DOMAIN System Call Reference
(Volume 1, ACLM - GPR)
Update 1

Order No. 008856
Revision 00
Software Release 9.2

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

Printed in U.S.A.

First Printing: February, 1986

This document was produced using the SCRIBE document preparation system. (SCRIBE is a registered trademark of Unilogic, Ltd.)

Preface

This manual is part of a two-volume set that describes the DOMAIN® system calls. Each volume consists of a section that introduces the system calls followed by sections that describe a separate operating system manager (e.g., the process manager, stream manager, and variable formatting package). The sections that describe the managers are in **alphabetical order by manager name** and consist of a description of the data types used by the manager, the syntax of the manager's programming calls, and the error messages generated by the manager.

For easy organization, we have numbered the pages of this two volume reference set by system manager. For example, the third page in the ACLM section is page ACLM-3.

Volume 1 includes descriptions of the following managers:

ACLM
CAL
EC2
ERROR
GM
GMF
GPR

Volume 2 includes descriptions of the following managers:

IPC	PROC1
MBX	PROC2
MS	RWS
MTS	SIO
MUTEX	SMD
NAME	STREAM
PAD	TIME
PBUFS	TONE
PFM	TPAD
PGM	VEC
PM	VFMT

You should use this manual with the programming handbooks listed under Related Documents. These programming handbooks give detailed instructions about using these programming calls.

Audience

This manual is intended for programmers who are writing application programs using DOMAIN system calls. Readers of this manual should be familiar with FORTRAN, Pascal, or C and the operating system as described in the *DOMAIN System User's Guide*. This manual is not intended as a tutorial document, but as a reference for programmers who need to use operating system services.

Related Documents

The *Programming With General System Calls* handbook, order no. 005506, documents how to write programs that use standard DOMAIN system calls including the ACLM, CAL, EC2, ERROR, MTS, NAME, PAD, PBUFS, PFM, PGM, PM, PROC1, PROC2, RWS, SIO, STREAM, TIME, TONE, TPAD, and VFMT calls.

The *Programming With System Calls for Interprocess Communication* handbook, order no. 005696, documents how to write programs that use the DOMAIN interprocess facilities including the MBX, MS, IPC, MUTEX, and EC2 calls.

The *Programming With DOMAIN 2D Graphics Metafile Resource* handbook, order no. 005097, documents how to write programs that use the DOMAIN 2D Graphics Metafile Resource.

The *Programming With DOMAIN Graphic Primitives* handbook, order no. 005808, documents how to write graphics programs that use the DOMAIN Graphics Primitive Resource.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

UPPERCASE	Uppercase words or characters in formats and command descriptions represent keywords that you must use literally.
lowercase	Lowercase words or characters in formats and command descriptions represent values that you must supply.
[ ]	Square brackets enclose optional items.
{ }	Braces enclose a list from which you must choose an item.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on the keyboard.
CTRL/Z	The notation CTRL/ followed by the name of a key indicates a control character sequence. Hold down <CTRL> while you type the character.
. . .	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.
. . .	Vertical ellipsis points mean that we have omitted irrelevant parts of a figure or example.

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. In order to make it easy for you to communicate with us, we provide the User Change Request (UCR) system for software-related comments, and the Reader's Response form for documentation comments. By using these formal channels, you make it easy for us to respond to your comments.

You can get more information about how to submit a UCR by consulting the *DOMAIN System Command Reference* manual. Refer to the CRUCR (Create User Change Request) Shell command description. You can view the same description on-line by typing:

$ HELP CRUCR <RETURN>

For documentation comments, a Reader's Response form is located at the back of each manual.

Contents

Introduction

This introductory section describes the DOMAIN system insert files and the format of the information found in the sections that follow. Each of these sections consist of a description of the data types used by a system manager, the syntax of the manager's programming calls, and the error messages generated by the system manager. We have arranged the sections of this manual **alphabetically**, by system manager name.

DOMAIN Insert Files

The DOMAIN system provides insert files that define data types, constants, values, and routine declarations. The insert files also define the exact form of each system call or routine. (Even the FORTRAN version does this using comments, although the FORTRAN compiler doesn't check the forms that you use.)

The DOMAIN system routines are divided, by function, into several subsystems. Each subsystem is controlled by a system manager. The routines of each subsystem are prefixed for easy indentification. A subsystem prefix consists of a number of identifying characters followed by the special underscore and dollar-sign characters, "_$." For example, the routines that perform stream functions are prefixed with STREAM_$. These subsystem prefixes are also used to distinguish DOMAIN data types and constants that are used by the subsystem routines.

Insert files are located in the directory /SYS/INS/. There is one insert file per subsystem for each programming language. Include the appropriate insert file for your programming language. For example, if you are using error routines in a Pascal program, you include the insert file, /SYS/INS/ERROR.INS.PAS. Using the same routines in a FORTRAN program, you include /SYS/INS/ERROR.INS.FTN. All insert files are specified using the syntax

/SYS/INS/subsystem-prefix.INS.language-abbreviation

where language abbreviation is PAS (Pascal), FTN (FORTRAN), or C (C). The listing on the next page shows all the available insert files.

In addition to including required subsystem insert files in a program, you must always include the BASE insert file for your programming language. You specify BASE insert files using the syntax

/SYS/INS/BASE.INS.language-abbreviation

These files contain some basic definitions that a number of subsystem routines use.

Summary of Insert Files

Insert File	Operating System Component
/SYS/INS/BASE.INS.lan	Base definitions -- must always be included
/SYS/INS/ACLM.INS.lan	Access control list manager
/SYS/INS/CAL.INS.lan	Calendar
/SYS/INS/ERROR.INS.lan	Error reporting
/SYS/INS/EC2.INS.lan	Eventcount
/SYS/INS/GM.INS.lan	Graphics Metafile Resource
/SYS/INS/GMF.INS.lan	Graphics Map Files
/SYS/INS/GPR.INS.lan	Graphics Primitives
/SYS/INS/IPC.INS.lan	Interprocess communications datagrams
/SYS/INS/KBD.INS.lan	[Useful constants for keyboard keys]
/SYS/INS/MBX.INS.lan	Mailbox manager
/SYS/INS/MS.INS.lan	Mapping server
/SYS/INS/MTS.INS.lan	Magtape/streams interface
/SYS/INS/MUTEX.INS.lan	Mutual exclusion lock manager
/SYS/INS/NAME.INS.lan	Naming server
/SYS/INS/PAD.INS.lan	Display Manager
/SYS/INS/PBUFS.INS.lan	Paste buffer manager
/SYS/INS/PFM.INS.lan	Process fault manager
/SYS/INS/PGM.INS.lan	Program manager
/SYS/INS/PM.INS.lan	User process routines
/SYS/INS/PROC1.INS.PAS	Process manager (Pascal only)
/SYS/INS/PROC2.INS.lan	User process manager
/SYS/INS/RWS.INS.lan	Read/write storage manager
/SYS/INS/SIO.INS.lan	Serial I/O
/SYS/INS/SMDU.INS.lan	Display driver
/SYS/INS/STREAMS.INS.lan	Stream manager
/SYS/INS/TIME.INS.lan	Time
/SYS/INS/TONE.lan	Speaker
/SYS/INS/TPAD.INS.lan	Touchpad manager
/SYS/INS/VEC.INS.lan	Vector arithmetic
/SYS/INS/VFMT.INS.lan	Variable formatter

The suffix ".lan" varies with the high-level language that you're using; it is either ".FTN", ".PAS", or ".C".

Organizational Information

This introductory section is followed by sections for each subsystem. The material for each subsystem is organized into the following three parts:

1. Detailed data type information (including illustrations of records for the use of FORTRAN programmers).

2. Full descriptions of each system call. Each call within a subsystem is ordered alphabetically.

3. List of possible error messages.

Data Type Sections

A subsystem's data type section precedes the subsystem's individual call descriptions. Each data type section describes the predefined constants and data types for a subsystem. These descriptions include an atomic data type translation (i.e., TIME_$REL_ABS_T = 4-byte integer) for use by FORTRAN programmers, as well as a brief description of the type's purpose. Where applicable, any predefined values associated with the type are listed and described. Following is an example of a data type description for the TIME_$REL_ABS_T type.

TIME_$REL_ABS_T

A 2-byte integer. Indicator of type of time. One of the following pre-defined values:

TIME_$RELATIVE
Relative time.

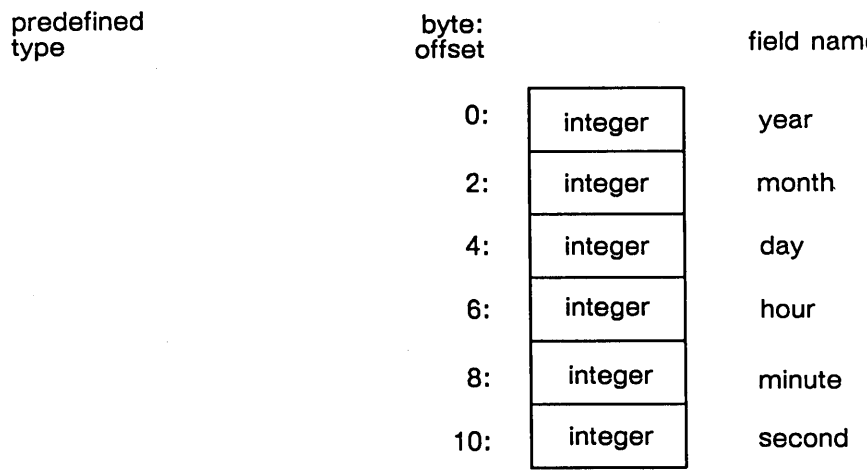
TIME_$ABSOLUTE
Absolute time.

In addition, the record data types are illustrated in detail. Primarily, we have geared these illustrations to FORTRAN programmers who need to construct record-like structures, but we've designed the illustrations to convey as much information as possible for all programmers. Each record type illustration:

- Clearly shows FORTRAN programmers the structure of the record that they must construct using standard FORTRAN data type statements. The illustrations show the size and type of each field.
- Describes the fields that make up the record.
- Lists the byte offsets for each field. These offsets are used to access fields individually.
- Indicates whether any fields of the record are, in turn, predefined records.

The following is the description and illustration of the CAL_$TIMEDATE_REC_T predefined record:

```
CAL_$TIMEDATE_REC_T                    Readable  time format.  The
                                       diagram below illustrates the
                                       CAL_$TIMEDATE_REC_T data type:
```




```
Field Description:
  year
  Integer representing the year.

  month
  Integer representing the month.

  day
  Integer representing the day.

  hour
  Integer  representing  the  hour
  (24 hr. format).

  minute
  Integer representing the minute.

  second
  Integer representing the second.
```

FORTRAN programmers, note that a Pascal variant record is a record structure that may be interpreted differently depending on usage. In the case of variant records, as many illustrations will appear as are necessary to show the number of interpretations.

System Call Descriptions

We have listed the system call descriptions alphabetically for quick reference. Each system call description contains:

- An abstract of the call's function.
- The order of call parameters.
- A brief description of each parameter.
- A description of the call's function and use.

These descriptions are standardized to make referencing the material as quick as possible.

Each parameter description begins with a phrase describing the parameter. If the parameter can be declared using a predefined data type, the descriptive phrase is followed by the phrase ",in XXX format," where XXX is the predefined data type. Pascal or C programmers, look for this phrase to determine how to declare a parameter.

FORTRAN programmers, use the second sentence of each parameter description for the same purpose. The second sentence describes the data type in atomic terms that you can use, such as "This is a 2-byte integer." In complex cases, FORTRAN programmers are referenced to the respective subsystem's data types section.

The rest of a parameter description describes the use of the parameter and the values it may hold.

The following is an example of a parameter description:

```
access
   New access mode, in MS_$ACC_MODE_T format. This is a 2-byte integer.
   Specify only one of the following predefined values:

      MS_$R          Read access.

      MS_$WR         Read and write access.

      MS_$RIW        Read with intent to write.

  An object which is locked MS_$RIW may not be changed to MS_$R.
```

Error Sections

Each error section lists the status codes that may be returned by subsystem calls. The following information appears for each error:

- Predefined constant for the status code.
- Text associated with the error.

ACLM

This section describes the call syntax for the ACLM programming calls. The ACLM calls do not use special data types or produce unique error messages. Refer to the Introduction at the beginning of this manual for a description of the call syntax format.

ACLM_$DOWN

Deasserts a program's subsystem manager rights.

FORMAT

```
ACLM_$DOWN
```

USAGE

This call deasserts a program's rights to gain access to an object in a protected subsystem, which were asserted by a previous call to ACLM_$UP.

ACLM_$UP

Asserts a program's subsystem manager rights.

FORMAT

```
ACLM_$UP
```

USAGE

This call asserts a program's rights to gain access to an object in a protected subsystem, until a corresponding call to ACLM_$DOWN is made.

Access Control List manager (ACLM) calls are used by subsystem manager programs in DOMAIN protected subsystems. A protected subsystem is a feature of the operating system that ensures that access to certain objects is restricted to certain programs which are called the managers of the subsystem that contains those objects.

In fact, even a subsystem manager, which has the right to gain access to the protected objects, must call ACLM_$UP to assert its rights before it can actually use a protected object. Calling ACLM_$DOWN deasserts a program's rights as a subsystem manager.

We recommend that you activate your rights as a subsystem manager for the minimum amount of time you will need them using ACLM_$UP and ACLM_$DOWN to bracket high-level statements or functions for which you need these rights. This ensures against inadvertent use of the protected objects.

Calling ACLM_$UP increments a counter in your process; calling ACLM_$DOWN decrements it. Subsystem manager operations are enabled whenever the counter is nonzero. Having a counter instead of a flag ensures that if one routine enables subsystem manager rights, and calls a routine that enables and disables subsystem manager rights, the calling routine does not inadvertently lose its rights.

Calling ACLM_$UP obtains all the subsystem rights to which you are entitled. If a program that is not a subsystem manager calls ACLM_$UP, it produces no effect, but does not return an error. Likewise, calling ACLM_$DOWN when subsystem manager rights were already deasserted has no effect.

Protected subsystems and the reasons for using them are discussed completely in the *Administering Your DOMAIN System*.

CAL

This section describes the data types, the call syntax, and the error codes for the CAL programming calls. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

CONSTANTS

CAL_$STRING_SIZE	80	Size of an ASCII string.

DATA TYPES

CAL_$DPVAL_T	A double-precision floating point value. A 2-element array of 4-byte integers. (REAL*8 for FORTRAN programs.)
CAL_$STRING_T	An array of up to CAL_$STRING_SIZE (80) characters. An ASCII string.

CAL_$TIMEDATE_REC_T

6 integer, readable time format. The diagram below illustrates the cal_$timedate_rec_t data type:

predefined type	byte: offset		field name
	0:	integer	year
	2:	integer	month
	4:	integer	day
	6:	integer	hour
	8:	integer	minute
	10:	integer	second

Field Description:

year
Integer representing the year.

month
Integer representing the month.

day
Integer representing the day.

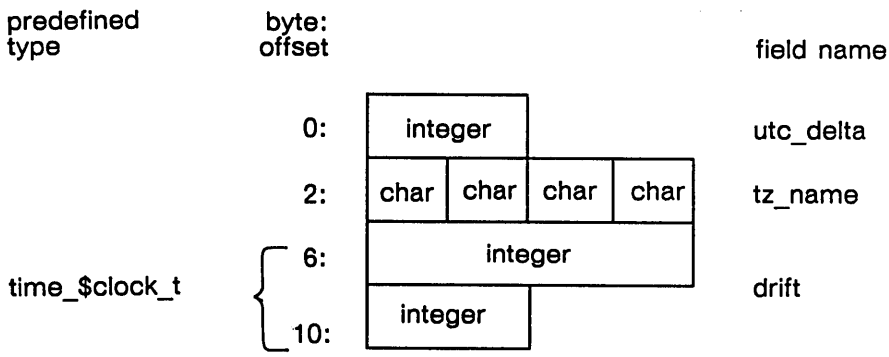
hour
Integer representing the hour (24 hr.).

minute
Integer representing the minute.

second
Integer representing the second.

CAL_$TIMEZONE_REC_T

Specifies time difference and timezone name. The diagram below illustrates the cal_$timezone_rec_t data type:



Field Description:

utc_delta
Number of minutes difference from UTC.

tz_name
Time zone name.

drift
Drift adjustment.

CAL_$TZ_NAME_T

An array of up to 4 characters. Time zone name.

CAL_$WEEKDAY_T

A 2-byte integer. Specifies the day of the week. One of the following pre-defined values:

CAL_$SUN
Sunday

CAL_$MON
Monday

CAL_$TUE
Tuesday

CAL_$WED
Wednesday

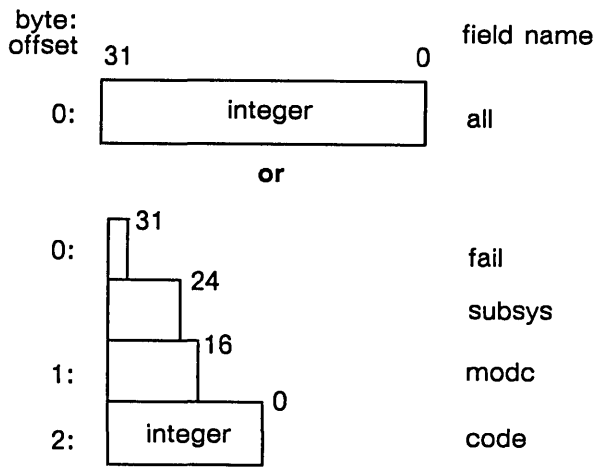
CAL_$THU
Thursday

CAL_$FRI
Friday

CAL_$SAT
Saturday

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

CAL_$ADD_CLOCK

Computes the sum of two times.

FORMAT

```
CAL_$ADD_CLOCK (clock1, clock2)
```

INPUT/OUTPUT PARAMETERS

clock1

Upon input — The Coordinated Universal Time clock value to be added to clock2, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

Upon output — The sum of clock1 and clock2, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

INPUT PARAMETERS

clock2

The Coordinated Universal Time clock value to be added to clock1, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types for more information.

CAL_$APPLY_LOCAL_OFFSET

Computes the local time from a UTC time.

FORMAT

```
CAL_$APPLY_LOCAL_OFFSET (clock)
```

INPUT/OUTPUT PARAMETERS

clock

Upon input	Coordinated Universal Time clock value to which a local time zone offset will be added, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.
Upon output	Adjusted clock value, representing the local time equivalent of the input parameter, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$APPLY_LOCAL_OFFSET adds the local time zone offset to the supplied clock value.

To set the local time zone offset, you may either execute the Shell command TZ (TIME_ZONE) as described in the *DOMAIN System Command Reference Manual*, or you may use the CAL_$WRITE_TIMEZONE procedure.

CAL_$CLOCK_TO_SEC

Converts system clock units to seconds.

FORMAT

```
seconds = CAL_$CLOCK_TO_SEC (clock)
```

RETURN VALUE

seconds
The computed equivalent of clock, in whole seconds. This is a 4-byte integer.

If the number of seconds calculated from the input value does not represent a whole number, the fractional portion is truncated.

INPUT PARAMETERS

clock
The value to be converted, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$CLOCK_TO_SEC converts a value in system clock representation (UTC) to an equivalent value in whole seconds.

The system clock value represents a time in units of 4 microseconds.

CAL_$CMP_CLOCK

Compares the values of two times.

FORMAT

```
integer  = CAL_$CMP_CLOCK (clock1, clock2)
```

RETURN VALUE

integer
The result of the logical compare of clock1 to clock2. This is a 2-byte integer.

```
Integer
returned

   1     if  clock1 > clock2
   0     if  clock1 = clock2
  -1     if  clock1 < clock2
```

INPUT PARAMETERS

clock1
The Coordinated Universal Time clock value to be compared to clock2, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

clock2
The Coordinated Universal Time clock value to be compared to clock1, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types for more information.

CAL_$DECODE_ASCII_DATE

Decodes an ASCII string containing a date specification.

FORMAT

```
CAL_$DECODE_ASCII_DATE (string, stringlength, year, month, day, status)
```

INPUT PARAMETERS

string
An ASCII character string, of length "stringlength" and in the form "year/month/day". This is an array of up to 80 characters.

stringlength
The number of characters in the string. This is a 2-byte integer.

OUTPUT PARAMETERS

year
The year decoded from the string. This is a 2-byte integer.

month
The month decoded from the string. This is a 2-byte integer.

day
The day decoded from the string. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the CAL Data Types section for more information.

Possible values are:

CAL_$BAD_SYNTAX
The string provided is not in "year/month/day" format.

CAL_$EMPTY_STRING
The string length is zero or the string contains only spaces.

CAL_$OUT_OF_RANGE
The value for month or day is invalid.

USAGE

CAL_$DECODE_ASCII_DATE translates the ASCII date in the supplied string into three integers representing the year, month, and day. The string must contain a year, a month, and a day, separated by slashes.

If a year between 80 and 99 is specified, CAL_$DECODE_ASCII_DATE adds 1900 to it before returning. If a year between 0 and 79 is specified, 2000 is added.

Leading and trailing spaces are ignored.

CAL_$DECODE_ASCII_TIME

Translates an ASCII string containing a time into integers.

FORMAT

```
CAL_$DECODE_ASCII_TIME (string, stringlength, hour, minute, second, status)
```

INPUT PARAMETERS

string
An ASCII character string of length "stringlength" in the form "hour:minute" or "hour:minute:second", in 24-hour format. This is an array of up to 80 characters.

stringlength
The number of characters in the string. This is a 2 -byte integer.

OUTPUT PARAMETERS

hour
The hour decoded from the string. This is a 2-byte integer.

minute
The minute decoded from the string. This is a 2-byte integer.

second
The second decoded from the string. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the CAL Data Types section for more information. Possible values are:

CAL_$BAD_SYNTAX
The string is not in either "hour:minute:second" or "hour:minute" format.

CAL_$EMPTY_STRING
String length is zero or the string contains only spaces.

CAL_$OUT_OF_RANGE
The supplied ASCII value for hour, minute, or second is invalid.

USAGE

CAL_$DECODE_ASCII_TIME translates the ASCII string into three integers, representing hours, minutes, and seconds.

