time period in two circumstances: by a companion task (running in a separate partition) when there is an outgoing message to transmit, and by the interrupt service routine when an incoming message is received. In the first case the companion task cancels the suspension by invoking the START$ routine described in the BOS Cobol System Subroutines Manual. In the second case INTSER ends the suspension by setting the $$SUSP flag, passed to it as an initial parameter, to zero.

2.100 $$TASK ????????????????? PIC S9 COMP

6.20 The Task Environment Indicator, $$TASK

System variable $$TASK is a PIC S9 COMP field which indicates if your program is currently running as a background job, a normal job, or a foreground job on V5.2 or earlier systems:

$$TASK	*Task environment*
-1	background job
0	normal job
+1	foreground job

A program is said to be running in a normal job environment (with $$TASK = 0) when it is the only job working for a particular operator, as is always the case under SBOS, or if it is running in a concurrent partition. (See $$PAR, used to distinguish different partition numbers.) In the multi-user environment under V5.2 or earlier, however, it is possible for an operator to run a background and a foreground job simultaneously and in this case these jobs will have $$TASK values of -1 and +1 respectively.

You may find it useful to test $$TASK and suppress non-vital screen messages when a job runs in the background, in order to prevent it being unnecessarily suspended. For example, the Global Cobol compiler avoids displaying its first pass error messages when it is executing in the background. Note that you can conveniently suppress all dialogue from a program, without modifying it in any way, by running it under job management and using the SUPPRESS option.

Under V6.0 Global System Manager (and later) $$TASK **always** has a value of 0.

2.101 $$USER User Number PIC 9(2) COMP R

The $$USER System Variable contains the current user number. A unique user number is allocated for each partition when the system is initiated. By default, the User Number ranges from 1 to 99. However, this limit can be extended so that User Number can range from 1 to 250. Code that manipulates $$USER should be aware of the revised upper limit.

[GIVE SOME EXAMPLES OF DODGY CODING TECHNIQUES]

Note that under single-user systems $$USER is always 1. Note also that $$USER should not be relied upon to be unique across a network, since the user number refers to users of a single computer.

2.102 $$WAIT ?????????????? PIC S9(2) COMP ???

WAIT

3. System Variables in the 32-bit Address Space

The following System Variables exist in a 32-bit Memory Page and are not accessible by 16-bit applications.

3.1 $$3000 Global 3000 Work Area PIC X(1024)

This work area is reserved for use by Global 3000.

3.2 $$AR32 Extended Application Work Area PIC X(256) R/W

The Extended Application Work Area consists of 256 bytes within the system area which are available for application use. The bytes are set to binary zero at the start of a session. However, once this area has been erased in this way Global System Manager never refers to it again. Thereafter the 256 bytes can be used for any application purpose whatsoever. The work area allows you to pass a small amount of information between programs very conveniently.

To refer to the Extended Application Work Area you simply redefine $$AR32. For example:

```
01    AP REDEFINES $$AR32
  03  APINIT      PIC 9 COMP        * INITIATION FLAG
  03  APPRIV      PIC 9(2) COMP     * OPERATOR PRIVILEGE LEVEL
  03  APNUMB      PIC 9(4) COMP     * OPERATOR NUMBER
```

In practice, of course, you would probably place the entire definition of the Extended Application Work Area in a copy book, so that each programmer need only code, for example:

```
COPY AP
```

The 32-bit only Extended Application Work Area, $$AR32, is completely separate from the 16-bit compatible Application Work Area, $$AREA (see section 2.??). Thus, the total work area available for 32-bit applications is 272 bytes.

3.3 $$EOJ Pointer to End Of Job Routine PIC PTR W

EOJ ROUTN BIT PROGRAM

3.4 $$GXEX GX Operation Exception Code PIC 9(4) COMP ???

CALL DO RETURN CODE

3.5 $$GXFN Last GX Operation Code PIC 9(4) COMP ???

LAST CALL DO FUNCTION

3.6 $$GXM BA$GXM Work Area PIC X(16384)

This work area is reserved for use by the BA$GXM DLM.

3.7 $$NEPT Entry Point of 32-bit Program PIC PTR ???

EPT FOR 32 BIT PROGRAM

3.8 $$NEXE Entry Point For B$EXE PIC PTR ???

EPT FOR B$EXE

3.9 $$POPA Pop Menu Handler Work Field PIC 9 COMP

This work field is reserved for use by the Pop Menu Handler.

3.10 $$DEBUG System Wide Debug Flag PIC 9(2) COMP R/W

This field is available for use as a general-purpose Debug Flag. The meaning of this flag is application dependent although the following values are advised:

0	Debug mode disabled
1	Application specific debug level-1
>1	All values higher than 1 are reserved for future use

System variables are elementary data items which are conceptually declared automatically in the data division of every compilation. The variables do not actually appear as data definitions; neither do they occupy working storage. They are located within a permanently available region known as the **System Area**, which is used to communicate parameter information between Global System Manager and an application program. They can be referenced from procedure division statements whenever a level 77 item with the same picture clause would be valid.

This chapter covers the most commonly used system variables. Additional system variables are defined in the following chapter and in the Global Development Job Management Manual, the Global Development Screen Presentation Manual, the Global Development File Management Manual and the Global Development Assembler Interface Manual.

The following two points should be considered when using system variables:

- Unless it is explicitly stated to the contrary, a system variable should be read-only as far as the application program is concerned;

- System variables may be redefined, but the redefinition must appear in the LINKAGE SECTION.