If the string specifies only hours and minutes, the value returned for seconds is zero.

Leading and trailing spaces are ignored.

CAL_$DECODE_ASCII_TZDIF

Translates an ASCII string specifying a time zone into an offset from UTC.

FORMAT

```
CAL_$DECODE_ASCII_TZDIF (string, stringlength, tz-dif, tz-name, status)
```

INPUT PARAMETERS

string

An ASCII string containing a time zone name or time zone difference. This is an array of up to 80 characters.

A time zone name is one of the following strings: (EST) Eastern Standard Time ,(EDT) Eastern Daylight Time, (CST) Central Standard Time, (CDT) Central Daylight Time, (MST) Mountain Standard Time, (MDT) Mountain Daylight Time, (PST) Pacific Standard Time, (PDT) Pacific Daylight Time, (GMT) Greenwich Mean Time, and (UTC) Coordinated Universal Time. These are the eight standard U.S. time zone names, plus those for Greenwich Mean Time and Coordinated Universal Time.

A time zone difference is a value which, when added to Coordinated Universal Time, produces the local time. Specify a time zone difference in the following form:

```
[ + | - ]hour:minute
```

The hour must be a number between 0 and 12; the minute must be 0 or 30. The sign is optional. For example, Eastern Daylight Time may be represented as -4:00.

stringlength

The number of characters in the string. This is a 2-byte integer.

OUTPUT PARAMETERS

tz-dif

The difference, in minutes, between the time zone specified in string and UTC. This is a 2-byte integer.

The value of tz-dif is negative for time zones west of the Greenwich Meridian and positive for time zones east of the Greenwich Meridian.

tz-name

The time zone name, in CAL_$TZ_NAME_T format. This is an array of up to 4 characters.

If the ASCII string contains a time zone name, this procedure returns that name in tz-name. If the ASCII string contains a time zone difference, this procedure returns spaces in tz-name.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the CAL Data Types section for more information. Possible values are:

CAL_$EMPTY_STRING
: String length is zero or the string contains only spaces.

CAL_$UNKNOWN_TIMEZONE
: The string contains a time zone name that is unknown to this procedure.

CAL_$BAD_SYNTAX
: The string appears to contain a time zone difference but is syntactically incorrect.

CAL_$INVALID_TZDIF
: The string contains a time zone difference, but the number of hours is greater than 12 or the number of minutes is not 0 or 30.

USAGE

CAL_$DECODE_ASCII_TZDIF translates the supplied ASCII string into an offset from UTC, in units of minutes. The ASCII string can contain a time zone name or a time zone difference.

CAL_$DECODE_LOCAL_TIME

Returns the local time in integer format.

FORMAT

```
CAL_$DECODE_LOCAL_TIME (decoded_clock)
```

OUTPUT PARAMETERS

decoded_clock
The local time, in CAL_$TIMEDATE_REC_T format. This data type is 12 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$DECODE_LOCAL_TIME returns the local time in "year, month, day, hour, minute, second" format.

CAL_$DECODE_TIME

Translates an internal system clock value into a readable date and time.

FORMAT

```
CAL_$DECODE_TIME (clock, decoded_clock)
```

INPUT PARAMETERS

clock
The value to be translated, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

OUTPUT PARAMETERS

decoded_clock
A date and time, in CAL_$TIMEDATE_REC_T format. This data type is 12 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$DECODE_TIME translates a time in TIME_$CLOCK_T form into CAL_$TIMEDATE_REC_T ("year, month, day, hour, minute, second") form.

This routine translates clock values, such as those returned from the TIME_$CLOCK, CAL_$GET_LOCAL_TIME, CAL_$APPLY_LOCAL_OFFSET, and CAL_$ENCODE TIME routines.

CAL_$ENCODE_TIME

Translates a date and time from integer format into a system clock representation.

FORMAT

```
CAL_$ENCODE_TIME (decoded_clock, clock)
```

INPUT PARAMETERS

decoded_clock
A date and time, in CAL_$TIMEDATE_REC_T format. This data type is 12 bytes long. See the CAL Data Types section for more information.

OUTPUT PARAMETERS

clock
The system clock equivalent value of decoded_clock, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$ENCODE_TIME translates the date and time specified by decoded_clock into the equivalent system representation.

CAL_$FLOAT_CLOCK

Converts a system clock representation to the equivalent number of seconds, in double-precision floating-point format.

FORMAT

```
CAL_$FLOAT_CLOCK (clock, float_seconds)
```

INPUT PARAMETERS

clock
The value to be converted, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

OUTPUT PARAMETERS

float_seconds
The converted value of clock, in seconds. This is in double-precision floating-point (REAL*8) format.

USAGE

CAL_$FLOAT_CLOCK converts a clock value in UTC format to the equivalent number of seconds expressed as a double-precision floating-point number.

Unlike CAL_$CLOCK_TO_SEC, CAL_$FLOAT_CLOCK does not truncate the fractional portion of the conversion.

CAL_$GET_INFO

Returns local time zone information.

FORMAT

```
CAL_$GET_INFO (timezone_info)
```

OUTPUT PARAMETERS

timezone_info

A record containing the name of the local time zone and its offset from UTC, in CAL_$TIMEZONE_REC_T format. This data type is 12 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$GET_INFO returns the name of the local time zone and the difference between local time and Coordinated Universal Time (UTC).

CAL_$GET_LOCAL_TIME

Returns the current local time in system clock representation.

FORMAT

```
CAL_$GET_LOCAL_TIME (clock)
```

OUTPUT PARAMETERS

clock

The current local time, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

This is the number of 4-microsecond periods that have elapsed since January 1, 1980, 00:00.

CAL_$REMOVE_LOCAL_OFFSET

Computes the UTC time from local time.

FORMAT

```
CAL_$APPLY_LOCAL_OFFSET (clock)
```

INPUT/OUTPUT PARAMETERS

clock

Upon input	Local time from which the local time offset will be removed, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.
Upon output	Adjusted clock value, representing the UTC equivalent of the input parameter, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$REMOVE_LOCAL_OFFSET subtracts the local time zone offset from the supplied clock value.

To set the local time zone offset, you may either execute the Shell command TZ (TIME_ZONE) as described in the *DOMAIN System Command Reference*, or you may use the CAL_$WRITE_TIMEZONE procedure.

CAL_$SEC_TO_CLOCK

Converts seconds to system clock units.

FORMAT

```
CAL_$SEC_TO_CLOCK (seconds, clock)
```

INPUT PARAMETERS

seconds
The value to be converted. This is a 4-byte integer.

OUTPUT PARAMETERS

clock
The computed equivalent of seconds, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$SEC_TO_CLOCK converts a value representing seconds to an equivalent value in 4-microsecond units.

No overflow detection is performed.

CAL_$SUB_CLOCK

Subtracts the values of two times.

FORMAT

```
value  = CAL_$SUB_CLOCK (clock1, clock2)
```

RETURN VALUE

value

The Boolean result of the subtraction of clock2 from clock1. The returned value is TRUE if the result is >= 0.

INPUT/OUTPUT PARAMETERS

clock1

Upon input	The Coordinated Universal Time clock value from which clock2 is subtracted, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.
Upon output	The difference between clock1 and clock2, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

INPUT PARAMETERS

clock2

The Coordinated Universal Time clock value to be subtracted from clock1, in TIME_$CLOCK_T format. This data type is 6 bytes long. See the CAL Data Types section for more information.

CAL_$WEEKDAY

Computes the day of the week given a year, month, and day.

FORMAT

```
weekday = CAL_$WEEKDAY (year, month, day)
```

RETURN VALUE

weekday
The computed day of the week, in CAL_$WEEKDAY_T format. This is a 2-byte integer. Returns one of the following predefined values:

```
CAL_$SUN, CAL_$MON, CAL_$TUE, CAL_$WED,
CAL_$THU, CAL_$FRI, CAL_$SAT.
```

Their ordinal values are 0 through 6.

INPUT PARAMETERS

year
The year for which the weekday is desired. This is a 2-byte integer.

month
The month for which the weekday is desired. This is a 2-byte integer.

day
The day of the month for which the weekday is desired. This is a 2-byte integer.

USAGE

CAL_$WEEKDAY computes the day of the week for any Gregorian date.

CAL_$WRITE_TIMEZONE

Writes local time zone information onto the boot volume.

FORMAT

```
CAL_$WRITE_TIMEZONE (timezone_info, status)
```

INPUT PARAMETERS

timezone_info

The time zone information to be recorded, in CAL_$TIMEZONE_REC_T format. This data type is 12 bytes long. See the CAL Data Types section for more information.

The supplied time zone information includes the name of the time zone and its offset form UTC.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the CAL Data Types section for more information.

USAGE

CAL_$WRITE_TIMEZONE writes the supplied time zone information onto the logical disk volume from which the operating system was started.

This procedure is invalid on a diskless node, and returns a nonzero status.

The time zone information written by this procedure is used by subsequent calls to CAL_$DECODE_LOCAL_TIME, CAL_$GET_LOCAL_TIME, CAL_$APPLY_LOCAL_OFFSET, and CAL_$GET_INFO.

A nonzero status indicates a system problem in reading or writing the volume.

ERRORS

STATUS_$OK
: Successful completion.

CAL_$BAD_SYNTAX
: Invalid syntax for date or time specification.

CAL_$EMPTY_STRING
: An empty string was passed to a decode routine.

CAL_$INVALID_TZDIF
: Invalid time-zone difference.

CAL_$OUT_OF_RANGE
: Date or time specification invalid.

CAL_$UNKNOWN_TIMEZONE
: Timezone specified is unknown.

EC2

This section describes the data types, the call syntax, and the error codes for the EC2 programming calls. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

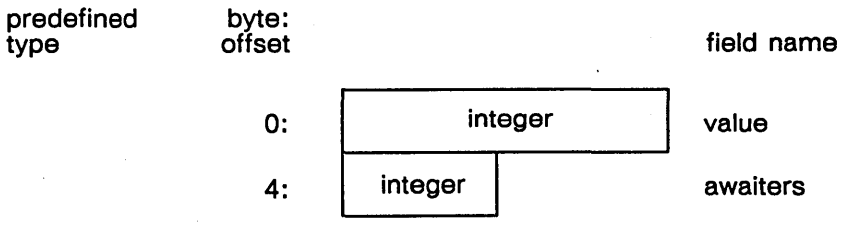
CONSTANTS

EC2_$ALWAYS_READY_EC	1	Replaces an EC2_$PTR pointer to indicate that the event is always ready.

DATA TYPES

EC2_$EVENTCOUNT_T

An eventcount. The diagram below illustrates the EC2_$EVENTCOUNT_T data type:



Field Description:

value
The value of the eventcount.

awaiters
Reserved for internal use by the EC2 manager.

EC2_$PTR_LIST_T

An array of up to 32 pointers to eventcounts. Each pointer is a 4-byte integer in EC2_$PTR_T format.

EC2_$PTR_T

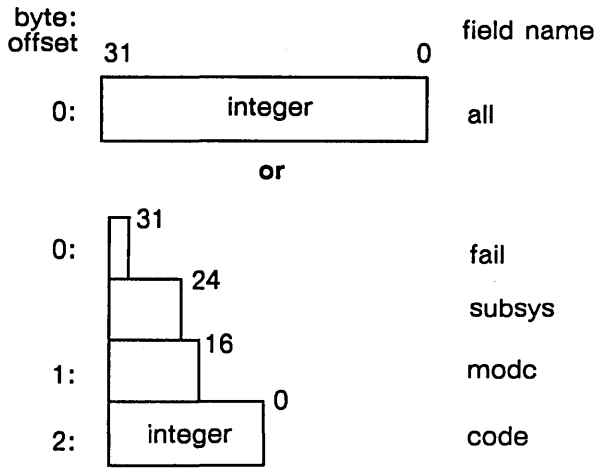
A 4-byte integer. A pointer to an eventcount.

EC2_$VAL_LIST_T

An array of trigger values for each of the eventcounts in an eventcount pointer list. Each trigger value is a positive, 4-byte integer.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

EC2_$ADVANCE

Advances the specified user-defined eventcount by one.

FORMAT

```
EC2_$ADVANCE (eventcount, status)
```

INPUT/OUTPUT PARAMETERS

eventcount

Eventcount to be advanced, in EC2_$EVENTCOUNT_T format. This data type is 6 bytes long. See the EC2 Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the EC2 Data Types section for more information.

USAGE

EC2_$ADVANCE advances a user-defined eventcount; do not use this call to advance a system-defined eventcount. A user-defined eventcount is one that a process establishes with EC2_$INIT.

In order to advance an eventcount, you must have read and write access to the emory location where the eventcount is located. Typically, the eventcount is in a mapped file, and you should use MS_$MAPL to map this file.

When you map a file containing an eventcount, you should request a shared write lock. Only processes on the same node can concurrently get shared write locks on the same file. See MS_$MAPL for more information about mapping a file.

EC2_$INIT

Initializes a user-defined eventcount.

FORMAT

```
EC2_$INIT (eventcount)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

eventcount

Initialized eventcount, in EC2_$EVENTCOUNT_T format. This data type is 6 bytes long. See the EC2 Data Types section for more information.

USAGE

Use this call to initialize a user-defined eventcount. Initialize the eventcount within a file that several programs will share. First, map the file for shared write access (by requesting a shared write lock.) Then use EC2_$INIT to initialize the eventcount. All programs that use the eventcount must first map the file containing the eventcount. See the Mapped Segment (MS) calls for more information on mapping.

Do not use EC2_$INIT to initialize a system-defined eventcount; the system automatically initializes eventcounts associated with system events. To use a system-defined eventcount, use the system call that gets the address of the eventcount you want to wait on. For example, use MBX_$GET_EC to get the address of a mailbox eventcount.

EC2_$READ

Returns the current value of an eventcount.

FORMAT

```
ec-value = EC2_$READ (eventcount)
```

RETURN VALUE

ec-value
Value of the eventcount. This is a positive, 4-byte integer.

INPUT PARAMETERS

eventcount
Eventcount, in EC2_$EVENTCOUNT_T format. This data type is 6 bytes long. See the EC2 Data Types section for more information.

OUTPUT PARAMETERS

None.

USAGE

Use EC2_$READ to read the value of an eventcount.

EC2_$WAIT

Waits until any of a list of eventcounts reaches a trigger value.

FORMAT

```
ec-satisfied = EC2_$WAIT (ec-plist, ec-vlist, ec-count, status)
```

RETURN VALUE

ec-satisfied
An ordinal number indicating the eventcount that is satisfied. This is a positive, 2-byte integer.

INPUT PARAMETERS

ec-plist
Array of pointers to eventcounts. Each pointer is a 4-byte integer in EC2_$PTR_T format. The total number of eventcounts in ec-plist lists in any one node cannot exceed 32.

ec-vlist
Array of trigger values for each of the eventcounts in the ec-plist. Each trigger value is a positive, 4-byte integer. When any of the eventcounts from the ec-plist reaches its trigger value, the EC2_$WAIT call returns.

ec-count
Number of eventcounts in the ec-plist. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the EC2 Data Types section for more information.

USAGE

EC2_$WAIT waits until one of the eventcounts in the ec-plist reaches its trigger value in the ec-vlist. When an eventcount reaches its trigger value, EC2_$WAIT returns an index value indicating the position (in the ec-plist) of the eventcount that is satisfied. The index starts from 1; that is, ec-satisfied equals 1 if the first eventcount in the ec-plist is satisfied.

Several eventcounts may have been satisfied by the time this call wakes your program. The index number returned refers to only one of the eventcounts. If more than one eventcount is satisfied, EC2_$WAIT returns the one with the smallest subscript.

EC2_$WAIT only returns when an eventcount advances, regardless of the asynchronous fault handling setting. An asynchronous fault, such as a "quit", is generated outside your program. If an asynchronous fault occurs during an EC2_$WAIT call, your program's response depends on the type of error handling that is in effect.

If asynchronous faults are enabled, a program can respond to an asynchronous fault with a clean-up handler or fault handler. If an asynchronous fault occurs during an EC2_$WAIT call, and asynchronous faults are enabled, the program will perform one of the following:

- Execute the clean-up handler, if the program has one.
- Execute the fault handler, if the program has one. If the fault handler returns control to the interrupted code, EC2_$WAIT continues waiting.
- If the program has neither a clean-up handler nor a fault handler, the program aborts if an asynchronous fault occurs.

If a program disables asynchronous faults and such a fault occurs during an EC2_$WAIT, then the program ignores the fault and continues waiting.

Note that the call EC2_$WAIT_SVC responds differently to asynchronous faults.

EC2_$WAIT_SVC

Waits until any of a list of eventcounts reaches a trigger value.

FORMAT

```
ec-satisfied = EC2_$WAIT_SVC (ec-plist, ec-vlist, ec-count, status)
```

RETURN VALUE

ec-satisfied
An ordinal number indicating the eventcount that was satisfied. This is a positive, 2-byte integer.

INPUT PARAMETERS

ec-plist
Array of pointers to eventcounts. Each pointer in the array is a 4-byte integer in EC2_$PTR_T format. The total number of eventcounts in ec-plist lists in any one node cannot exceed 32.

ec-vlist
Array of trigger values for each of the eventcounts in the ec-plist. Each trigger value is a positive, 4-byte integer. When any of the eventcounts from the ec-plist reaches its trigger value, the EC2_$WAIT_SVC call returns.

ec-count
Number of eventcounts in the ec-plist. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the EC2 Data Types section for more information.

USAGE

EC2_$WAIT_SVC waits until one of the eventcounts in the ec-plist reaches its trigger value in the ec-vlist. When an eventcount reaches its trigger value, EC2_$WAIT_SVC returns an index value indicating the position (in the ec-plist) of the eventcount that is satisfied. The index starts from 1; that is, ec-satisfied equals 1 if the first eventcount in the ec-plist is satisfied.

Several eventcounts may have been satisfied by the time this call wakes your program. The index number returned refers to only one of the eventcounts. If more than one eventcount is satisfied, EC2_$WAIT_SVC returns the one with the smallest subscript.

In certain cases, EC2_$WAIT_SVC returns the error EC2_$WAIT_QUIT if an asynchronous fault occurs during the EC2_$WAIT_SVC call. An asynchronous fault, such as a "quit", is generated outside your program. If an asynchronous fault occurs during an EC2_$WAIT call, your program's response depends on the type of error handling that is in effect.

If asynchronous faults are enabled, a program can respond to an asynchronous fault with a clean-up handler or a fault handler. If an asynchronous fault occurs during an EC2_$WAIT_SVC call, and asynchronous faults are enabled, the program will perform one of the following:

- Execute the clean-up handler, if the program has one.
- Execute the fault handler, if the program has one. If the fault handler returns control to the interrupted code, EC2_$WAIT_SVC returns the error EC2_$WAIT_QUIT.
- If the program has neither a clean-up handler nor a fault handler, the program aborts if an asynchronous fault occurs.

If a program disables asynchronous faults and such a fault occurs during an EC2_$WAIT_SVC, then the program does not handle the fault. However, EC2_$WAIT_SVC returns the error EC2_$WAIT_QUIT.

Note that the call EC2_$WAIT responds differently to asynchronous faults.

ERRORS

STATUS_$OK
: Successful completion.

EC2_$BAD_EVENTCOUNT
: Bad eventcount.

EC2_$INTERNAL_ERROR
: Internal error.

EC2_$NO_WAIT_ENTRIES
: Internal table exhausted.

EC2_$WAIT_QUIT
: Process quit while waiting.

ERROR

This section describes the data types and the call syntax for the ERROR programming calls. The ERROR calls do not produce unique error messages. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

DATA TYPES

ERROR_$INTEGER32

A 2-byte integer. Possible values are integers from -1 through 2147483647.

ERROR_$STRING_PTR_T

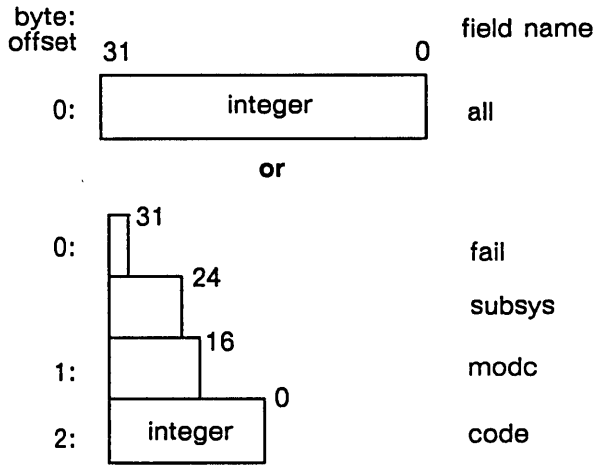
A 4-byte integer. A pointer to an ERROR_$STRING_T data type.

ERROR_$STRING_T

An array of up to 80 characters.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

ERROR_$CODE

Returns the module-specific code from a status code.

FORMAT

```
code = ERROR_$CODE (status)
```

RETURN VALUE

code
The module-specific code component of the supplied status code. This is a 2-byte integer.

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

USAGE

ERROR_$CODE extracts and returns the module-specific code from the supplied status code. The module-specific code is the rightmost 16 bits of a STATUS_$T data type.

This routine is intended for use by FORTRAN programs that need to check for specific status codes. Pascal programs can refer to this component directly.

ERROR_$FAIL

Returns the state of the fail bit of a status code.

FORMAT

```
fail = ERROR_$FAIL (status)
```

RETURN VALUE

fail
The value of the fail bit of the status code. This is a Boolean (logical) value.

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

USAGE

ERROR_$FAIL extracts and returns the value of the fail bit of the supplied status code. The fail bit is bit number 31 in the STATUS_$T data type.

This routine is intended for use by FORTRAN programs that need to check for specific status codes. Pascal programs can refer to this component directly.

ERROR_$FIND_TEXT

Finds the text associated with a status code and returns pointers.

FORMAT

```
ERROR_$FIND_TEXT (status, subsys_p, subsys_l, module_p, module_l,
code_p, code_l)
```

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

OUTPUT PARAMETERS

subsys_p
The returned pointer to the subsystem name, in ERROR_$STRING_PTR_T format. This is a 4-byte integer.

subsys_l
The number of characters in the string pointed to by subsys_p. This is a 2-byte integer.

module_p
The returned pointer to the module name, in ERROR_$STRING_PTR_T format. This is a 4-byte integer.

module_l
The number of characters in the string pointed to by module_p. This is a 2-byte integer.

code_p
The returned pointer to the diagnostic text, in ERROR_$STRING_PTR_T format. This is a 4-byte integer.

code_l
The number of characters in the string pointed to by code_p. This is a 2-byte integer.

USAGE

ERROR_$FIND_TEXT looks up and returns pointers to the text associated with a status code.

Text is associated with three components of the STATUS_$T type: subsystem name ("subsys"), module name ("module"), and error text ("code"). If ERROR_$FIND_TEXT cannot find the text associated with a component in the status code, a string length of zero is returned for the component. In this case, the pointer for that component is not useable.

If the subsystem text length is zero, the status is invalid. If the module text length is zero, both the module and code fields are invalid.

FORTRAN programs should use ERROR_$GET_TEXT instead of this routine.

ERROR_$GET_TEXT

Finds the text associated with a status code and returns strings.

FORMAT

```
ERROR_$GET_TEXT (status, subsys_t, subsys_l, module_t, module_l,
code_t, code_l)
```

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

OUTPUT PARAMETERS

subsys_t
The text string containing the subsystem name, in ERROR_$STRING_T format. This is an array of up to 80 characters.

subsys_l
The number of characters in the subsystem name. This is a 2-byte integer.

module_t
The text string containing the module name, in ERROR_$STRING_T format. This is an array of up to 80 characters.

module_l
The number of characters in the module name. This is a 2-byte integer.

code_t
The text string containing the diagnostic text, in ERROR_$STRING_T format. This is an array of up to 80 characters.

code_l
The number of characters in the diagnostic text. This is a 2-byte integer.

USAGE

ERROR_$GET_TEXT looks up and returns the text associated with a status code.

Text is associated with three components of the STATUS_$T type: subsystem name ("subsys"), module name ("module"), and error text ("code"). If ERROR_$GET_TEXT cannot find the text associated with a component in the status code, a string length of zero is returned for the component.

If the subsystem text length is zero, the status is invalid. If the module text length is zero, both the module and code fields are invalid.

The returned strings are not blank-filled. They contain only the number of characters necessary to represent the names and diagnostic text.

ERROR_$INIT_STD_FORMAT

Establishes the values to be used in subsequent calls to ERROR_$STD_FORMAT.

FORMAT

```
ERROR_$INIT_STD_FORMAT (stream-id, prefix-char, command, length)
```

INPUT PARAMETERS

stream_id
The stream on which to write the error output, in STREAM_$ID_T format. This is a 2-byte integer. This is usually STREAM_$ERROUT (Stream ID = 3).

prefix-char
The prefix element of the error format. This is one character. For system messages, this value is usually a question mark (?).

command
The command name, in ERROR_$STRING_T format. This is an array of up to 80 characters.

length
The length of the command name, in bytes. This value can be zero.

USAGE

This call establishes constant values for the standard error reporting format. Subsequent calls to ERROR_$STD_FORMAT cause error messages to use the values supplied in this call.

Multiple calls may be made to ERROR_$INIT_STD_FORMAT, but the information is kept on a per-process-level basis. Thus, successive calls to ERROR_$INIT_STD_FORMAT on the same process level replace previous error format definitions.

Calling ERROR_$INIT_STD_FORMAT and ERROR_$STD_FORMAT is equivalent to calling ERROR_$PRINT_FORMAT. For programs that use common subroutines, the former method provides more flexibility. For example, if an application's command level sets the command name with ERROR_$INIT_STD_FORMAT, it automatically provides the common lower-level modules with the correct command name for their error messages. Also, because ERROR_$STD_FORMAT has fewer parameters, it is easier to code using the pair of calls instead of using ERROR_$PRINT_FORMAT.

ERROR_$MODULE

Returns the module component from a status code.

FORMAT

```
module = ERROR_$MODULE (status)
```

RETURN VALUE

module
The module component of the supplied status code. This is a 2-byte integer.

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

USAGE

ERROR_$MODULE extracts and returns the module component of the supplied status code. The module is found in bits 23 through 16 of the STATUS_$T data type.

This routine is intended for use by FORTRAN programs that need to check for specific status codes. Pascal programs can refer to this component directly.

ERROR_$PRINT

Prints error text associated with a status code.

FORMAT

```
ERROR_$PRINT (status)
```

INPUT PARAMETERS

status

A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

USAGE

ERROR_$PRINT looks up the text associated with the status code and writes it to the error output stream.

If text is associated with all fields in the status code (subsystem, module, and code), a line is output containing the subsystem and module names.

If the text for any of the three fields is not found, the status code is displayed in hexadecimal, along with the subsystem and module names, if known.

The STCODE command, which can be used to view error messages, uses ERROR_$PRINT to output the error text.

ERROR_$PRINT_FORMAT

Prints a status code in the given error format.

FORMAT

```
ERROR_$PRINT_FORMAT (status, stream-id, prefix-char, command, length,
                     control-string, a1, a2, ... a10)
```

INPUT PARAMETERS

status
The status code to be displayed in standard error format, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

If the status code is zero, the dash and following error text are omitted from the message.

stream-id
The stream on which to write the error output, in STREAM_$ID_T format. This is a 2-byte integer. This value is usually STREAM_$ERROUT (Stream ID + 3).

prefix-char
The prefix element of the error format. This is one character. For system error messages, this value is usually a question mark (?).

command
The command name, in ERROR_$STRING_T format. This is an array of up to 80 characters.

length
Length of the command name, in bytes. This is a 2-byte integer. If length is zero, the command name portion of the standard error format is omitted.

control-string
A character string that contains text and control information for encoding the arguments that follow. It is a VFMT-COMMENT control string that must at least contain the two special characters (%, $). For detailed information on VFMT control strings, see Chapter 3 of *Programming With General System Calls*.

a1, a2, ... a10
One-to-ten substitution arguments that contain data for encoding using the control-string parameter.

If you are encoding ASCII text strings, you must provide two variables for each text string: a character string containing the string, and a 2-byte integer variable containing the length of the string.

USAGE

ERROR_$PRINT_FORMAT prints an error in the standard error format.

ERROR_$PRINT_FORMAT takes a variable number of arguments in the a1...a10 parameters. However, all arguments up to and including the control string must be given.

This routine uses the same control string format as the variable formatting routine VFMT_$WRITE.

ERROR_$PRINT_NAME

Prints error text associated with a status code, along with a user-supplied name.

FORMAT

```
ERROR_$PRINT_NAME (status, name, namelength)
```

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

name
The name to be printed. This is an array of up to 80 characters.

namelength
The length of the name to be printed, in bytes. This is a 2-byte integer.

USAGE

ERROR_$PRINT_NAME looks up the text associated with the status code and writes it to the error output stream, along with the supplied name.

If text is associated with all fields in the status code (subsystem, module, and code), output appears with the supplied name first, followed by a descriptive error message corresponding to the status code, followed the subsystem and module names in parentheses.

If the text for any of the three fields is not found, the status code is displayed in hexadecimal, along with the subsystem and module names, if known. The supplied name is also displayed, in the form shown above.

ERROR_$STD_FORMAT

Prints the status code in the standard error format using the values specified in the last call to ERROR_$INIT_STD_FORMAT.

FORMAT

```
ERROR_$STD_FORMAT (status, control-string, a1, a2, ... a10)
```

INPUT PARAMETERS

status

The status to be printed in standard error format, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

control-string

A character string that contains text and control information for encoding the arguments that follow. It is a VFMT-COMMENT control string that must at least contain the two special characters (%, $). For more information on VFMT control strings, see Chapter 3 of *Programming With General System Calls*.

a1, a2, ... a10

One-to-ten substitution arguments that contain data for encoding using the control-string parameter.

If you are encoding ASCII text strings, you must provide two variables for each text string: a character string containing the string, and a 2-byte integer variable containing the length of the string.

USAGE

Programs using ERROR_$STD_FORMAT must first call ERROR_$INIT_STD_FORMAT to establish constant values for the standard error reporting format.

This routine uses the same control string format as the variable formatting routine VFMT_$WRITE.

ERROR_$SUBSYS

Returns the subsystem component from a status code.

FORMAT

```
subsys = ERROR_$SUBSYS (status)
```

RETURN VALUE

subsys
The subsystem component of the supplied status code. This is a two-byte integer.

INPUT PARAMETERS

status
A status code returned from DOMAIN software, in STATUS_$T format. This data type is 4 bytes long. See the ERROR Data Types section for more information.

USAGE

ERROR_$SUBSYS extracts and returns the subsystem component of the supplied status code. The subsystem is found in bits 30 through 24 of the STATUS_$T data type.

This routine is intended for use by FORTRAN programs that need to check for specific status codes. Pascal programs can refer to this component directly.

GM

This section describes the data types, the call syntax, and the error codes for the GM programming calls. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

CONSTANTS

GM_$MAX_ACLASS	16	The maximum number of attribute classes is 16.
GM_$MAX_ARRAY_LENGTH	1000	The maximum number of elements in a gm_$array..._t is 1000.
GM_$MAX_BLOCK	40	The maximum number of attribute blocks is 40.
GM_$MAX_CURSOR_PATTERN_WORDS		16 The maximum number of words in a cursor pattern.
GM_$MAX_DRAW_PATTERN_BYTES		8 The maximum number of bytes in a draw pattern.
GM_$MAX_FILE	16	The maximum number of files is 16.
GM_$MAX_FILL_PATTERN_LWORDS		32 The max number of long words in a fill pattern.
GM_$MAX_FONT	32	The maximum number of font family identification numbers is 32.
GM_$MAX_FONT_FAMILY	8	The maximum number of font families is 8.
GM_$MAX_GRID	4	The maximum number of grids that may be associated with a vewport.
GM_$MAX_INSTANCE_DEPTH	32	The maximum instancing depth.
GM_$MAX_PIXEL_VALUE	255	The maximum value for color map entries; the numbers are 0 through 255.
GM_$MAX_PLANE_ID	7	The maximum number of planes.
GM_$MAX_PRIM_ID	16	The maximum number of primitive commands is 16.
GM_$MAX_SEGMENT	65536	The maximum number of segments; the numbers are 0 through 65536
GM_$MAX_SEGMENT_ID	16#7FFFFFFF	The largest possible segment id.
GM_$MAX_SEGMENT_NAME_LENGTH		12 Maximum length for segment names is 12.
GM_$MAX_STRING_LENGTH	12	The maximum length of a GM string is 12.
GM_$MAX_VIEWPORT	64	The maximum number of viewports is 64.
GM_$OUT1_CIRCLE	16#40	Opcode to format vector output.
GM_$OUT1_CIRCLE_FILL	16#41	Opcode to format vector output.

GM_$OUT1_CURVE	16#50	Opcode to format vector output.
GM_$OUT1_DRAW_RASTER_OP	16#82	Opcode to format vector output.
GM_$OUT1_DRAW_STYLE	16#81	Opcode to format vector output.
GM_$OUT1_DRAW_VALUE	16#80	Opcode to format vector output.
GM_$OUT1_EOF	16#00	Opcode to format vector output.
GM_$OUT1_FILL_BACKGROUND_VALUE	16#91	Opcode to format vector output.
GM_$OUT1_FILL_PATTERN	16#92	Opcode to format vector output.
GM_$OUT1_FILL_VALUE	16#90	Opcode to format vector output.
GM_$OUT1_FONT_FAMILY	16#A3	Opcode to format vector output.
GM_$OUT1_PLANE_MASK	16#82	Opcode to format vector output.
GM_$OUT1_POLYLINE_2D	16#20	Opcode to format vector output.
GM_$OUT1_POLYLINE_CLOSE_2D	16#21	Opcode to format vector output.
GM_$OUT1_POLYLINE_FILL_2D	16#22	Opcode to format vector output.
GM_$OUT1_PRIMITIVE	16#60	Opcode to format vector output.
GM_$OUT1_RECTANGLE	16#30	Opcode to format vector output.
GM_$OUT1_RECTANGLE_FILL_2D	16#31	Opcode to format vector output.
GM_$OUT1_TEXT	16#70	Opcode to format vector output.
GM_$OUT1_TEXT_BACKGROUND_VALUE	16#A1	Opcode to format vector output.
fd		
GM_$OUT1_TEXT_SIZE	16#A2	Opcode to format vector output.
GM_$OUT1_TEXT_VALUE	16#A0	Opcode to format vector output.

DATA TYPES

GM_$ACC_CREATE_T

A 2-byte integer. Specifies the access mode. One of the following predefined values:

GM_$WRITE
An error is returned if the file already exists.

GM_$OVERWRITE
The previous version is deleted if the file already exists.

GM_$UPDATE
The previous version is opened if the file already exists.

GM_$ACC_OPEN_T

A 2-byte integer. Specifies the read/write accessibility. One of the following predefined values:

GM_$WR
Access is read or write.

GM_$R
Access is read only.

GM_$CWR
Access is read or write; if the file does not exist, create it.

GM_$ARRAY16_T

An array of 2-byte integers with MAX_ARRAY_LENGTH elements. A list of coordinate points.

GM_$ARRAY32_T

An array of 4-byte integers with MAX_ARRAY_LENGTH elements. A list of coordinate points.

GM_$ARRAYREAL_T

An array of floating-point numbers with MAX_ARRAY_LENGTH elements. A list of coordinate points.

GM_$BORDER_UNIT_T

A 2-byte integer. The units for border size. One of the following predefined values:

GM_$PIXELS
Expresses edge width in pixels.

GM_$FRACTIONS
Expresses edge width as fractions of the total GM bitmap size.

GM_$BOUNDSREAL_T

Defines the bounds of a rectangular area. The diagram below illustrates the gm_$boundsreal_t data type:

byte: offset		field name
0:	real	xmin
4:	real	ymin
8:	real	xmax
12:	real	ymax

Field Description:

xmin
The x-coordinate of the bottom-left corner of the rectangle.

ymin
The y-coordinate of the bottom-left corner of the rectangle.

xmax
The x-coordinate of the top-right corner of the rectangle.

ymax
The y-coordinate of the top-right corner of the rectangle.

GM_$COLOR_ENTRY_T

A 3-element array of real values. Specifies color values in this order: red, green, blue.

GM_$COLOR_VECTOR_T

An array of 4-byte integers, of up to 256 elements. Specifies a list of color values.

GM_$COMMAND_TYPE_T

A 2-byte integer. Specifies the command type as follows: One of the following predefined values:

GM_$TACLASS
Attribute class.

GM_$TCIRCLE_2D
Circle.

GM_$TCURVE_2D
Curve.

GM_$TDRAW_RASTER_OP
Raster operations used in drawing.

GM_$TDRAWSTYLE
Line style used in drawing.

GM_$TDRAWVALUE
Pixel value used in drawing.

GM_$TFILLVALUE
Fill value used in drawing.

GM_$TFILLPATTERN
Fill pattern used in drawing.

GM_$TFONTFAMILY
Font family.

GM_$TINSTANCE_SCALE_2D
Scale and translate a segment instance.

GM_$TINSTANCE_TRANS_2D
Translate a segment instance.

GM_$TINSTANCE_TRANSFORM_2D
Rotate and translate a segment instance.

GM_$TPLANEMASK
Segment: change plane mask.

GM_$TPOLYLINE
Draw a linked set of line segments.

GM_$TPRIMITIVE
Draw a primitve.

GM_$TRECTANGLE
Draw a rectangle.

GM_$TTAG
Insert a tag.

GM_$TTEXT_2D
Write a text string.

GM_$TTEXTBVALUE
Background value for text.

GM_$TTEXTSIZE
Size for text.

GM_$TTEXTVALUE
The pixel value for text.

GM_$CONC_MODE_T

A 2-byte integer. Defines the number of concurrent users a file may have. One of the following predefined values:

GM_$1W
N readers or 1 writer is allowed.

GM_$COWRITERS
More than 1 writer is allowed, but all must be on the same node.

GM_$CURSOR_PATTERN_T

A gm_$max_cursor_pattern_words-element array of 2-byte integers. Specifies the values that set the cursor pattern.

GM_$CURSOR_STYLE_T

A 2-byte integer. Specifies the type of cursor. One of the following predefined values:

GM_$BITMAP
Only value is bitmap.

GM_$CURVE_T

A 2-byte integer. Specifies the type of curve. One of the following predefined values:

GM_$SPLINE_CUBIC_P
Draw a smooth curve through n points.

GM_$ARC_3P
Draw an arc through three points.

GM_$DATA_TYPE_T

A 2-byte integer. Specifies the form in which to store data. One of the following predefined values:

GM_$16
Data is stored as 2-byte integers.

GM_$32
Data is stored as 4-byte integers.

GM_$REAL
Data is stored as 4-byte integers.

GM_$DISPLAY_CONFIG_T

A 2-byte integer. Returns the current display configuration. One of the following predefined values:

GM_$BW_800X1024
4-bit two-board black and white portrait.

GM_$BW_1024X800
4-bit two-board black and white landscape.

GM_$COLOR_1024X1024X4
4-bit two-board color configuration.

GM_$COLOR_1024X1024X8
8-bit three-board color configuration.

GM_$COLOR_1024X800X4
4-bit two-board color configuration.

GM_$COLOR_1024X800X8
8-bit three-board color configuration.

GM_$COLOR_1280X1024X8
8-bit two-board color configuration.

GM_$COLOR1_1024X800X8
8-bit two-board color configuration.

GM_$COLOR2_1024X800X4
4-bit single-board color configuration.

GM_$DISPLAY_MODE_T

A 2-byte integer. Specifies the mode of operation. One of the following predefined values:

GM_$BORROW
Uses the entire screen.

GM_$MAIN_BITMAP
Displays within a bitmap allocated in main memory.

GM_$DIRECT
Displays within a Display Manager window.

GM_$NO_BITMAP
Allows editing of files without display.

GM_$WITHIN_GPR
Displays the output of the metafile within a bitmap that you initialize using routines of DOMAIN graphics primitives.

GM_$DRAW_PATTERN_T

An array of up to gm_$max_draw_pattern_bytes characters. Specifies the bit pattern to use in drawing lines.

GM_$EVENT_T

A 2-byte integer. Specifies the type of input event; same as gpr_$event_t. One of the following predefined values:

GM_$KEYSTROKE
Returned when you type a keyboard character.

GM_$BUTTONS
Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
Returned when you move the mouse or bitpad puck or use the touchpad.

GM_$ENTERED_WINDOW
Returned when the cursor enters a window in which the GM bitmap resides. Direct mode only.

GM_$LEFT_WINDOW
Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode only.

GM_$LOCATOR_STOP
Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

GM_$FILL_PATTERN_T

A gm_$max_fill_pattern_lwords-element array of 4-byte integers. Specifies the pattern to use in filling areas.

GM_$FONT_TYPE_T

A 2-byte integer. Specifies the type of font. One of the following predefined values:

GM_$PIXEL
A font defined by pixels.

GM_$STROKE
A font defined by strokes.

GM_$GRID_STYLE_T

A 2-byte integer. Specifies the type of grid style to be displayed in a viewport. One of the following predefined values:

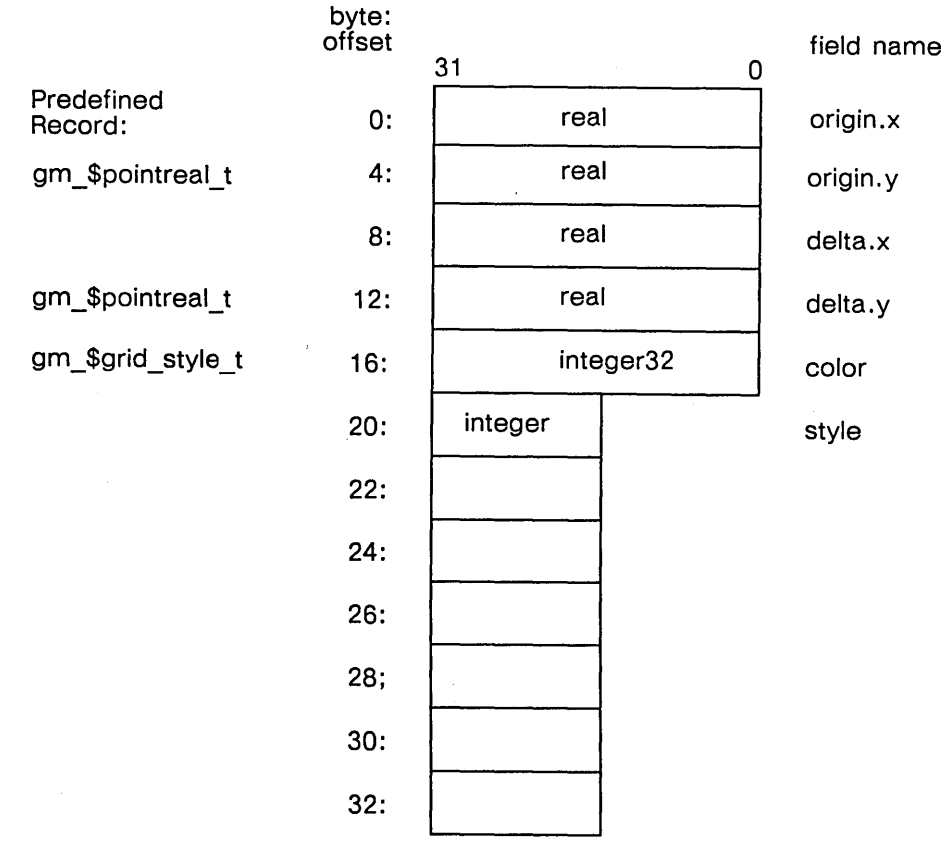
GM_$GRID_POINT
The grid intersections are shown by points.

GM_$GRID_CROSS
The grid intersections are shown by cross hairs.

GM_$GRID_BOX
The grid is shown as boxes

GM_$GRID_T

Specifies the characteristics of the grid. The diagram below illustrates the gm_$grid_t data type:



Field Description:

origin
The x and y coordinates of the origin of the grid in segment coordinates.

delta
The delta-x and delta-y of the grid in segment coordinates.

color
The color of the grid.

style
The style of the grid display. This begins the variant part of the data structure. The value of style determines which fields require information if you are establishing a grid, or which fields

contain information if you are inquiring about a grid.

If the value of style is gm_$grid_point, do not add or try to retrieve any more information.

If the value of style is gm_$grid_cross, insert or retrieve the cross width and and height in the next two integer fields (address 22 and 24).

Predefined Record: gm_$point16_t

byte: offset		field name
20:	integer	style
22:		cross_size.x
24:		cross_size.y

Field Description:

If the value of style is gm_$grid_box, insert or retrieve "prepeat" (line repetition factor) at address 22, "plength" (length of line pattern) at address 24, and "pattern" (line pattern) at byte addresses 26 - 33.

Predefined Record: gm_$draw_pattern_t

byte: offset		field name
20:	integer	style
22:	integer	prepeat
24:	integer	plength
26:		
28:		pattern
30:		
32:		

GM_$GRID_ARRAY_T

A list of grid specifications. See gm_$grid_t for a complete description and a diagram of one element of the array. This array holds up to gm_$max_grid elements.

GM_$GRID_FLAGS_T

A 2-byte integer. Specifies whether the snap grid is visible/invisible and/or whether snapping is

enabled. Any combination of the following predefined values:

GM_$GRID_VISIBLE
Indicates that the grid is visible.

GM_$GRID_SNAPTO
Indicates that the grid uses snapping.

GM_$HIGHLIGHT_T

A 2-byte integer. Specifies the type of highlighting. One of the following predefined values:

GM_$OUTLINE
Only value: Highlighting appears as an outline.

GM_$KEYSET_T

Specifies the set of characters that make up a keyset associated with the graphics input event types GM_$KEYSTROKE and GM_$BUTTONS. This is a 16-element array of 2-byte integers. For a FORTRAN subroutine to use in building a set of characters, see the routine GM_$INPUT_ENABLE in this volume.

GM_$LINE_STYLE_T

A 2-byte integer. Specifies the type of lines. One of the following predefined values:

GM_$SAME_LINE_STYLE
The line style does not change.

GM_$SOLID
The line style is solid.

GM_$DOTTED
The line style is dotted.

GM_$PATTERNED
The line style is patterned.

GM_$MODELCMD_MODE_T

A 2-byte integer. Specifies an editing mode for modeling commands. One of the following predefined values:

GM_$MODELCMD_INSERT
Modeling commands insert a command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_SET_FLAG = false.

GM_$MODELCMD_REPLACE
Modeling commands replace the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_SET_FLAG = true.

GM_$MODELCMD_RUBBERBAND
Modeling commands XOR the previous modeling command on the screen, thus erasing it, then XOR the given modeling command onto the screen. Only bitplane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

GM_$PLANE_LIST_T

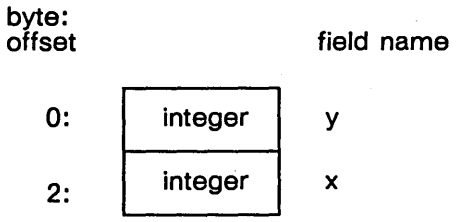
A 2-byte integer. Specifies a value between 0 and gm_$max_plane_id inclusive, depending on the type of node.

GM_$PLANE_MASK_T

A 2-byte integer. Specifies a set of planes from GM_$PLANE_LIST_T.

GM_$POINT16_T

Specifies the X- and Y- coordinates of a point. The diagram below illustrates the gm_$point16_t data type:



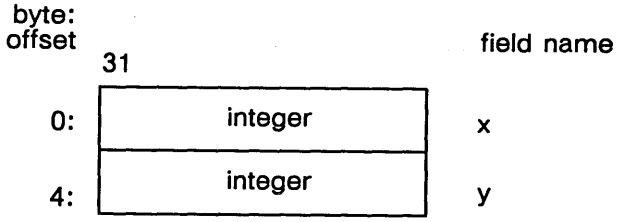
Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINT32_T

Specifies the X- and Y-coordinates of a point. The diagram below illustrates the gm_$point32_t data type:



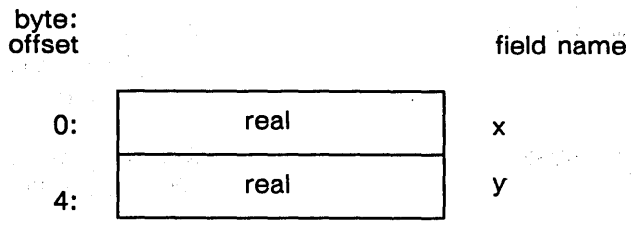
Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINTREAL_T

Specifies the X- and Y-coordinates of a point. The diagram below illustrates the gm_$pointreal_t data type:



Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINT_ARRAY16_T

An array of GM_$POINT16_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINT_16_T illustrates a single element.

GM_$POINT_ARRAY32_T

An array of GM_$POINT32_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINT32_T illustrates a single element.

GM_$POINT_ARRAYREAL_T

An array of GM_$POINTREAL_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINTREAL_T illustrates a single element.

GM_$PRIMITIVE_PTR_T

Pointer to procedure for user-defined primitive, with the following argument protocol:

```
in N_POINTS            2-byte integer
in POINTS     array of GM_$POINT16_T,
in N_PARAMETERS        2-byte integer
in PARAMETERS array of GM_$POINTREAL_T
out STATUS             status_$T
```

GM_$PRINT_STYLE_T

A 2-byte integer. Specifies the type of output. One of the following predefined values:

GM_$GMF
Output is a graphics map file.

GM_$OUT1
Output is a vector command file.

GM_$POSTSCRIPT
Output file is a PostScript file.

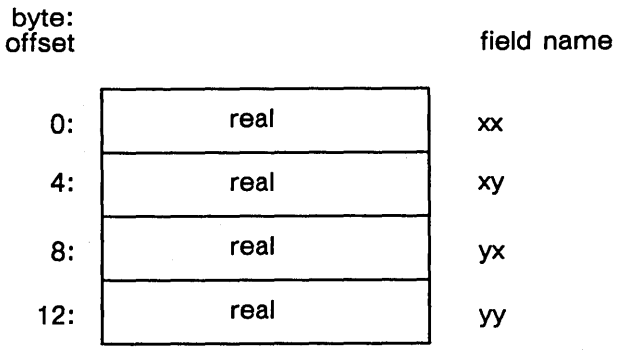
GM_$REFRESH_PTR_T

Pointer to procedure for refreshing windows, with the following argument protocol:

```
in UNOBSCURED : boolean
in POS_CHANGE : boolean
```

GM_$ROTATE_REAL2X2_T

Specifies x- and y-coordinates for rotation. The diagram below illustrates the gm_$rotate_real2x2_t data type:



Field Description:

xx
The xx-coordinates for rotation.

xy
The xy-coordinates for rotation.

yx
The yx-coordinates for rotation.

yy
The yy-coordinates for rotation.

GM_$SEARCH_COMMAND_T

A 2-byte integer. Specifies the steps of a command search. One of the following predefined values:

GM_$CNEXT
Find the next command which falls within the pick aperture, moving forward in the segment.

GM_$STEP
Find the next command in the segment, independent of the pick aperture.

GM_$START
Move to the start of the segment, independent of the coordinates of the pick aperture.

GM_$END
Move to the end of the segment, independent of the coordinates of the pick aperture.

GM_$SEARCH_SEGMENT_T

A 2-byte integer. Specifies the steps of a segment search. One of the following predefined values:

GM_$SETUP
Make the top segment of the current viewport the start of the list of picked segments. The rest of the list is emptied.

GM_$DOWN
Find the first segment instanced by the current segment, which when instanced falls within the pick aperture.

GM_$NEXT
Find the next segment within the segment one higher in the list of picked segments, which falls within the pick aperture.

GM_$UP
Move up one level in the list of picked segments.

GM_$TOP
Proceed to top segment in the list of picked segments, destroying the rest of the list of picked segments.

GM_$CLEAR
Clear the entire list of picked segments, allowing all segments to be edited or deleted.

GM_$BOTTOM
Perform GM_$DOWN repeatedly until a segment is reached for which GM_$DOWN finds nothing.

GM_$NEXTBOTTOM
Perform GM_$BOTTOM. If nothing is found, perform GM_$NEXT until a segment

is found. An alternative to GM_$NEXT is one or more uses of GM_$UP followed by a GM_$NEXT. When a GM_$NEXT finds a segment, perform a GM_$BOTTOM from there.

GM_$SEGMENT_ID_T

A 4-byte integer. Specifies a value between 0 and gm_$max_segment_id inclusive.

GM_$SEGMENT_ID_LIST_T

Specifies an array of GM_$SEGMENT_ID_T with MAX_ARRAY_LENGTH elements.

GM_$STRING_T

An array of up to 256 characters. Specifies a string of characters.

GM_$VIEW_REFRESH

A 2-byte integer. Specifies the refresh state of the viewport. One of the following predefined values:

GM_$REFRESH_INHIBIT
When you change commands in the file, the viewport is rewritten when you call GM_$VIEWPORT_REFRESH. GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state. Other conditions are mode-dependent.

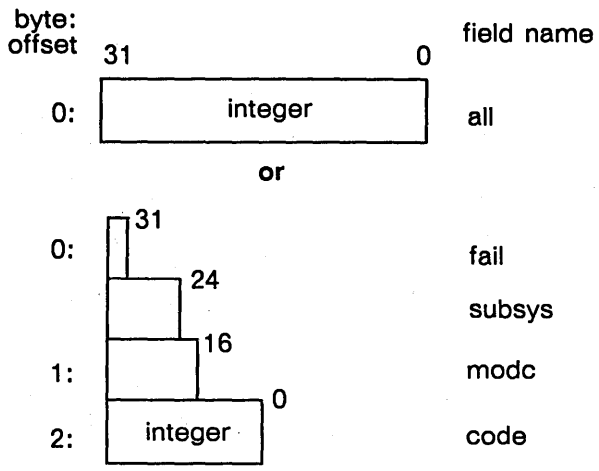
GM_$REFRESH_WAIT
When you change commands in the file, the viewport is rewritten when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. Other conditions are mode-dependent.

GM_$REFRESH_UPDATE
Every time you change any command in the file, this viewport is completely corrected if it is the current viewport.

GM_$REFRESH_PARTIAL
Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

GM_$ABLOCK_ASSIGN_DISPLAY

Assigns an attribute block (by number) to an attribute class, for the entire display.

FORMAT

```
GM_$ABLOCK_ASSIGN_DISPLAY (aclass_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to which the attribute block will be assigned. This is a 2-byte integer.

ablock_id
The identification number of the attribute block to be assigned to the attribute class. This is a 2-byte integer.

To assign the default attributes to an attribute class for the display, use ablock_id = 1.

To undo the asssignment of an attribute block to an attribute class for the display, use ablock_id = 0.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_ASSIGN_DISPLAY to assign an existing attribute block to an attribute class for all viewports in the display.

Use GM_$ABLOCK_INQ_ASSIGN_DISPLAY to inquire about the current attribute block number assigned to a particular class for the display.

Assignments of attribute blocks to attribute classes for individual viewports using GM_$ABLOCK_ASSIGN_VIEWPORT will override assignments made by GM_$ABLOCK_ASSIGN_DISPLAY.

GM_$ABLOCK_ASSIGN_VIEWPORT

Assigns an attribute block (by number) to an attribute class, for one viewport.

FORMAT

```
GM_$ABLOCK_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to which the attribute block will be assigned. This is a 2-byte integer.

To assign the default attributes to an attribute class for one viewport, use ablock_id = 1.

viewport_id
The identification number of the viewport in which to assign the attribute block to the attribute class. This is a 2-byte integer.

ablock_id
The identification number of the attribute block to be assigned to the attribute class. This is a 2-byte integer.

To undo the asssignment of an attribute block to an attribute class for one viewport, use ablock_id = 0.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_ASSIGN_VIEWPORT to assign an existing attribute block to an attribute class for one viewport in the display.

Use GM_$ABLOCK_INQ_ASSIGN_VIEWPORT to inquire about the current attribute block number assigned to a particular class for a particular viewport.

Assignments of attribute blocks to attribute classes for individual viewports using GM_$ABLOCK_ASSIGN_VIEWPORT will override assignments made by GM_$ABLOCK_ASSIGN_DISPLAY.

GM_$ABLOCK_COPY

Copies all attributes from one existing attribute block to another.

FORMAT

```
GM_$ABLOCK_COPY (source_ablock_id, destination_ablock_id, status)
```

INPUT PARAMETERS

source_ablock_id
The identification number of the existing attribute block from which attributes will be copied. This is a 2-byte integer.

destination_ablock_id
The identification number of the existing attribute block to which the attributes of the attribute block source_block_id will be copied. This is a 2-byte integer.

You may not copy attributes into attribute blocks 0 and 1 (default). Attribute block 0 is a list of no-change attribute values; attribute block 1 is a list of default attribute values.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_CREATE to establish a new attribute block identical to an existing one. Use GM_$ABLOCK_COPY to copy attributes from an existing attribute block to an existing one.

GM_$ABLOCK_CREATE

Creates an attribute block and initializes it equivalent to an existing block.

FORMAT

```
GM_$ABLOCK_CREATE (source_ablock_id, ablock_id, status)
```

INPUT PARAMETERS

source_ablock_id
The identification number of the existing attribute block used as the source for the block generated with GM_$ABLOCK_CREATE. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id
The identification number assigned to the attribute block generated by GM_$ABLOCK_CREATE. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_CREATE to establish a new attribute block identical to an existing one. Use GM_$ABLOCK_COPY to copy attributes from an existing attribute block to an existing one.

Currently, you are limited to 10 attribute blocks, including the two preassigned ones.

GM_$ABLOCK_INQ_ASSIGN_DISPLAY

Returns the current attribute block number assigned to a particular attribute class for the display.

FORMAT

```
GM_$ABLOCK_INQ_ASSIGN_DISPLAY (aclass_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class for which to return the current attribute block assignment. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id
The identification number of the attribute block currently assigned to the specified attribute class for the display. This is a 2-byte integer.

If you have not assigned an attribute block to the specified attribute class for the display, the returned value is 0 (no assignment).

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_ASSIGN_DISPLAY to assign an attribute block to a display.

GM_$ABLOCK_INQ_ASSIGN_VIEWPORT

Returns the current attribute block number assigned to a particular attribute class for one viewport.

FORMAT

```
GM_$ABLOCK_INQ_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class for which to return the current attribute block assignment. This is a 2-byte integer.

viewport_id
The identification number of the viewport for which to return the current attribute block identification number. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id
The identification number of the attribute block assigned to the attribute class for the display. This is a 2-byte integer.

If you have not assigned an attribute block to the special attribute class for the display, the returned value is 0 (no assignment).

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_ASSIGN_VIEWPORT to assign an attribute block to a viewport.

GM_$ABLOCK_INQ_DRAW_RASTER_OP

Returns the raster operation code for drawing lines for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_RASTER_OP (ablock_id, raster_op, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the raster operation codes. This is a 2-byte integer.

OUTPUT PARAMETERS

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15, or -1. The default value is 3. This sets all destination bit values to source bit values.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_DRAW_RASTER_OP to change the draw raster operation code in an attribute block.

GM_$ABLOCK_INQ_DRAW_STYLE

Returns the line style set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                           pattern_length, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the drawing style. This is a 2-byte integer.

OUTPUT PARAMETERS

style
The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. One of the following values is returned:

GM_$SOLID Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is GM_$SOLID.

GM_$DOTTED Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.

GM_$PATTERNED
Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.

GM_$SAME_DRAW_STYLE
Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor
The number of times each bit in this pattern is replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. The replication factor changes the scaling applied to the pattern.

pattern
The bit pattern, in GM_$DRAW_PATTERN_T format. This is an array of 8 bytes constituting a 64-bit pattern. Only the bits specified in the pattern-length parameter are used.

pattern_length
The length of the bit pattern, in bits. This is a 2-byte integer. The returned values range from 1 to 64.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_DRAW_STYLE to change the line style in an attribute block.

GM_$ABLOCK_INQ_DRAW_VALUE

Returns the value for drawing lines set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the drawing value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The line drawing value. This is a 4-byte integer. The default draw value is 1.

A value of -1 means that when this attribute block is selected, the draw value is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_DRAW_VALUE to change the line drawing value in an attribute block. The effect is influenced by the plane mask and the raster op.

GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE

Returns the background value for filling areas in the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the fill background value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The fill background value of the specified attribute block. This is a 4-byte integer. The default value is -2, the same as the viewport background.

The value -1 means that fill background pixels are to be left unchanged; that is, the fill background is "transparent."

The value -3 means that when this attribute block is selected, the fill background value is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                    |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_INQ_FILL_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE to change the fill value in an attribute block.

GM_$ABLOCK_INQ_FILL_PATTERN

Returns the pattern set for filling areas for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_PATTERN (ablock_id, scale, size, pattern, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the fill pattern. This is a 2-byte integer.

OUTPUT PARAMETERS

scale
The number of times each bit in this pattern is to be replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer.

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32.

pattern
The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_FILL_PATTERN to change the fill pattern in an attribute block.

GM_$ABLOCK_INQ_FILL_VALUE

Returns the value set for filling areas for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablockid
The identification number of the attribute block for which to return the fill value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The value for filling areas. This is a 4-byte integer. The default fill value is 1.

A value of -1 indicates that when this attribute block is selected, the fill value is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_FILL_VALUE to change the fill value in an attribute block.

GM_$ABLOCK_INQ_FONT_FAMILY

Returns the font family identification number set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FONT_FAMILY (ablock_id, font_family_id, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text font family. This is a 2-byte integer.

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default value is 1.

A value of -1 indicates that when this attribute block is selected, the font family is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_FONT_FAMILY to change the text font family identification in this attribute block.

GM_$ABLOCK_INQ_PLANE_MASK

Returns the value of the plane mask set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_PLANE_MASK (ablock_id, change, mask, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the plane mask values. This is a 2-byte integer.

OUTPUT PARAMETERS

change
A Boolean (logical) variable that indicates whether the plane mask is to be changed when the specified attribute block is selected. When true, the plane mask is to be changed to "mask." A value of change = false means that when this attribute block is selected, the plane mask is not to be changed. In this case, the value of mask is undefined.

mask
The plane mask, specifying which planes are currently in use, in GM_$PLANE_MASK_T format. This is a 2-byte integer. This value may be any combination of the set of integer values from 0 to 7. Each integer corresponds to a plane in use. For example, if 0 and 7 are set, planes 0 and 7 are in use. The default is that all planes are in use and can be modified.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Operations can occur only on the planes specified in the mask. A program can use this routine, for example, to perform drawing operations only into certain planes in the bitmap.

Use GM_$ABLOCK_SET_PLANE_MASK to set the plane mask in an attribute block.

GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE

Returns the text background value set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text background value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The value to use for the text background in this attribute block. This is a 4-byte integer.

The default text background value is -2. This specifies that the viewport background value is used as the text background. For borrowed displays and main memory bitmaps, this is always 0.

A value from 0 to 255 means to use that value.

-1 means that text background pixels are to be left unchanged; that is, the text background is "transparent."

-3 means that when this attribute block is selected, the text background value is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_INQ_TEXT_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE to set the text background value in an attribute block.

GM_$ABLOCK_INQ_TEXT_SIZE

Returns the size of text set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_SIZE (ablock_id, size, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text size. This is a 2-byte integer.

OUTPUT PARAMETERS

size
The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value. The default text size is 10.0.

A value of -1 indicates that when this attribute block is selected, the text size is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The choice of a font from a font family is based on the specified text size. The largest font in the font family that does not exceed the text size is used. The size of a font is defined as the largest ascender height of any character in the font; the descender is ignored.

Use GM_$ABLOCK_SET_TEXT_SIZE to set the text size in an attribute block.

GM_$ABLOCK_INQ_TEXT_VALUE

Returns the value for writing text for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The value to use for writing text. This is a 4-byte integer. The default text value is 1.

A value of -1 indicates that when this attribute block is selected, the text value is not to be changed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_SET_TEXT_VALUE to set the text value in an attribute block.

GM_$ABLOCK_SET_DRAW_RASTER_OP

Changes the raster operation code for drawing lines for this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_RASTER_OP (ablock_id, raster_op, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the drawing style. This is a 2-byte integer.

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15. The default value is 3. This sets all destination bit values to source bit values.

Assigning the value = -1 means that when this attribute block is selected, the draw raster op value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_DRAW_RASTER_OP to retrieve the current raster operations in an attribute block.

GM_$ABLOCK_SET_DRAW_STYLE

Changes the value of the line style in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                           pattern_length, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the drawing style. This is a 2-byte integer.

style
The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SOLID Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is GM_$SOLID.

GM_$DOTTED Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.

GM_$PATTERNED
Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.

GM_$SAME_DRAW_STYLE
Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor
The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, repeat_factor is ignored and assumed to be 1.

pattern
The bit pattern, in GM_$DRAW_PATTERN_T format. This is an array of 8 bytes constituting a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length
The length of the bit pattern, in bits. This is a 2-byte integer. Currently, pattern_length is ignored and assumed to be 64.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The following defines a line pattern with dashes and spaces, twelve and four pixels long, respectively:

```
pattern          : STATIC gm_$draw_pattern_t :=
    [ CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    ];
```

Use GM_$ABLOCK_INQ_DRAW_STYLE to retrieve the current line style.

GM_$ABLOCK_SET_DRAW_VALUE

Changes the value for drawing lines in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the drawing value. This is a 2-byte integer.

value
The value to use in drawing lines. This is a 4-byte integer. The default value is 1.

Assigning the value = -1 means that when this attribute block is selected, the draw value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_DRAW_VALUE to retrieve the current draw value in an attribute block.

GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE

Changes the background value for filling areas in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill background value. This is a 2-byte integer.

value
The fill background value to use in the specified attribute block. This is a 4-byte integer. The default value is -2, the same as the viewport background.

Assigning a value from 0 to 255 means to use that value.

Assigning a value of -1 means that fill background pixels are to be left unchanged; that is, the fill background is "transparent."

Assigning the value = -3 means that when this attribute block is selected, the fill background value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_SET_FILL_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE to retrieve the current fill background value in an attribute block.

GM_$ABLOCK_SET_FILL_PATTERN

Changes the fill pattern in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_PATTERN (ablock_id, scale, size, pattern, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill pattern. This is a 2-byte integer.

scale
The number of times each bit in this pattern is to be replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern), 0 (when clearing a pattern), or -1 (when specifying "no change").

A value scale = 0 indicates that filled areas are to be filled with a solid color and that the pattern is to be ignored. In this case, the fill value is assigned to every pixel in the interior of the specified area.

Assigning the value scale = -1 means that when this attribute block is selected, the fill pattern is not to be changed.

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern
The 32 x 32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_FILL_PATTERN to retrieve the current fill pattern in an attribute block.

GM_$ABLOCK_SET_FILL_VALUE

Changes the value for filling areas in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill value. This is a 2-byte integer.

value
The value for filling areas in the specified attribute block. This is a 4-byte integer. The default value is 1.

Assigning the value = -1 means that when this attribute block is selected, the fill value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_FILL_VALUE to retrieve the current fill value in an attribute block.

GM_$ABLOCK_SET_FONT_FAMILY

Changes the font family in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FONT_FAMILY (ablock_id, font_family_id, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text font family. This is a 2-byte integer.

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default text font family identification number is 1.

Assigning value = -1 means that when this attribute block is selected, the font family is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_FONT_FAMILY to retrieve the current text font family identification in an attribute block.

Use GM_$FONT_FAMILY_INQ_ID to retrieve the identification number of a font family for which you know the name.

GM_$ABLOCK_SET_PLANE_MASK

Changes the value of the plane mask in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_PLANE_MASK (ablock_id, change, mask, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the plane mask. This is a 2-byte integer.

change
A Boolean (logical) variable that indicates whether the plane mask is to be changed when the specified attribute block is selected. When change is set to true, the plane mask is to be changed to "mask". Assigning change = false means that when this attribute block is selected, the plane mask is not to be changed.

mask
The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer.

The default value is [0...7], in GM_$PLANE_MASK_T format, or 255 when expressed as a 2-byte integer. The default is that all planes are in use and can be modified.

FORTRAN programmers should encode the plane mask in a 2-byte integer in the range of 0-255 (1 means plane 0 is on, 2 means plane 1 is on, 3 means planes 0 and 1 are on, 255 means planes 0 through 7 are on).

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Operations can occur only on the planes specified in the mask. A program can use this routine, for example, to perform drawing operations only into certain planes in the bitmap.

Use GM_$ABLOCK_INQ_PLANE_MASK to retrieve the current plane mask.

FORTRAN programmers might want to include the parameter definitions given below:

```
 integer*2
+    bit0,
+    bit1,
+    bit2,
+    bit3,
+    bit4,
+    bit5,
+    bit6,
+    bit7
 parameter (
+    bit0  16#0001,
+    bit1  16#0002,
+    bit2  16#0004,
+    bit3  16#0008,
+    bit4  16#0010,
+    bit5  16#0020,
+    bit6  16#0040,
+    bit7  16#0080)
```

Example:

In FORTRAN, to enable planes 2 and 5, use the following:

```
CALL GM_$PLANE_MASK( bit2 + bit5, status )
```

In Pascal, to enable planes 2 and 5, use the following:

```
GM_$PLANE_MASK( [ 2, 5 ], status )
```

GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE

Changes the background value for text in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE (ablock_id, value,status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text background value. This is a 2-byte integer.

value
The value to use for the text background in this attribute block. This is a 4-byte integer.

The default text background value is -2. This specifies that the viewport background value is used as the text background. For borrowed displays and main memory bitmaps, this is always 0.

Assigning a value from 0 to 255 means to use that value.

Assigning a value of -1 means that text background pixels are to be left unchanged; that is, the text background is "transparent."

Assigning the value = -3 means that when this attribute block is selected, the text background value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012  | 34567890
----------------------------------|---------
GM_$ABLOCK_SET_TEXT_BACKGROUND_V  | ALUE
```

Use GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE to retrieve the current text background value in an attribute block.

GM_$ABLOCK_SET_TEXT_SIZE

Changes the size of text in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_SIZE (ablock_id, size, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text size. This is a 2-byte integer.

size
The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value. The default text size is 10.0.

The value of -1 indicates that when this attribute block is selected, the text size is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The choice of a font from a family is based on the specified text size. The largest font in the family that does not exceed this height is used. The size of a font is defined as the largest ascender height on any character in the font; descender sizes are ignored.

Use GM_$ABLOCK_INQ_TEXT_SIZE to retrieve the current text size in an attribute block.

GM_$ABLOCK_SET_TEXT_VALUE

Changes the value for writing text set for this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text value. This is a 2-byte integer.

value
The value to use for writing text. This is a 4-byte integer. The default text value is 1.

Assigning the value = -1 means that when this attribute block is selected, the text value is not to be changed.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$ABLOCK_INQ_TEXT_VALUE to retrieve the current text value in an attribute block.

GM_$ACLASS

Inserts a command into the current segment: change to a different attribute class.

FORMAT

```
GM_$ACLASS (aclass_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to use. This is 2-byte integer.

The maximum number of attribute classes is 16.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$CIRCLE_[16,32,REAL]

Inserts a command into the current segment: draw a circle.

FORMAT

```
GM_$CIRCLE_16 (center, radius, fill, status)

GM_$CIRCLE_32 (center, radius, fill, status)

GM_$CIRCLE_REAL (center, radius, fill, status)
```

INPUT PARAMETERS

center

The point that is the center of the circle. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$INQ_CIRCLE_16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$INQ_CIRCLE_32

GM_$POINTREAL
: A two-element array of real values for GM_$INQ_CIRCLE_REAL

See the GM_$ Data Types section for more information.

radius

The radius of the circle, in the appropriate format:

A 2-byte integer for GM_$CIRCLE_16

A 4-byte integer for GM_$CIRCLE_32

A real value for GM_$CIRCLE_REAL

fill

A Boolean (logical) value which specifies whether to fill the circle.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_CIRCLE_[16,32,REAL] to retrieve the parameters of a circle command inserted by GM_$CIRCLE_[16,32,REAL].

Circles may be scaled, rotated, and/or reflected. However, when you apply a transform in which one axis is stretched more than another, you get a circle of undefined size, not a distorted circle.

Before supplying coordinate data to GM_$CIRCLE_REAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$COMMAND_DELETE

Deletes the current command.

FORMAT

```
GM_$COMMAND_DELETE (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

After you delete the current command, the command before it in the current segment becomes the current command.

Use GM_$PICK_COMMAND to change the current command.

GM_$COMMAND_INQ_BOUNDS

Returns the bounds of the current command in the current segment.

FORMAT

```
GM_$COMMAND_INQ_BOUNDS (bounds,status)
```

OUTPUT PARAMETERS

bounds
Bounds of the command in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use this call to obtain the bottom left-hand and top right-hand coordinates of the current command in the current segment.

Use GM_$SEGMENT_INQ_BOUNDS to obtain the bounds of a segment.

Use GM_$FILE_INQ_BOUNDS to obtain the bounds of the primary segment in a file.

GM_$COORD_BITMAP_TO_SEG_2D

Converts bitmap coordinates to segment coordinates.

FORMAT

```
GM_$COORD_BITMAP_TO_SEG_2D (bitmap_position, segment_position,status)
```

INPUT PARAMETERS

bitmap_position
The bitmap coordinates to be converted to segment coordinates, expressed as an (x,y) pair in terms of a fraction of the GM bitmap in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

segment_position
The converted segment coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This routine converts the bitmap coordinates to segment coordinates of the current segment in the current viewport.

In within-GPR mode, use GM_$COORD_PIXEL_TO_SEG_2D.

This routine is commonly used immediately after GM_$INPUT_EVENT_WAIT to convert a GM bitmap location to segment coordinates, so that a new command can be inserted into the metafile. For example:

```
GM_$INPUT_EVENT_WAIT(false, etype, edata, bitmap_pos,
                     viewport_id, status);

GM_$COORD_BITMAP_TO_SEG_2D (bitmap_pos, segment_pos, status);

GM_$CIRCLE_REAL (segment_pos, radius, false, status);
```

GM_$COORD_PIXEL_TO_SEG_2D

Converts GPR bitmap coordinates used in within-GPR mode to segment coordinates, using a specified transformation.

FORMAT

```
GM_$COORD_PIXEL_TO_SEG_2D (rotate, translate, pixel_position,
                           segment_position, status)
```

INPUT PARAMETERS

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

pixel_position

The pixel coordinates to be converted to segment coordinates, expressed as an (x,y) pair, in GM_$POINT16_T format. This is a 2-byte integer array of two elements. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

segment_position

The converted segment coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

In modes other than within-GPR mode, use GM_$COORD_BITMAP_TO_SEG_2D.

GM_$COORD_SEG_TO_BITMAP_2D

Converts segment coordinates to bitmap coordinates.

FORMAT

```
GM_$COORD_SEG_TO_BITMAP_2D (segment_position, bitmap_position, status)
```

INPUT PARAMETERS

segment_position
The segment coordinates to be converted to bitmap coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of a fraction of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This routine converts the segment coordinates of the current segment in the current viewport to bitmap coordinates.

In within-GPR mode, use GM_$COORD_SEG_TO_PIXEL_2D.

GM_$COORD_SEG_TO_PIXEL_2D

Converts within-GPR segment coordinates to GPR bitmap coordinates, using a specified transformation.

FORMAT

```
GM_$COORD_SEG_TO_PIXEL_2D (rotate, translate, segment_position,
                           pixel_position, status)
```

INPUT PARAMETERS

question

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

segment_position

The segment coordinates to be converted to pixel coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

pixel_position

The converted pixel coordinates expressed as an (x,y) pair, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

In modes other than within-GPR mode, use GM_$COORD_SEG_TO_BITMAP_2D.

GM_$CURSOR_INQ_ACTIVE

Returns the status of the cursor: displayed or not displayed.

FORMAT

```
GM_$CURSOR_INQ_ACTIVE (active, status)
```

OUTPUT PARAMETERS

active
A Boolean (logical) value that indicates whether or not the cursor is displayed. The parameter is set to true if the cursor is displayed; it is set to false if the cursor is not displayed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

GM_$CURSOR_INQ_PATTERN

Returns the type, pattern, and origin of the cursor.

FORMAT

```
GM_$CURSOR_INQ_PATTERN (style, pattern_size, pattern, origin, status)
```

OUTPUT PARAMETERS

style
The cursor style, in GM_$CURSOR_STYLE_T format. This is a 2-byte integer. Currently, the only valid value is GM_$BITMAP.

pattern_size
The size of the cursor pattern, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, neither coordinate size may exceed 16. See the GM_$ Data Types section for more information.

pattern
The cursor pattern, in GM_$CURSOR_PATTERN_T format. This is an array of (pattern_size.y) 2-byte integers. The length of the array is determined by the y value of pattern_size.

origin
The offset from the pixel at the upper left of the cursor to the pixel at the origin of the cursor, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

When the cursor is moved using GM_$CURSOR_SET_POSITION, the pixel that is the cursor's origin is placed at the specified location.

The first element (x) indicates the number of cursor pixels that will be displayed to the left of the specified cursor location. The second element (y) indicates the number of cursor pixels that will be displayed above the specified cursor location. Both numbers must be between 0 and 15; only the first four bits are considered.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

GM_$CURSOR_INQ_POSITION

Returns the position of the cursor.

FORMAT

```
GM_$CURSOR_INQ_POSITION (bitmap_position, status)
```

OUTPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of fractions of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

GM_$CURSOR_SET_ACTIVE

Specifies whether or not the cursor is displayed.

FORMAT

```
GM_$CURSOR_SET_ACTIVE (active, status)
```

INPUT PARAMETERS

active

A Boolean (logical) value that indicates whether or not the cursor is displayed. The parameter is set to true if the cursor is displayed; it is set to false if the cursor is not displayed.

The default value for active is false.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$CURSOR_INQ_ACTIVE to retrieve the display status of the cursor.

GM_$CURSOR_SET_PATTERN

Specifies a cursor pattern, type, and origin.

FORMAT

```
GM_$CURSOR_SET_PATTERN (style, pattern_size, pattern, origin, status)
```

INPUT PARAMETERS

style
The cursor style, in GM_$CURSOR_STYLE_T format. Currently, the only valid value is GM_$BITMAP.

pattern_size
The size of the cursor pattern, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, neither coordinate size may exceed 16. See the GM_$ Data Types section for more information.

pattern
The cursor pattern, in GM_$CURSOR_PATTERN_T format. This is an array of (pattern_size.y) 2-byte integers. The length of the array is determined by the y value of pattern_size.

The default cursor uses the standard Display Manager pattern.

origin
The offset from the pixel at the upper left of the cursor to the pixel at the origin of the cursor, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

When the cursor is moved using GM_$CURSOR_SET_POSITION, the pixel that is the cursor's origin is placed at the specified location.

The first element (x) indicates the number of cursor pixels that will be displayed to the left of the specified cursor location. The second element (y) indicates the number of cursor pixels that will be displayed above the specified cursor location. Both numbers must be between 0 and 15; only the first four bits are considered.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The default value is the standard Display Manager pattern.

Use GM_$CURSOR_INQ_PATTERN to retrieve the current pattern of the cursor.

You must place a cursor pattern smaller than 16x16 in the high-order bits of the first words of the pattern:

```
VAR
{ note that a cursor pattern smaller than 16x16
  starts in the high order bits, and starts
  in word 1 of the array }

cursor_pattern1 : gm_$cursor_pattern_t
                := [16#8080,16#4100,16#2200,16#1400,
                    16#800,16#1400,16#2200,16#4100,16#8080];
cursor_size : gm_$point16_t := [9,9];
cursor_origin : gm_$point16_t := [4,4];

.
.
.

gm_$cursor_set_pattern(gm_$bitmap,cursor_size,
                       cursor_pattern1,cursor_origin, status);
```

GM_$CURSOR_SET_POSITION

Moves the cursor on the screen.

FORMAT

```
GM_$CURSOR_SET_POSITION (bitmap_position, status)
```

INPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of fractions of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$CURSOR_INQ_POSITION to retrieve the current position of the cursor.

GM_$CURVE_2D[16,32,REAL]

Inserts a command into the current segment: draw a curve.

FORMAT

```
GM_$CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                  parameter_array, status)
```

INPUT PARAMETERS

curve_type
The type of curve to be drawn, in GM_$CURVE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$ARC_3P Specifies an arc to be drawn through three points (n_points) in the point array (point_array). The value for n_points must equal 3.

GM_$SPLINE_CUBIC_P
Specifies a smooth curve (parametric cubic spline) to be drawn through the specified number of point (n_points) in the point array (point_array).

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinate points, each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$CURVE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$CURVE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$CURVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters. This is an array of real values.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Currently, n_parameters and parameter_array are not used.

Use GM_$INQ_CURVE_2D[16,32,REAL] to retrieve the parameters of a curve command inserted by GM_$CURVE_2D[16,32,REAL].

Curves are limited to 1000 (GM_$MAX_ARRAY_LENGTH) points.

Before supplying coordinate data to GM_$CURVE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$DATA_COERCE_INQ_REAL

Returns the data type to which real coordinates are converted.

FORMAT

```
GM_$DATA_COERCE_INQ_REAL (data_type, status)
```

OUTPUT PARAMETERS

data_type

The form in which to store data, in GM_$DATA_TYPE_T format. This is a 2-byte integer. Data sent to the package as real variables can be stored in another form. Currently, the only valid value is GM_$32.

You must set the data type to GM_$32 because real data must be coerced to 32-bit data before it can be stored.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$DATA_COERCE_SET_REAL to force real variables that you send to the package to be stored in another form.

GM_$DATA_COERCE_SET_REAL

Specifies the data type to which subsequent real coordinates are converted.

FORMAT

```
GM_$DATA_COERCE_SET_REAL (data_type, status)
```

INPUT PARAMETERS

data_type

The form in which to store data, in GM_$DATA_TYPE_T format. This is a 2-byte integer. Data sent to the package as real variables can be stored in another form. Currently, the only valid value is GM_$32.

You must set the data type to GM_$32 because real data must be coerced to 32-bit data before it can be stored.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$DATA_COERCE_INQ_REAL to retrieve the data type to which real coordinate data is to be coerced.

Currently, supplying real coordinate data before calling this routine is an error.

GM_$DISPLAY_FILE

Displays the entire current file in the current viewport.

FORMAT

```
GM_$DISPLAY_FILE (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This command changes the view transformation to a value which will cause the entire metafile to be displayed as follows: one of the two dimensions fills 95 percent of the current viewport, and the other dimension fills less than or equal to 95 percent of the current viewport.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_FILE_PART

Displays part of the current file in the current viewport.

FORMAT

```
GM_$DISPLAY_FILE_PART (bounds, status)
```

INPUT PARAMETERS

bounds

The part of the primary segment of this file to be displayed, in terms of segment coordinates. This is a four-element array of real numbers (xmin, ymin, xmax, ymax), in GM_$BOUNDSREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This command sets the view transformation to a value which causes the specified part of the file to be displayed as follows: one of the two dimensions fills the viewport, and the other dimension does not overflow the viewport.

The GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_INQ_COLOR_MAP

Returns the values in the display color map.

FORMAT

```
GM_$DISPLAY_INQ_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry to be read. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-plane configuration
1 - 256	For color displays in 8-plane configuration

OUTPUT PARAMETERS

values
Color value entries, in GM_$COLOR_VECTOR_T format. This is an array of real values. The array must be at least (3 * n_entries) in length.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$DISPLAY_SET_COLOR_MAP to change the value of the display color map.

GM_$DISPLAY_REFRESH

Redisplays all uninhibited viewports of the display.

FORMAT

```
GM_$DISPLAY_REFRESH (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Viewports that are in the GM_$REFRESH_INHIBIT refresh state are not displayed.

GM_$DISPLAY_SEGMENT

Displays the specified segment (and all called segments) in the current viewport.

FORMAT

```
GM_$DISPLAY_SEGMENT (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This command changes the view transformation to a value which will cause the entire segment to be displayed as follows: one of the two dimensions fills 95 percent of the current viewport, and the other dimension fills less than or equal to 95 percent of the current viewport.

Use GM_$DISPLAY_FILE to display the entire file.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_SEGMENT_GPR_2D

In within-GPR mode, allows you to display a segment within a GPR bitmap.

FORMAT

```
GM_$DISPLAY_SEGMENT_GPR_2D (segment_id, rotate, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

You must specify the transform which relates segment coordinates of your selected segment to display coordinates (GPR coordinates). This is specified as a 2x2 rotation to be applied to coordinates in the segment, then a translation to be applied after this rotation.

If you have put data into segments with y pointing up, you will have to insert negative values into your transform.

The attributes you are currently using in your current GPR attribute block are used, until modified by attribute commands in the file.

In direct mode, a program must acquire the display before calling GM_$DISPLAY_SEGMENT_GPR_2D. The graphics metafile package will not acquire the display.

Rotation: Use the following to display segment little_seg at (400,300), at triple size and rotated 50 degrees:

```
rotate.xx := 3.0 * cos(50.0 * 3.14159/180.0);
rotate.xy := 3.0 * sin(50.0 * 3.14159/180.0);
rotate.yx := -rotate.xy;
rotate.yy := rotate.xx;
rpoint.x := 400.0;
rpoint.y := 300.0;
GM_$DISPLAY_SEGMENT_GPR_2D(little_seg, rotate,
                           rpoint, status);
```

Distortion: Use the following to display segment distort_seg at (12.5, 14.5), with a scale of 1 in the x direction and a scale of 3 in the y direction, unrotated:

```
rotate.xx := 1.0;
rotate.xy := 0.0;
rotate.yx := 0.0;
rotate.yy := 3.0;
rpoint.x  := 12.5;
rpoint.y  := 14.5;
GM_$DISPLAY_SEGMENT_GPR_2D(distort_seg, rotate,
                           rpoint, status);
```

GM_$DISPLAY_SEGMENT_PART

Displays part of the specified segment (and all called segments) in the current viewport.

FORMAT

```
GM_$DISPLAY_SEGMENT_PART (segment_id, bounds, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

bounds
The part of this segment to be displayed, in terms of segment coordinates. This is a four-element array of real values (xmin, ymin, xmax, ymax), in GM_$BOUNDSREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This command sets the view transformation to a value which causes the specified part of the segment to be displayed as follows: one of the two dimensions fills the viewport, and the other dimension does not overflow the viewport.

It is necessary that ymax be greater than ymin and that xmax be greater than xmin.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_SET_COLOR_MAP

Changes values in the display color map.

FORMAT

```
GM_$DISPLAY_SET_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry to be read. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-plane configuration
1 - 256	For color displays in 8-plane configuration

values
Color value entries, in GM_$COLOR_VECTOR_T format. This is an array of real values. The array must be at least (3 * n_entries) in length.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The GM package initializes the color map to 0 = black, 1 = white.

Use GM_$DISPLAY_INQ_COLOR_MAP to retrieve the value of the display color map.

GM_$DRAW_RASTER_OP

Inserts a command into the current segment: change the logical raster operations to be performed when drawing.

FORMAT

```
GM_$DRAW_RASTER_OP (raster_op, status)
```

INPUT PARAMETERS

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15.

The default raster op value is 3. This sets all destination bit values to source bit values.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_DRAW_RASTER_OP to retrieve the parameters of a raster op command inserted by GM_$DRAW_RASTER_OP.

GM_$DRAW_STYLE

Inserts a command into the current segment: set the line style (solid, dotted).

FORMAT

```
GM_$DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)
```

INPUT PARAMETERS

style

The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SOLID Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default drawing style is GM_$SOLID.

GM_$DOTTED Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.

GM_$PATTERNED
Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.

GM_$SAME_DRAW_STYLE
Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor

The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, repeat_factor is ignored and assumed to be 1.

pattern

The bit pattern, in GM_$DRAW_PATTERN_T format. This is an 8-byte array constituting of a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length

The length of the bit pattern, in bits. This is a 2-byte integer. Allowed values are 1 through 64. Currently, pattern_length is ignored and assumed to be 64.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

The following defines a line pattern with dashes and spaces, twelve and four pixels long, respectively:

```
pattern         : STATIC gm_$draw_pattern_t :=
    [ CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    ];
```

When a styled line is drawn, pixels along the path which are not in the pattern are not affected. In other words, the implicit draw background value is transparent.

Use GM_$INQ_DRAW_STYLE to retrieve the current line style.

GM_$DRAW_VALUE

Inserts a command into the current segment: set the value used when drawing lines.

FORMAT

```
GM_$DRAW_VALUE (value, status)
```

INPUT PARAMETERS

value
The value used in drawing lines. This is a 4-byte integer.

The default draw value is 1.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_DRAW_VALUE to retrieve the current draw value.

GM_$FILE_CLOSE

Closes the current file, saving revisions or not.

FORMAT

```
GM_$FILE_CLOSE (save, status)
```

INPUT PARAMETERS

save

A Boolean (logical) value that indicates whether to save revisions. Set to true to save revisions to the currently open segment; set to false not to save revisions.

Currently, save is always assumed to be true.

If a segment is open in this file, the segment is closed and then the file is closed. If no segment was open, save is ignored.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$FILE_COMPACT

Creates a new compacted GM file.

FORMAT

```
GM_$FILE_COMPACT(name, name_length, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$FILE_COMPACT changes the name of the the input file from * to *.bak (deleting any existing *.bak) and then creates a new compacted GM file named *.

With GM_$FILE_COMPACT, you can develop a file compacting utility like the following:

```
PROGRAM compact;

%NOLIST;
%INCLUDE '/sys/ins/base.ins.pas';
%INCLUDE '/sys/ins/gmr.ins.pas';
%INCLUDE '/sys/ins/pfm.ins.pas';
%LIST;

VAR

    name        : name_$pname_t;
    length      : INTEGER;
    size        : gm_$point16_t := [ 0, 0 ];
    status      : status_$t;

BEGIN

    WRITE( 'File name:  ' );
    READLN( name );

    length := LASTOF( name );
    WHILE ( name[ length ] = ' ' ) AND ( length > 0 )
    DO length := length - 1;

    GM_$INIT
        ( gm_$no_bitmap
        , 0
        , size
        , 1
        , status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    GM_$FILE_COMPACT
        ( name
        , length
        , status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    GM_$TERMINATE
        ( status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    END.
```

GM_$FILE_CREATE

Creates a new graphics metafile and makes it the current file.

FORMAT

```
GM_$FILE_CREATE (name, name_length, access, concurrency, file_id, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

access
The access mode, in GM_$ACC_CREATE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$WRITE If the file already exists, an error message is returned.

GM_$OVERWRITE
If the file already exists, the previous version is deleted.

GM_$UPDATE If the file already exists, the previous version is opened.

concurrency
The concurrency mode, defining the number of concurrent users the file may have, in GM_$CONC_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$1W N readers or 1 writer is permitted.

GM_$COWRITERS
More than 1 writer is permitted, but all users must be on the same node.

Only one segment in the file may be open at a time, and only one writer may be writing to a segment at a time.

OUTPUT PARAMETERS

file_id
The identification number assigned to the file. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The GM_$UPDATE access mode of GM_$FILE_CREATE and the GM_$CWR access mode of GM_$FILE OPEN produce identical results.

GM_$FILE_INQ_BOUNDS

Returns the bounds of the primary segment of a file.

FORMAT

```
GM_$FILE_INQ_BOUNDS (bounds,status)
```

OUTPUT PARAMETERS

bounds

Bounds of the primary segment of the file in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM_$ Data Types section for more information.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use this routine to obtain the bottom left-hand and top right-hand coordinates of the current file.

Use GM_$SEGMENT_INQ_BOUNDS to obtain the boundary of the current segment.

Use GM_$COMMAND_INQ_BOUNDS to obtain the boundary of the current command.

GM_$FILE_INQ_PRIMARY_SEGMENT

Returns the segment number assumed to be the start of the current file.

FORMAT

```
GM_$FILE_INQ_PRIMARY_SEGMENT (segment_id, status)
```

OUTPUT PARAMETERS

segment_id

The number of the primary segment of the current file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The primary segment is assumed to be the start of the picture.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$FILE_OPEN

Reopens an existing file and makes it the current file.

FORMAT

```
GM_$FILE_OPEN (name, name_length, access, concurrency, file_id, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

access
The read/write accessibility, GM_$ACC_OPEN_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$WR — Read or write. In this access mode, it is an error to attempt to open a nonexistent file.

GM_$R — Read only. In this access mode, it is an error to attempt to open a nonexistent file.

GM_$CWR — Read or write; if file does not exist, create it.

The GM_$UPDATE access mode of GM_$FILE_CREATE and the GM_$CWR access mode of GM_$FILE OPEN produce identical results.

concurrency
The concurrency mode, defining the number of concurrent users the file may have, in GM_$CONC_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$1W — N readers or 1 writer is permitted.

GM_$COWRITERS — More than 1 writer is permitted, but all users must be on the same node.

In GM_$COWRITERS concurrency mode, only one segment in the file may be open at a time, and only one writer may be writing to a segment at a time.

OUTPUT PARAMETERS

file_id
The identification number assigned to the file. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

In modes other than GM_$CWR, it is an error to attempt to open a nonexistent file.

GM_$FILE_SELECT

Makes the specified file the current file.

FORMAT

```
GM_$FILE_SELECT (file_id, status)
```

INPUT PARAMETERS

file_id
The identification number of the file which is to become the current file. This is a 2-byte integer.

A file identification number is assigned by the GM package when GM_$FILE_CREATE or GM_$FILE_OPEN is called.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

When a file is created or opened, it becomes the current file. After closing the current file, you must select any other open file before you can use it.

GM_$FILE_SET_PRIMARY_SEGMENT

Changes the segment number assumed to be the start of the current file.

FORMAT

```
GM_$FILE_SET_PRIMARY_SEGMENT (segment_id, status)
```

INPUT PARAMETERS

segment_id
The number of the primary segment of the current file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The primary segment is assumed to be the start of the picture.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$FILL_BACKGROUND_VALUE

Inserts a command into the current segment: set the value used for pixels not in the fill pattern when filling an area.

FORMAT

```
GM_$FILL_BACKGROUND_VALUE (value, status)
```

INPUT PARAMETERS

value
The value used in filling areas. This a 4-byte integer.

The default value is -2. This sets the fill background value equal to the viewport background value.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_FILL_BACKGROUND_VALUE to retrieve the parameters of a fill background value command inserted by GM_$FILL_BACKGROUND_VALUE.

GM_$FILL_PATTERN

Inserts a command into the current segment: set the pattern used for the interior of filled areas.

FORMAT

```
GM_$FILL_PATTERN (scale, size, pattern, status)
```

INPUT PARAMETERS

scale

The number of times each bit in this pattern is to be replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern) or 0 (when clearing a pattern).

A value scale = 0 indicates that filled areas are to be filled with a solid color and that the pattern is to be ignored. In this case, the fill value is assigned to every pixel in the interior of the specified area.

The default value is scale = 0 (solid fill).

size

The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern

The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default fill pattern is all ones.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_FILL_PATTERN to retrieve the parameters of a fill pattern command inserted by GM_$FILL_PATTERN.

GM_$FILL_VALUE

Inserts a command into the current segment: set the value used when filling an area.

FORMAT

```
GM_$FILL_VALUE (value, status)
```

INPUT PARAMETERS

value
The value used in filling areas. This is a 4-byte integer.

The default fill value is 1.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_FILL_VALUE to retrieve the parameters of a fill value command inserted by GM_$FILL_VALUE.

GM_$FONT_FAMILY

Inserts a command into the current segment: set the font family used when writing text.

FORMAT

```
GM_$FONT_FAMILY (font_family_id, status)
```

INPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

The default font family ID is 1.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

As is characteristic of other attribute commands, this command specifies the font family to be used for subsequent text commands of the current segment and all segments instanced from the current segment.

Use GM_$INQ_FONT_FAMILY to get the value stored for the current GM_$FONT_FAMILY command.

GM_$FONT_FAMILY_EXCLUDE

Undoes the inclusion of a font family.

FORMAT

```
GM_$FONT_FAMILY_EXCLUDE (font_family_id, status)
```

INPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Attempting to exclude a font family which is referenced by a font family command (as generated by GM_$FONT_FAMILY) is an error.

GM_$FONT_FAMILY_INCLUDE

Specifies a font family to use in this metafile.

FORMAT

```
GM_$FONT_FAMILY_INCLUDE (pathname, pathname_length, font_type,
                         font_family_id, status)
```

INPUT PARAMETERS

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the pathname. This is a 2-byte integer.

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL A font described pixel by pixel, including all DOMAIN standard fonts.

GM_$STROKE A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$FONT_FAMILY_INQ_ID to retrieve the font family identification of a previously included font family.

Currently, you must include at least one font family before text commands will be displayed.

GM_$FONT_FAMILY_INQ_ID

Returns the identification number of a previously included font family.

FORMAT

```
GM_$FONT_FAMILY_INQ_ID (pathname, pathname_length, font_type,
                        font_family_id, status)
```

INPUT PARAMETERS

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL A font described pixel by pixel, including all DOMAIN standard fonts.

GM_$STROKE A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$FONT_FAMILY_INCLUDE to change a font family to use in this metafile.

GM_$FONT_FAMILY_RENAME

Changes the font family file corresponding to this font family identification.

FORMAT

```
GM_$FONT_FAMILY_RENAME (font_family_id, pathname, pathname_length,
                        font_type, status)
```

INPUT PARAMETERS

font_family_id
The identification number previously assigned to a font family. This is a 2-byte integer.

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the new pathname. This is a 2-byte integer.

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL A font described pixel by pixel, including all DOMAIN standard fonts.

GM_$STROKE A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$INIT

Initializes the graphics metafile package and opens the display.

FORMAT

```
GM_$INIT (display_mode, unit, size, n_planes, status)
```

INPUT PARAMETERS

display_mode

One of four modes of operation. Graphics metafile routines can operate by borrowing the entire display, by using a Display Manager window, by creating a main memory bitmap but no display bitmap, and by building a file without a main memory or display memory bitmap. The value is in GM_$DISPLAY_MODE_T format. Specify only one of the following predefined values:

GM_$BORROW
: Uses the entire screen.

GM_$DIRECT Displays within a Display Manager window.

GM_$MAIN_BITMAP
: Displays within a bitmap allocated in main memory.

GM_$NO_BITMAP
: Allows editing of files without display.

GM_$WITHIN_GPR
: Displays the output of the metafile within a bitmap that you initialize using routines of the DOMAIN graphics primitives.

To use the mode GM_$WITHIN_GPR, you must initialize GPR before calling GM_$INIT. In this mode, you have full control of the screen, but you must handle viewports and input yourself using GPR or other routines. In this mode, these parameters are ignored: unit, size, and n_planes.

GM_$WITHIN_GPR is useful when you already have a user interface and want to use it rather than GM for viewing. GM_$WITHIN_GPR allows you to build sequences of commands using the GM routines which change the contents of a metafile. You can then display the file using GM_$DISPLAY_SEGMENT_GPR_2D. This is the only GM display routine you may use in this mode.

unit

This parameter has three possible meanings as follows:

The display unit, if the display mode is GM_$BORROW. This is a 2-byte integer. Currently, the only valid display unit number for borrow-display mode is 1.

The stream identifier for the pad, if the display mode is GM_$DIRECT. Use STREAM_$ID_T format. This is a 2-byte integer.

Any value, such as zero, in GM_$MAIN_BITMAP, GM_$NO_BITMAP, or GM_$WITHIN_GPR modes.

size

The size of the bitmap, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width in pixels; the second element is the bitmap height in pixels. Each value may be any number between 1 and 4096 (limits are reduced to the display or window size if necessary). See the GM_$ Data Types section for more information.

n_planes

The number of bitmap planes. This is a 2-byte integer. The following are valid values.

```
For display memory bitmaps:

1      For monochromatic displays
1 - 4  For color displays in two-board configuration
1 - 8  For color displays in three-board configuration


For main memory bitmaps: 1 - 8 for all displays
```

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The GM package currently sets color 1 to white on color nodes for portability of applications developed on monochrome nodes. The viewport borders are drawn with color 1. For nonwhite cursors and viewport boundaries on color nodes, use GM_$DISPLAY_SET_COLOR_MAP to respecify color 1.

You can use the "unit" parameter to display metafiles in a window other than the window from which you executed your GM program:

```
VAR
   wndw : pad_$window_desc_t;
   instid,stid : stream_$id_t;
   bitmap_size : gm_$point16_t := [1024,1024];

BEGIN {program}

wndw.top := 0;
wndw.left := 0;
wndw.width := 300;
wndw.height := 300;

pad_$create_window ('',0, pad_$transcript, 1,
                    wndw, stid, st);
pad_$create ('',0,pad_$input,stid, pad_$bottom,
             [pad_$init_raw],5, instid, st);

            { The "unit" parameter is the stream id of the pad
              in which you want to display metafiles. }

gm_$init (gm_$direct,stid,bitmap_size,8,st);
```

The graphics metafile package has its own "clean-up" handler that terminates GM whenever faults are encountered. It is not necessary for an application to install its own fault handler for this purpose. In fact, an application-installed fault handler will not work because GM will no longer be initialized by the time the fault handler is called.

GM_$INPUT_DISABLE

Disables an input event type.

FORMAT

```
GM_$INPUT_DISABLE (event_type, status)
```

INPUT PARAMETERS

event_type

The input event type to be disabled, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE
: Returned when you type a keyboard character.

GM_$BUTTONS
: Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
: Returned when you move the mouse or bitpad puck, or the touchpad.

GM_$ENTERED_WINDOW
: Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.

GM_$LEFT_WINDOW
: Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.

GM_$LOCATOR_STOP
: Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INPUT_ENABLE to enable an input event type.

GM_$INPUT_ENABLE

Enables an input event type.

FORMAT

```
GM_$INPUT_ENABLE (event_type, key_set, status)
```

INPUT PARAMETERS

event_type

The event type to be disabled, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE
: Returned when you type a keyboard character.

GM_$BUTTONS
: Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
: Returned when you move the mouse or bitpad puck, or the touchpad.

GM_$ENTERED_WINDOW
: Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.

GM_$LEFT_WINDOW
: Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.

GM_$LOCATOR_STOP
: Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

key_set

The set of specifically enabled characters when the event type is GM_$KEYSTROKE or GM_$BUTTONS, in GM_$KEYSET_T format. This is an array of up to 256 characters.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INPUT_DISABLE to disable an input event type.

The routines GM_$INPUT_ENABLE and GM_$INPUT_EVENT_WAIT may acquire the display (in direct mode only). GM_$INPUT_ENABLE will acquire the display when locator events are enabled; GM_$INPUT_EVENT_WAIT acquires the display before waiting for an event (and releases it after waiting).

GM_$INPUT_ENABLE expects a Pascal set of characters as one input argument. The following subroutine provides a way to build a set of characters for a FORTRAN program using this call.

```
BUILD_SET -- Builds a Pascal set of characters for FORTRAN users.


INPUT ARGUMENTS

    list           -- An integer*2 array, up to 256 entries long.
                      This array contains the ordinal values of the
                      characters to be included in the set.  For
                      example, if you wish to include the capital
                      letters A through Z, make the array
                      26 entries long, including the values 65
                      through 90.

    no_of_entries  -- The number of entries used in list.
                      An integer*2 scalar.

OUTPUT ARGUMENTS

    returned_set   -- The equivalent of the Pascal set of
                      characters. This can be of any type,
                      as long as it is 32 bytes long.
                      Use integer*4 returned_set(8).


This program does not check for errors.  Therefore, values
can be outside the range 0 to 255, although this can give
unpredictable results.  The program does not check to see
if the value has already appeared in the list.

The subroutine builds the set anew each time; it does not allow
you to add new elements to an existing set.
```

The following program builds a set of characters for FORTRAN users.

```
      PROGRAM build_set

      subroutine build_set(list,no_of_entries,returned_set)

      integer*2 list(1),no_of_entries,returned_set(0:15)
      integer*2 i,mask(0:15),word,bit
      data mask/1,2,4,8,16#10,16#20,16#40,16#80,16#100,16#200,
     1   16#400,16#800,16#1000,16#2000,16#4000,16#8000/


c     A Pascal set of characters is a 256-bit "array." The bit
c     corresponding to the ordinal position of the character is
c     1 if the bit is in the set and 0 if the character is absent
c     from the set. In this example, the set is initialized
c     to 0, that is, no characters are present.

      do 100 i=0,15
         returned_set(i) = 0
100      continue
c
c     Go through the list, setting the bits for each character listed.
c     Note that Pascal numbers the bits right to left.
c     Therefore, a set containing only char(0), that is NULL, has
c     only the least-significant bit set in the last word of the set.

      do 110 i=1,no_of_entries
c
c     Set the appropriate bit.

         word = 15 - (list(i)/16)
         bit = mod(list(i),16)
         returned_set(word) = or(returned_set(word),mask(bit))
110   continue
c
      return
      end
```

GM_$INPUT_EVENT_WAIT

Checks for or waits until an occurrence of an enabled input event.

FORMAT

```
GM_$INPUT_EVENT_WAIT (wait, event_type, event_data, bitmap_position,
                      viewport_id, segment_position, status)
```

INPUT PARAMETERS

wait

A Boolean (logical) value that specifies when control returns to the calling program. Set to true to wait for an enabled event to occur; set to false to return control to the calling program immediately, whether or not an event has occurred.

OUTPUT PARAMETERS

event_type

The event type which occurred, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE
: Returned when you type a keyboard character.

GM_$BUTTONS
: Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
: Returned when you move the mouse or bitpad puck, or the touchpad.

GM_$ENTERED_WINDOW
: Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.

GM_$LEFT_WINDOW
: Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.

GM_$LOCATOR_STOP
: Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

event_data

The keystroke or button character associated with the event, or the character that identifies the window associated with an entered window event. This is a character. This parameter is not modified for other events.

bitmap_position

The position in the display bitmap at which graphics input occurred, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

viewport_id
The identification number of the viewport in which the location "bitmap_position" is found. This is a 2-byte integer.

segment_position
The position at which graphics input occurred, converted to segment coordinates of the viewport primary segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

If the location "bitmap_position" is not in any viewport, viewport_id is zero and segment_position is undefined.

If the location "bitmap_position" is in a viewport which is not displaying, segment_position is undefined.

The routines GM_$INPUT_ENABLE and GM_$INPUT_EVENT_WAIT may acquire the display (in direct mode only). GM_$INPUT_ENABLE will acquire the display when locator events are enabled; GM_$INPUT_EVENT_WAIT acquires the display before waiting for an event (and releases it after waiting).

GM_$INQ_ACLASS

Returns the value stored for the current (GM_$ACLASS) command.

FORMAT

```
GM_$INQ_ACLASS (aclass_id, status)
```

OUTPUT PARAMETERS

aclass_id
The identification number of the attribute class to use. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Inquiring about a command that is not an GM_$ACLASS command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_BITMAP_SIZE

Returns the size of the GM bitmap in pixels.

FORMAT

```
GM_$INQ_BITMAP_SIZE (size, planes, status)
```

OUTPUT PARAMETERS

size

The size of the GM bitmap created when the GM package was initialized, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

In direct mode, this routine returns the size of the part of the Display Manager window in which the GM package was initialized, excluding the edges of the window reserved by the Display Manager.

In borrow mode, this is the size of the part of the borrowed display in which the GM package was initialized.

In main-bitmap mode, this is the size of the main memory bitmap which was created when the GM package was initialized.

planes

The number of planes in the GM bitmap created when the GM package was initialized. This is a 2-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$INQ_CIRCLE_[16,32,REAL]

Returns the values stored for the current (GM_$CIRCLE) command.

FORMAT

```
GM_$INQ_CIRCLE_16 (center, radius, fill, status)

GM_$INQ_CIRCLE_32 (center, radius, fill, status)

GM_$INQ_CIRCLE_REAL (center, radius, fill, status)
```

OUTPUT PARAMETERS

center

The point that is the center of the circle. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$INQ_CIRCLE_16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$INQ_CIRCLE_32

GM_$POINTREAL
: A two-element array of real values for GM_$INQ_CIRCLE_REAL

See the GM_$ Data Types section for more information.

radius

The radius of the circle, in the appropriate format:

A 2-byte integer for GM_$INQ_CIRCLE_16

A 4-byte integer for GM_$INQ_CIRCLE_32

A real value for GM_$INQ_CIRCLE_REAL

fill

A Boolean (logical) value which specifies whether the circle is filled.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The parameters are returned as they were supplied to the GM_$CIRCLE command that inserted this command into the metafile.

Use $GM_$INQ_COMMAND_TYPE to get the command type and the data type of this command.

Use $GM_$COMMAND_DELETE and GM_$CIRCLE_[16,32,REAL] to change the parameters of this command.

Inquiring about a command that is not a GM_$CIRCLE command results in an error.

Currently, you must use GM_$INQ_CIRCLE_16 if the stored data type is GM_$16; you must use GM_$INQ_CIRCLE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_COMMAND_TYPE

Returns the command type and the data type of the current command in the current segment.

FORMAT

```
GM_$INQ_COMMAND_TYPE  (command_type, data_type, status)
```

OUTPUT PARAMETERS

command_type

The type of command, in GM_$COMMAND_TYPE_T format. This is a 2-byte integer. One of the following predefined values is returned:

```
GM_$TACLASS
GM_$TCIRCLE_2D
GM_$TCURVE_2D
GM_$TDRAW_RASTER_OP
GM_$TDRAWSTYLE
GM_$TDRAWVALUE
GM_$TFILLBVALUE
GM_$TFILLPATTERN
GM_$TFILLVALUE
GM_$TFONTFAMILY
GM_$TINSTANCE_SCALE_2D
GM_$TINSTANCE_TRANS_2D
GM_$TINSTANCE_TRANSFORM_2D
GM_$TPLANEMASK
GM_$TPOLYLINE_2D
GM_$TPRIMITIVE_2D
GM_$TRECTANGLE
GM_$TTAG
GM_$TTEXT_2D
GM_$TTEXTBVALUE
GM_$TTEXTSIZE
GM_$TTEXTVALUE
```

data_type

The data storage type, in GM_$DATA_TYPE_T format. The possible values for this parameter are the following:

GM_$16	Data is stored as GM_$POINT16_T
GM_$32	Data is stored as GM_$POINT32_T

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_POLYLINE, GM_$INQ_TEXT, and other similar commands with data storage types to get the parameters of the command.

Use GM_$SEGMENT_CREATE, GM_$SEGMENT_OPEN and GM_$SEGMENT_CLOSE to change the current segment. Use GM_$PICK_COMMAND to change the current command.

If the current command is the blank space at the start of the segment, as it is after GM_$PICK_COMMAND(GM_$START,STATUS), this routine returns a GM_$NO_CURRRENT_COMMAND error.

GM_$INQ_CONFIG

Returns the current configuration of the display device.

FORMAT

```
GM_$INQ_CONFIG (configuration, status)
```

OUTPUT PARAMETERS

configuration

Current display configuration, in GM_$DISPLAY_CONFIG_T format. This is a 2-byte integer. One of the following predefined values is returned:

Returned Value	Display Type
GM_$BW_800x1024	monochromatic portrait
GM_$BW_1024x800	monochromatic landscape
GM_$COLOR_1024x1024x4	color 1024 x 1024 (DN6xx) 2-board config
GM_$COLOR_1024x1024x8	color 1024 x 1024 (DN6xx) 3-board config
GM_$COLOR_1024x800x4	color 1024 x 800 (DN5xx) 2-board config
GM_$COLOR_1024x1024x8	color 1024 x 800 (DN5xx) 3-board config
GM_$COLOR1_1024X800X8	color 1024 x 800 (DN570) 2-board config
GM_$COLOR_1280X1024X8	color 1280 x 1024 (DN580) 2-board config
GM_$COLOR2_1024X800X4	color 1024 x 800 (DN3000) 1-board config

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM Data Types section for more information.

USAGE

GM_$INQ_CONFIG is the only GM routine call that is usable when the graphics metafile package is not initialized.

GM_$INQ_CURVE_2D[16,32,REAL]

Returns the values stored for the current (GM_$CURVE) command.

FORMAT

```
GM_$INQ_CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                      parameter_array, status)
```

OUTPUT PARAMETERS

curve_type
The type of curve, in GM_$CURVE_T format. This is a 2-byte integer. One of the following values is returned:

GM_$ARC_3P Specifies an arc to be drawn through three points (n_points) in the point array (point_array). The value for n_points must equal 3.

GM_$SPLINE_CUBIC_P
Specifies a smooth curve (parametric cubic spline) to be drawn through the specified number of point (n_points) in the point array (point_array).

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinate points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INQ_CURVE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INQ_CURVE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INQ_CURVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters. This is an array of reals.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Currently, n_parameters and parameter_array are not used.

Inquiring about a command that is not a GM_$CURVE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$CURVE_2D[16,32,REAL] to change the parameters of this command.

Currently, you must use GM_$INQ_CURVE_16 if the stored data type is GM_$16; you must use GM_$INQ_CURVE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_DRAW_RASTER_OP

Returns the values stored for the current (GM_$DRAW_RASTER_OP) command.

FORMAT

```
GM_$INQ_DRAW_RASTER_OP (raster_op, status)
```

OUTPUT PARAMETERS

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15. The default value is 3.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$DRAW_RASTER_OP to change the raster operation codes.

Inquiring about a command that is not a GM_$DRAW_RASTER_OP command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_DRAW_STYLE

Returns the values stored for the current (GM_$DRAW_STYLE) command.

FORMAT

```
GM_$INQ_DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)
```

OUTPUT PARAMETERS

style

The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. One of the following values is returned:

GM_$SOLID — Default. Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is solid.

GM_$DOTTED — Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.

GM_$PATTERNED — Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.

GM_$SAME_DRAW_STYLE — Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor

The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, repeat_factor is ignored and assumed to be 1.

pattern

The bit pattern, in GM_$DRAW_PATTERN_T format. This is an 8-byte array constituting a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length

The length of the pattern. This is a 2-byte integer. Currently, pattern_length is ignored and assumed to be 64.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$SET_DRAW_STYLE to change the line style.

Inquiring about a command that is not a GM_$DRAW_STYLE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_DRAW_VALUE

Returns the value stored for the current (GM_$DRAW_VALUE) command.

FORMAT

```
GM_$INQ_DRAW_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value used in drawing lines. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$DRAW_VALUE to change the line drawing value.

Inquiring about a command that is not a GM_$DRAW_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FILL_BACKGROUND_VALUE

Returns the value stored for the current (GM_$FILL_BACKGROUND_VALUE) command.

FORMAT

```
GM_$INQ_FILL_BACKGROUND_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The fill background value used in this command. This is a 4-byte integer. The default value is -2, the same as the viewport background.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Inquiring about a command that is not a GM_$FILL_BACKGROUND_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$FILL_BACKGROUND_VALUE to change the fill background value.

GM_$INQ_FILL_PATTERN

Returns the value stored for the current (GM_$FILL_PATTERN) command.

FORMAT

```
GM_$INQ_FILL_PATTERN (scale, size, pattern, status)
```

OUTPUT PARAMETERS

scale
The number of times each bit in this pattern is replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern) or 0 (when clearing a pattern).

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a 2-byte integer array of two elements. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern
The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$FILL_PATTERN to change the fill pattern.

Inquiring about a command that is not a GM_$FILL_PATTERN command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FILL_VALUE

Returns the value stored for the current (GM_$FILL_VALUE) command.

FORMAT

```
GM_$INQ_FILL_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value used in filling areas. This is a 4-byte integer. The default fill value is 1.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$FILL_VALUE to change the fill value.

Inquiring about a command that is not a GM_$FILL_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FONT_FAMILY

Returns the value stored for the current (GM_$FONT_FAMILY) command.

FORMAT

```
GM_$INQ_FONT_FAMILY (font_family_id, status)
```

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default value is 1.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$FONT_FAMILY to change the font family,

Inquiring about a command that is not a GM_$FONT_FAMILY command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_INSTANCE_SCALE_2D[16,32,REAL]

Returns the value stored for the current (GM_$INSTANCE_SCALE_2D) command.

FORMAT

```
GM_$INQ_INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to be instanced, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

scale
A real number indicating the scaling factor.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_SCALE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_SCALE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_SCALE_2DREAL

See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Scaling is performed before translation.

Use GM_$INSTANCE_SCALE_2D[16,32,REAL] to change the segment instanced and its scale and translation parameters.

Inquiring about a command that is not a GM_$INSTANCE_SCALE_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_SCALE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_SCALE_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_INSTANCE_TRANSFORM_2D[16,32.REAL]

Returns the value stored for the current (GM_$INSTANCE_TRANSLATE) command.

FORMAT

```
GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL] (segment_id, rotate, translate,
                                           status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to transform, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of 4 real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation, in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_TRANSFORM_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_TRANSFORM_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_TRANSFORM_2DREAL

See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSFORM_2DREA | L
```

Use GM_$INSTANCE_TRANSFORM_2D[16,32,REAL] to change the segment instanced and its translation and rotation parameters.

Inquiring about a command that is not a GM_$INSTANCE_TRANSFORM_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_TRANSFORM_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_TRANSFORM_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_INSTANCE_TRANSLATE_2D[16,32,REAL]

Returns the value stored for the current (GM_$INSTANCE_TRANSLATE_2D) command.

FORMAT

```
GM_$INQ_INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2D32 (segment_id,  translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to be instanced, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_TRANSLATE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_TRANSLATE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_TRANSLATE_2DREAL

See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSLATE_2DREA | L
```

Use GM_$INSTANCE_TRANSLATE_2D[16,32,REAL] to change the segment instanced and its translation parameters.

Inquiring about a command that is not a GM_$INSTANCE_TRANSLATE_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_TRANSLATE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_TRANSLATE_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_PLANE_MASK

Returns the value stored for the current (GM_$PLANE_MASK) command.

FORMAT

```
GM_$INQ_PLANE_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer. (See the description under GM_$PLANE_MASK.)

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PLANE_MASK to set the plane mask.

Inquiring about a command that is not a GM_$PLANE_MASK command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_POLYLINE_2D[16,32,REAL]

Returns the values stored for the current (GM_$POLYLINE_2D) command.

FORMAT

```
GM_$INQ_POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2DREAL (n_points, point_array, close, fill, status)
```

OUTPUT PARAMETERS

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INQ_POLYLINE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INQ_POLYLINE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INQ_POLYLINE_2DREAL

See the GM_$ Data Types section for more information.

close
A Boolean (logical) value which specifies whether the first and last points are connected. Set the parameter to true to close the polygon. You must use close when you want to fill a polygon.

fill
A Boolean (logical) value which specifies whether to fill the polygon or not. Filled polygons must be closed. Set the parameter to true to fill the polygon; set it to false for an unfilled polygon.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The parameters are returned as they were supplied to the command GM_$POLYLINE_[16,32,REAL] which inserted this command into the metafile.

Currently, you must use GM_$INQ_POLYLINE_16 if the stored data type is gm_$16; you must use GM_$INQ_POLYLINE_32 or _REAL if the stored data type is gm_$32.

Inquiring about a command that is not a GM_$POLYLINE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_PRIMITIVE_2D[16,32,REAL]

Returns the values stored for the current (GM_$PRIMITIVE) command.

FORMAT

```
GM_$INQ_PRIMITIVE_2D16  (primitive_type, n_points, point_array, n_parameters,
                         parameter_array, status)

GM_$INQ_PRIMITIVE_2D32 (primitive_type, n_points, point_array, n_parameters,
                        parameter_array, status)

GM_$INQ_PRIMITIVE_2DREAL (primitive_type, n_points, point_array, n_parameters,
                          parameter_array, status)
```

OUTPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$PRIMITIVE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$PRIMITIVE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$PRIMITIVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters, in GM_$ARRAYREAL_T format. This is an array of real values.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PRIMITIVE_2D[16,32,REAL] to change the current value of the primitive command.

Currently, you must use GM_$INQ_PRIMITIVE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_PRIMITIVE_2D32 or _REAL if the stored data type is GM_$32.

GM_$INQ_RECTANGLE_[16,32,REAL]

Returns the values stored for the current (GM_$RECTANGLE) command.

FORMAT

```
GM_$INQ_RECTANGLE_16 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_32 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_REAL (point1, point2, fill, status)
```

OUTPUT PARAMETERS

point1, point2

The coordinates of two diagonally opposite corners, each a pair (x,y) of integers in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$RECTANGLE_16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$RECTANGLE_32

GM_$POINTREAL
: A two-element array of real values for GM_$RECTANGLE_REAL

See the GM_$ Data Types section for more information.

The GM package sorts rectangle coordinates before storing them. The returned parameter point1 will contain the smaller x value and the smaller y value, regardless of the order in which you supplied the data.

fill

A Boolean (logical) value which specifies whether to fill the rectangle or not. Set the parameter to true to fill the rectangle; set it to false for an unfilled rectangle.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$RECTANGLE_[16,32,REAL] to change the values for this command.

Use GM_$INQ_RECTANGLE_[16,32,REAL] to retrieve the parameters of a rectangle command inserted by GM_$RECTANGLE_[16,32,REAL].

Currently, you must use GM_$INQ_RECTANGLE_16 if the stored data type is GM_$16; you must use GM_$INQ_RECTANGLE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_TAG

Returns the value stored for the current (GM_$TAG) command.

FORMAT

```
GM_$INQ_TAG (string, string_length, status)
```

OUTPUT PARAMETERS

string
The text string stored, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length
The length of the string. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$TAG to change the text string stored in this command.

GM_$INQ_TEXT_2D[16,32,REAL]

Returns the value stored for the current (GM_$TEXT_2D[16,32,REAL]) command.

FORMAT

```
GM_$INQ_TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)
```

OUTPUT PARAMETERS

point

The coordinates of the point at which to locate text. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$TEXT_2D16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$TEXT_2D32

GM_$POINTREAL
: A two-element array of real values for GM_$TEXT_2DREAL

See the GM_$ Data Types section for more information.

rotate

The angle at which this text string is to be written, in degrees. This is a real value. A value of 0.0 degrees indicates left to right text. Other values indicate clockwise rotation. For example, -90.0 degrees specifies bottom to top.

string

The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length

The length of the string. This is a 2-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$TEXT_2D[16,32,REAL] to change the text string.

Inquiring about a command that is not a GM_$TEXT_2D[16,32,REAL] command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$INQ_TEXT_2D[16,32,REAL] to retrieve the parameters of a text command inserted by GM_$TEXT_2D[16,32,REAL].

Currently, you must use GM_$INQ_TEXT_2D16 if the stored data type is GM_$16; you must use GM_$INQ_TEXT_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_TEXT_BACKGROUND_VALUE

Returns the value stored for the current (GM_$TEXT_BACKGROUND_VALUE) command.

FORMAT

```
GM_$INQ_TEXT_BACKGROUND_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The background value to use when writing text. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$TEXT_BACKGROUND_VALUE to change the text background value.

Inquiring about a command that is not a GM_$TEXT_BACKGROUND_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_TEXT_SIZE

Returns the value stored for the current (GM_$TEXT_SIZE) command.

FORMAT

```
GM_$INQ_TEXT_SIZE (size, status)
```

OUTPUT PARAMETERS

size

The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$TEXT_SIZE to change the text size.

Inquiring about a command that is not a GM_$TEXT_SIZE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_TEXT_VALUE

Returns the value stored for the current (GM_$TEXT_VALUE) command.

FORMAT

```
GM_$INQ_TEXT_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value to use when writing text. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$TEXT_VALUE to change the text value.

Inquiring about a command that is not a GM_$TEXT_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INSTANCE_SCALE_2D[16,32,REAL]

Inserts a command into the current segment: instance the specified segment with the specified scale and translation parameters.

FORMAT

```
GM_$INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to instance, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

scale
A real number indicating the scaling factor.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_SCALE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_SCALE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_SCALE_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Scaling is performed before translation.

Use GM_$SEGMENT_GET_ID to find the number of a previously defined segment for which you know only the name.

Use GM_$INQ_INSTANCE_SCALE_2D[16,32,REAL] to retrieve the parameters of an instance scale command inserted by GM_$INSTANCE_SCALE_[16,32,REAL].

Before supplying coordinate data to GM_$INSTANCE_SCALE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$INSTANCE_TRANSFORM_2D[16,32.REAL]

Inserts a command to instance the specified segment with the specified rotation and translation applied.

FORMAT

```
GM_$INSTANCE_TRANSFORM_2D16 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2D32 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2DREAL (segment_id, rotate, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to transform, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_TRANSFORM_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_TRANSFORM_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_TRANSFORM_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                    |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSFORM_2DREA | L
```

Rotation is performed before translation.

Use GM_$SEGMENT_GET_ID to find the number of a previously defined segment for which you know only the name.

Use GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL] to retrieve the parameters of an instance transform command inserted by GM_$INSTANCE_TRANSFORM_2D[16,32,REAL].

You must call GM_$DATA_COERCE_SET_REAL before supplying coordinate data to GM_$INSTANCE_TRANSFORM_2DREAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

Rotation: Use the following to include an instance of segment little_seg at (400,300), at triple size and rotated 50 degrees:

```
rotate.xx := 3.0 * cos(50.0 * 3.14159/180.0);
rotate.xy := 3.0 * sin(50.0 * 3.14159/180.0);
rotate.yx := -rotate.xy;
rotate.yy := rotate.xx;
point.x := 400;
point.y := 300;
GM_$INSTANCE_TRANSFORM_2D16(little_seg, rotate, point, status);
```

Distortion: Use the following to include an instance of segment distort_seg at (12.5, 14.5), with a scale of 1 in the x direction and a scale of 3 in the y direction, unrotated:

```
rotate.xx := 1.0;
rotate.xy := 0.0;
rotate.yx := 0.0;
rotate.yy := 3.0;
rpoint.x  := 12.5;
rpoint.y  := 14.5;
GM_$INSTANCE_TRANSFORM_2DREAL(distort_seg, rotate, rpoint, status);
```

GM_$INSTANCE_TRANSLATE_2D[16,32,REAL]

Inserts a command into the current segment: instance the identified segment with the specified translation.

FORMAT

```
GM_$INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2D32 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to instance, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INSTANCE_TRANSLATE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INSTANCE_TRANSLATE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$INSTANCE_TRANSLATE_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012  | 34567890
----------------------------------|---------
GM_$INQ_INSTANCE_TRANSLATE_2DREA  | L
```

Use GM_$INQ_INSTANCE_TRANSLATE_2D[16,32,REAL] to retrieve the parameters of an instance translate command inserted by GM_$INSTANCE_TRANSLATE_2D[16,32,REAL].

You must call GM_$DATA_COERCE_SET_REAL before supplying coordinate data to GM_$INSTANCE_TRANSLATE_2DREAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$MODELCMD_INQ_MODE

Returns the values stored for the current (GM_$MODELCMD_SET_MODE) command.

FORMAT

```
GM_$MODELCMD_INQ_MODE (gm_modelcmd_mode, status)
```

OUTPUT PARAMETERS

gm_modelcmd_mode
The editing mode, in GM_$MODELCMD_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$MODELCMD_INSERT
: Modeling commands insert a command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = false.

GM_$MODELCMD_REPLACE
: Modeling command replaces the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = true.

GM_$MODELCMD_RUBBERBAND
: Modeling commands XOR the previous command on the screen, thus erasing it, then XOR the given command onto the screen. Only bit plane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$MODELCMD_SET_MODE to set the model command mode.

GM_$MODELCMD_SET_MODE

Sets the modeling command mode.

FORMAT

```
GM_$MODELCMD_SET_MODE (gm_modelcmd_mode, status)
```

INPUT PARAMETERS

gm_modelcmd_mode
The editing mode to use, in GM_$MODELCMD_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$MODELCMD_INSERT
Modeling commands insert a command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = false.

GM_$MODELCMD_REPLACE
Modeling commands replace the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = true.

GM_$MODELCMD_RUBBERBAND
Modeling commands XOR the previous modeling command on the screen, thus erasing it, then XOR the given modeling command onto the screen. Only bitplane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Rubberband mode provides an interactive capability. The mode uses a pointing device and allows an application program to get information from a user. The application program then uses the information to insert or replace the command with the changes the user has specified.

You may still set or inquire the replace flag by calling GM_$REPLACE_$SET_FLAG GM_$REPLACE_$INQ_FLAG, respectively. In new programs, use the GM_$MODELCMD... routines.

Use GM_$MODELCMD_INQ_MODE to get the values stored in this command.

GM_$PICK_COMMAND

Within the current segment, selects a command which contains a selected point on the display.

FORMAT

```
GM_$PICK_COMMAND (search_rule, status)
```

INPUT PARAMETERS

search_rule

The search rule to apply in selecting the command, in GM_$SEARCH_COMMAND_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$CNEXT — Find the next command which falls within the pick aperture, moving forward in the segment.

GM_$STEP — Find the next command in the segment, independent of the coordinates of the pick aperture.

GM_$START — Move to the start of the segment, independent of the coordinates of the pick aperture.

If search_rule = GM_$START, the current command is changed to equal beginning-of-segment (no current command), allowing commands to be added at the beginning of the segment.

GM_$END — Move to the end of the segment, independent of the coordinates of the pick aperture.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The current command is changed to equal the picked command.

Instance commands are treated like any other command in their context. To pick "into" an instanced segment, use GM_$PICK_SEGMENT.

Commands are picked only if the pick aperture intersects the drawn command. An exception is the GM_$CURVE_2D... commands. These commands are picked if the bounding box of the command intersects the pick aperture.

GM_$PICK_HIGHLIGHT_COMMAND

Highlights the current command on the display.

FORMAT

```
GM_$PICK_HIGHLIGHT_COMMAND (highlight, time, status)
```

INPUT PARAMETERS

highlight
The method to be used for highlighting the command, in GM_$HIGHLIGHT_T format. This is a 2-byte integer. Currently, the only possible value is GM_$OUTLINE. This value draws a rectangular outline around the command, leaves it displayed for the specified amount of time, and then erases it.

time
The number of seconds for which the command is to be highlighted. This is a real value.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This operation is performed only if the viewport primary segment of the current viewport is the current segment.

The outline drawn around picked commands and picked segments is temporary. The outline remains on the screen for the requested number of seconds and is then erased.

GM_$PICK_HIGHLIGHT_SEGMENT

Within the current file, highlights the specified segment.

FORMAT

```
GM_$PICK_HIGHLIGHT_SEGMENT (highlight, time, status)
```

INPUT PARAMETERS

highlight
The method to be used for highlighting the segment, in GM_$HIGHLIGHT_T format. This is a 2-byte integer. Currently, the only possible value is GM_$OUTLINE. This value draws a rectangular outline around the segment, leaves it displayed for the specified amount of time, and then erases it.

time
The number of seconds for which the segment is to be highlighted. This is a real value.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This operation is performed only if the viewport primary segment of the current viewport is the first segment in the pick list.

The outline drawn around picked segments and picked commands is temporary. The outline remains on the screen for the requested number of seconds and is then erased.

GM_$PICK_INQ_CENTER

Returns the center of the pick aperture.

FORMAT

```
GM_$PICK_INQ_CENTER (center, status)
```

OUTPUT PARAMETERS

center
The (x,y) coordinates of the center of the pick aperture, in GM_$POINTREAL_T format. This is a two-element array of real values.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

A program must call GM_$PICK_SET_CENTER and GM_$PICK_SET_SIZE after the use of the commands GM_$FILE_DISPLAY or GM_$SEGMENT_DISPLAY.

GM_$PICK_INQ_LIST

Returns the current list of picked segments.

FORMAT

```
GM_$PICK_INQ_LIST (max_length, length, list, status);
```

INPUT PARAMETERS

max_length
The maximum length of list you are prepared to receive. This is a 2-byte integer.

OUTPUT PARAMETERS

length
The number of segment ID's returned. This is a 2-byte integer.

list
An array of segment ID's, each in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$PICK_INQ_MASK

Returns the value of the mask used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_INQ_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The pick mask value. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PICK_SET_MASK to change the current value of the pick mask.

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$PICK_INQ_SIZE

Returns the size of the pick aperture.

FORMAT

```
GM_$PICK_SET_SIZE (size, status)
```

OUTPUT PARAMETERS

size

The x and y tolerances for the pick aperture, in segment coordinates of the current segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PICK_SET_SIZE to change the size of the pick aperture.

GM_$PICK_INQ_THRESHOLD

Returns the value of the threshold used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_INQ_THRESHOLD (threshold, status)
```

OUTPUT PARAMETERS

threshold
The pick threshold value. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PICK_SET_THRESHOLD to change the current value of the pick threshold.

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$PICK_SEGMENT

Selects a segment which contains a specified point on the display.

FORMAT

```
GM_$PICK_SEGMENT (search_rule, segment_id, n_instances, bounds, status)
```

INPUT PARAMETERS

search_rule
The search rule to apply in selecting the command, in GM_$SEARCH_SEGMENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SETUP Make the top segment of the current viewport the start of the list of picked segments. The rest of the list is emptied.

GM_$DOWN Find the first segment instanced by the current segment which, when instanced, falls within the pick aperture.

GM_$NEXT Find the next segment within the segment one higher in the list of picked segments, which falls within the pick aperture.

GM_$UP Move up one level in the list of picked segments.

GM_$TOP Proceed to top segment in the list of picked segments, destroying the rest of the list of picked segments.

GM_$CLEAR Clear the entire list of picked segments, allowing all segments to be edited or deleted.

GM_$BOTTOM Perform GM_$DOWN repeatedly until a segment is reached for which GM_$DOWN finds nothing.

GM_$NEXTBOTTOM
Perform GM_$BOTTOM. If nothing is found, perform GM_$NEXT, or one or more GM_$UPs followed by a GM_$NEXT, until a GM_$NEXT finds a segment. When a GM_$NEXT finds a segment, perform a GM_$BOTTOM from there.

OUTPUT PARAMETERS

segment_id
The identification number of the picked segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

n_instances
The number of instances of this segment in the file. This is a 4-byte integer.

bounds
(Reserved for future extension.) A GM_$BOUNDSREAL_T variable. This is an array of 4 real values.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PICK_SEGMENT to set the current segment.

If a segment is picked, the picked segment becomes the last segment on the list of picked segments. If no segment is picked, the list of picked segments is unchanged. While a segment is in the picked list, it may not be deleted or edited.

Use GM_$INQ_PICK_LIST to examine the current list of picked segments.

Each segment listed in the list of picked segments is instanced in the preceding segment.

GM_$PICK_SET_CENTER

Changes the center of the pick aperture.

FORMAT

```
GM_$PICK_SET_CENTER (center, status)
```

INPUT PARAMETERS

center

The (x,y) coordinates of the center of the pick aperture, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

When picking commands, GM_$PICK_SET_CENTER uses the segment coordinates of the current segment.

When picking segments, in other than no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the viewport primary segment of the segment in which pick segment operations were initialized.

When picking segments in no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the primary segment of the file.

The PICK routines search for any segments/commands which fall into the following region:

```
(center.x - size.x  to  center.x + size.x,
 center.y - size.y  to  center.y + size.y)
```

GM_$PICK_SET_MASK

Changes the value of the mask used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_SET_MASK (mask, status)
```

INPUT PARAMETERS

mask

The pick mask value. This is a 4-byte integer.

The pick mask is initialized to 16#7FFF (all segments with nonzero pickable values are pickable).

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

Use GM_$PICK_INQ_MASK to retrieve the current value of the pick mask.

GM_$PICK_SET_SIZE

Specifies the size of the pick aperture.

FORMAT

```
GM_$PICK_SET_SIZE (size, status)
```

INPUT PARAMETERS

size

The x and y tolerances for the pick aperture, in segment coordinates of the current segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$PICK_INQ_SIZE to retrieve the size of the pick aperture.

When picking commands, GM_$PICK_SET_CENTER uses the segment coordinates of the current segment.

When picking segments, in other than no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the viewport primary segment of the segment in which pick segment operations were initialized.

When picking segments, in no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the primary segment of the file.

The dimensions for the pick aperture are the following: (2 * size.x, 2 * size.y).

The PICK routines search for any segments/commands which fall into the following region:

```
(center.x - size.x  to  center.x + size.x,
 center.y - size.y  to  center.y + size.y).
```

GM_$PICK_SET_THRESHOLD

Returns the value of the threshold used in pick search operations in the current segment.

FORMAT

```
GM_$PICK_SET_THRESHOLD (threshold, status)
```

INPUT PARAMETERS

threshold
The pick threshold value. This is a 4-byte integer.

The pick threshold is initialized to 1 (all segments with nonzero pickable values are pickable).

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

Use GM_$PICK_INQ_THRESHOLD to change the current value of the pick threshold.

GM_$PICK_TRANSFORM_POINT

Transforms the coordinates of a point from the coordinate system of the viewport segment to the coordinate system of the picked segment.

FORMAT

```
GM_$PICK_TRANSFORM_POINT (vsegment_position, psegment_position, status)
```

INPUT PARAMETERS

vsegment_position
A point in viewport coordinates, in GM_$POINTREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

psegment_position
A point in picked segment coordinates in GM_$POINTREAL_T format. See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Strictly speaking, in this context "picked segment" means a selected instance of a segment.

GM_$PLANE_MASK

Inserts a command into the current segment: change the plane mask.

FORMAT

```
GM_$PLANE_MASK (mask, status)
```

INPUT PARAMETERS

mask

The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer.

The default value is [0...7], in GM_$PLANE_MASK_T format, or 255 when expressed as a 2-byte integer. The default is that all planes are in use and can be modified.

FORTRAN programmers should encode the plane mask in a 2-byte integer in the range of 0-255 (1 means plane 0 is on, 2 means plane l is on, 3 means planes 0 and 1 are on; 255 means planes 0 through 7 are on).

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_PLANE_MASK to get the value stored for the current GM_$PLANE_MASK command.

FORTRAN programmers might want to include the parameter definitions given below:

```
      integer*2
     +    bit0,
     +    bit1,
     +    bit2,
     +    bit3,
     +    bit4,
     +    bit5,
     +    bit6,
     +    bit7
      parameter (
     +    bit0  16#0001,
     +    bit1  16#0002,
     +    bit2  16#0004,
     +    bit3  16#0008,
     +    bit4  16#0010,
     +    bit5  16#0020,
     +    bit6  16#0040,
     +    bit7  16#0080)
```

Example:

In FORTRAN, to enable planes 2 and 5, use the following:

```
    CALL GM_$PLANE_MASK( bit2 + bit5, status )
```

In Pascal, to enable planes 2 and 5, use the following:

```
    GM_$PLANE_MASK( [ 2, 5 ], status )
```

GM_$POLYLINE_2D[16,32,REAL]

Inserts a command into the current segment: draw a linked set of line segments.

FORMAT

```
GM_$POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2DREAL (n_points, point_array, close, fill, status)
```

INPUT PARAMETERS

n_points
The number of points in the list of points. This is a 2-byte integer. Polylines are limited to 1000 (GM_$MAX_ARRAY_LENGTH) points.

point_array
A list of coordinate points, each a pair (x,y) of integers in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$POLYLINE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$POLYLINE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$POLYLINE_2DREAL

See the GM_$ Data Types section for more information.

close
A Boolean (logical) value which specifies whether the first and last points are connected. Set the parameter to true to close the polygon. You must use close when you want to fill a polygon.

fill
A Boolean (logical) value which specifies whether to fill the polygon or not. Filled polygons must be closed. Set the parameter to true to fill the polygon; set it to false for an unfilled polygon.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_POLYLINE_2D[16,32,REAL] to retrieve the parameters of a polyline command inserted by GM_$POLYLINE_2D[16,32,REAL].

Currently, you must use GM_$INQ_POLYLINE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_POLYLINE_2D32 or _2DREAL if the stored data type is GM_$32.

Selecting close = false and fill = true results in an error.

Before supplying coordinate data to GM_$POLYLINE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$PRIMITIVE_2D[16,32,REAL]

Inserts a command into the current segment: draw a primitive.

FORMAT

```
GM_$PRIMITIVE_2D16 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2D32 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2DREAL (primitive_type, n_points, point_array, n_parameters,
                      parameter_array, status)
```

INPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

Each distinct value of primitive_type corresponds to a different user-defined primitive display routine. For each primitive_type you use, you must write a user-defined display routine to be used when displaying (GM_$PRIMITIVE_2D) commands of that primitive type. You define a specified display routine to be used for displaying a specified primitive type using the routine GM_$PRIMITIVE_DISPLAY_2D.

n_points
The number of points in the list of points. This is a 2-byte integer. The number of points is limited to 1000 (GM_$MAX_ARRAY_LENGTH).

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$PRIMITIVE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$PRIMITIVE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$PRIMITIVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters, in GM_$ARRAYREAL_T format. This is an array of real values.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Before supplying coordinate data to GM_$PRIMITIVE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$PRIMITIVE_DISPLAY_2D

Assigns the specified user-defined routine to the specified user-defined primitive type number.

FORMAT

```
GM_$PRIMITIVE_DISPLAY_2D (primitive_type, display_procedure_ptr, status)
```

INPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

display_procedure_ptr
Entry point for the application-supplied procedure that displays (GM_$PRIMITIVE_2D) commands of the specified primitive type, in GM_$PRIMITIVE_PTR_T format. This is a pointer to a procedure.

When a (GM_$PRIMITIVE_2D) command of the specified primitive type is encountered during display operations, the graphics metafile package calls the application-supplied procedure to display the command. Four input parameters are passed to the application-supplied procedure:

```
n_points: the number of points in point_array.  This is
          a 2-byte integer.

point_array: the list of points, transferred to display
             coordinates.  This is an array of pairs
             (x,y) of 2-byte integers.

n_parameters: the number of parameters in parameter_array.
              This is a 2-byte interger.

parameter_array: the list of parameters.  This is an
                 array of reals.
```

If you use a value of NIL for display_procedure_ptr, no routine is called at display time. You can use this to undo an assignment of a procedure to a specified user-defined primitive-type number.

In FORTRAN, pass procedure pointers as indicated in the description of GM_$REFRESH_SET_ENTRY. Use 0 (not NIL) to indicate a zero value.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Successive calls to GM_$PRIMITIVE_DISPLAY_2D for the same primitive type override previously defined entry points.

The user-supplied routine may contain only GPR drawing routine calls.

GM_$PRINT_FILE

Converts the current metafile to the specified file for subsequent printing on a hard-copy device.

FORMAT

```
GM_$PRINT_FILE (file_name, file_name_length, size, invert, print_style,
                bpi, status)
```

INPUT PARAMETERS

file_name
Pathname of the output file, in NAME_$PNAME_T format. This is an array of up to 256 characters.

If you specify a file name that already exists, the old contents of the file are overwritten.

file_name_length
Number of characters in the pathname. This is a 2-byte integer.

size
Pair (x, y) of coordinates, in GM_$POINT16_T. This is a two-element array of 2-byte integers. See the GM Data Types section for more information.

invert
Boolean (logical) value specifying whether to invert the file or not. Set to true to invert the file. Set to false to print the file without inverting it.

print_style

Type of output file to be created, in GM_$PRINT_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$GMF — Specifies that you want to copy a metafile to a bitmap for storage in a graphics map file (GMF).

GM_$OUT1 — Specifies that you want to create a vector command file. Only one plane, plane 0, is stored.

GM_$POSTSCRIPT — Specifies that you want to create a PostScript file

For print_style = GM_$OUT1, all coordinates are transformed to display coordinates in accordance with the size parameter of the GM_$PRINT_... routine that you used to create the vector command file. In the GM_$OUT1 file, the origin of coordinates is the top left, not the bottom left as in the metafile. The GM_$OUT1 file is scaled to the size parameter using the standard 95% rule that one dimension fills 95% of the size block, and the other dimension does not overflow the block.

For print_style = GM_$GMF, the bpi value sets the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the specified number of bits per inch. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image.

bpi

Number of bits per inch in the output GMF (graphics map file). This is a 2-byte integer.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM Data Types section for more information.

USAGE

GM_$PRINT_FILE prints the primary segment of the file. To print some other segment, you may make the desired segment the primary segment, call GM_$PRINT_FILE, and then restore the previous value of the primary segment.

GM_$PRINT_FILE_PART

Converts part of the current metafile to the specified file for subsequent printing on a hard-copy device.

FORMAT

```
GM_$PRINT_FILE_PART (bounds, file_name, file_name_length, size, invert,
                     print_style, bpi, status)
```

INPUT PARAMETERS

bounds
Part of this file to be printed, in segment coordinates of the primary segment, in GM_$BOUNDSREAL_T format. This is a four-element array of real values (xmin, ymin, xmax, ymax).

file_name
Pathname of the output file, in NAME_$PNAME_T format. This is an array of up to 256 characters.

file_name_length
Number of characters in the pathname. This is a 2-byte integer.

If you specify a file name that already exists, the old contents of the file are overwritten.

size
Pair (x,y) of coordinates, in GM_$POINT16_T. This is a two-element array of 2-byte integers.

invert
Boolean (logical) value specifying whether to invert the file or not. Set to true to invert the file. Set to false to print the file without inverting it.

print_style
Type of output file to be created, in GM_$PRINT_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$GMF — Specifies that you want to copy a metafile to a bitmap for storage in a graphics map file (GMF).

GM_$OUT1 — Specifies that you want to create a vector command file.

GM_$POSTSCRIPT — Specifies that you want to create a PostScript file.

For print_style = GM_$OUT1, all coordinates are transformed to display coordinates in accordance with the size parameter of the GM_$PRINT_... routine that you used to create the vector command file. In the GM_$OUT1 file, the origin of coordinates is the top left, not the bottom left as in the metafile. The GM_$OUT1 file is scaled to the size parameter using the standard 95% rule that one dimension fills 95% of the size block, and the other dimension does not overflow the block.

For print_style = GM_$GMF, the bpi value sets the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the specified number of bits per inch. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image.

bpi

Number of bits per inch in the output GMF (graphics map file). This is a 2-byte integer.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM Data Types section for more information.

USAGE

GM_$PRINT_FILE_PART prints the primary segment of the file. To print some other segment, you may make the desired segment the primary segment, call GM_$PRINT_FILE_PART, and then restore the previous value of the primary segment.

GM_$RECTANGLE_[16,32,REAL]

Inserts a command into the current segment : draw a rectangle with sides parallel to the x and y axes.

FORMAT

```
GM_$RECTANGLE_16 (point1, point2, fill, status)

GM_$RECTANGLE_32 (point1, point2, fill, status)

GM_$RECTANGLE_REAL (point1, point2, fill, status)
```

INPUT PARAMETERS

point1, point2

The coordinates of two diagonally opposite corners, each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$RECTANGLE_16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$RECTANGLE_32

GM_$POINTREAL
: A two-element array of real values for GM_$RECTANGLE_REAL

fill

A Boolean (logical) value which specifies whether to fill the rectangle or not. Set the parameter to true to fill the rectangle; set it to false for an unfilled rectangle.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_RECTANGLE_[16,32,REAL] to retrieve the parameters of a rectangle command inserted by GM_$RECTANGLE_[16,32,REAL].

Currently, you must use GM_$INQ_RECTANGLE_16 if the stored data type is GM_$16; you must use GM_$INQ_RECTANGLE_32 or _REAL if the stored data type is GM_$32.

Before supplying coordinate data to GM_$RECTANGLE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$REFRESH_SET_ENTRY

Specifies a user-defined routine to be called when the display is refreshed as a result of a DM refresh window or POP command.

FORMAT

```
GM_$REFRESH_SET_ENTRY (refresh_procedure_ptr, status)
```

INPUT PARAMETERS

refresh_procedure_ptr

refresh_procedure_ptr
Entry point for the application-supplied procedure to refresh the display, in GM_$REFRESH_PTR_T format. This is a pointer to procedure.

In direct mode, when the Display Manager refreshes the window in which the GM bitmap is contained, the specified application-supplied procedure is called. Two input parameters are passed to the application-supplied procedure:

```
unobscured    When false, this Boolean value indicates
              that the window is obscured.

position_changed
              When true, this Boolean value indicates
              that the window has moved or grown since
              the display was released.
```

The "pointer to procedure" data type is an extension to the Pascal language. See the *DOMAIN Pascal Language Reference* for an explanation of this extension.

This routine requires you to pass procedure pointers. To do so in FORTRAN programs, use the following technique. First declare the subroutines to be passed as EXTERNAL. Then pass their names using the IADDR function to simulate the Pascal pointer mechanism. For example:

```
EXTERNAL REFRESH_WINDOW
.
.
.
CALL GPR_$SET_REFRESH_ENTRY (IADDR(REFRESH_WINDOW),IADDR,
STATUS)
```

In FORTRAN, use 0 (not NIL) to indicate a zero value.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Successive calls to GM_$REFRESH_SET_ENTRY override previously defined entry points.

In within-GPR mode, use GPR_$SET_REFRESH_ENTRY.

GM_$REPLACE_INQ_FLAG

Returns the current value of the replace flag (Obsolete). New programs use GM_$MODELCMD_INQ_MODE.

FORMAT

```
GM_$REPLACE_INQ_FLAG (yes_no, status)
```

OUTPUT PARAMETERS

yes_no
A Boolean value indicating whether the replace flag is set. True indicates that the flag is set (new commands replace the current command); false indicates that the flag is cleared (new commands are inserted after the current command). The default value is false.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This routine is functional, but new programs should use GM_$MODELCMD_INQ_MODE.

GM_$REPLACE_SET_FLAG

Sets or clears a flag which causes subsequent commands to replace the current command rather than being inserted after it (Obsolete). New programs use GM_$MODELCMD_SET_MODE.

FORMAT

```
GM_$REPLACE_SET_FLAG (yes_no, status)
```

INPUT PARAMETERS

yes_no

A Boolean value indicating whether the replace flag is set. Use true to set the flag (new commands replace the current command); use false to clear the flag (new commands are inserted after the current command).

The default value is false (new commands are inserted after the current command).

When the replace flag is set, and you call a routine which creates a command of the same command type as the current command, the new command replaces the current command.

If you call a routine which creates a different command type, the replace flag is automatically cleared and the new command is inserted after the current command.

Changing the current command (for example, by calling GM_$COMMAND_DELETE) automatically clears the replace flag.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This routine is functional, but new programs should use GM_$MODELCMD_SET_MODE.

Only primitive and instance command types may be replaced. The replace flag may only be set if the current command is a primitive or instance command.

GM_$SEGMENT_CLOSE

Closes the current segment, saving revisions or not.

FORMAT

```
GM_$SEGMENT_CLOSE (save, status)
```

INPUT PARAMETERS

save

A Boolean (logical) value that indicates whether to save revisions. Set to true to save revisions; set to false not to save revisions.

You must set save to true. Do not assume that it is true by default.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$SEGMENT_COPY

Copies the entire contents of another segment into the current segment.

FORMAT

```
GM_$SEGMENT_COPY (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to be copied, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The entire contents of the specified segment are copied into the current segment after the current command. The current command is set equal to the last copied command.

You cannot copy a segment from one file to another file.

Use the following procedure to make a new segment named "newcopy," which is an exact copy of an existing segment. The identification of the existing segment is 'source_seg_id':

```
GM_$SEGMENT_CREATE ('newcopy', 7, segment_id, status);
GM_$SEGMENT_COPY (source_seg_id, status)
GM_$SEGMENT_CLOSE (true, status)
```

The two copies may then be edited independently and instanced independently.

GM_$SEGMENT_CREATE

Creates a new segment.

FORMAT

```
GM_$SEGMENT_CREATE (name, name_length, segment_id, status)
```

INPUT PARAMETERS

name

The pathname of the segment, in NAME_$PNAME_T format. This is a character string.

Currently, the segment name is truncated to twelve characters.

Segments in the same file must have different segment names. Note that "SEG" and "seg" are different segment names; the comparison is case-sensitive.

Verification that each name is unique carries a performance penalty. Therefore, you have the option of not naming segments and using the the segment identification number to reference segments. To create an unnamed segment, set the value for name to 0:

```
GM_$SEGMENT_CREATE ( '', 0, seg_id, status)
```

You can use GM_$SEGMENT_RENAME to give a name to an unnamed segment or to remove the name of a segment.

name_length

The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

segment_id

The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The segment name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another

You must close the current segment before creating a new segment.

When a segment is created, its pickable and visible values are set to 255.

For a segment name, you can use any collection of byte values of length 1 through 12. Trailing blanks in segment names are not discarded.

If you are careful, you may use a number for the segment name:

```
VAR
     number: integer32;
  .
  .
  .
GM_$SEGMENT_CREATE(number,4,seg_id,status);
```

GM_$SEGMENT_DELETE

Deletes the current segment.

FORMAT

```
GM_$SEGMENT_DELETE (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

There must be no references to the deleted segment.

If you delete a segment, its identification number will be reassigned. The smallest unused identification number is reassigned first.

You may not delete the file's primary segment. If you attempt to do so, you will get this error message: gm_$illegal_value.

GM_$SEGMENT_ERASE

Deletes all commands in the current segment.

FORMAT

```
GM_$SEGMENT_ERASE (status)
```

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

After this routine is performed, the current segment is still the current segment, but it contains no commands.

GM_$SEGMENT_INQ_BOUNDS

Returns the bounds of a segment.

FORMAT

```
GM_$SEGMENT_INQ_BOUNDS (bounds,status)
```

INPUT PARAMETERS

seg_id
Segment ID in GM_$SEGMENT_ID_T format. This is a positive 4-byte integer.

OUTPUT PARAMETERS

bounds
Bounds of the segment in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM Data Types section for more information.

USAGE

Use this call to obtain the coordinates of the bottom left-hand boundary and the top right-hand boundary of the segment.

Use GM_$FILE_INQ_BOUNDS to obtain the bounds of the primary segment of a file.

Use GM_$COMMAND_INQ_BOUNDS to obtain the bounds of the current command.

GM_$SEGMENT_INQ_COUNT

Returns the number of segments in this metafile and a segment number guaranteed to be greater than or equal to the largest segment number.

FORMAT

```
GM_$SEGMENT_INQ_COUNT (count, max_segid, status)
```

OUTPUT PARAMETERS

count
The number of segments in the metafile, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

max_segid
A number greater than or equal to the largest segment ID in the file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

When you retrieve the count and maximum segment ID, you can then look at every segment by checking segment numbers from 0 to this maximum value (0 is used).

GM_$SEGMENT_INQ_CURRENT

Returns the name, segment identification, and number of instances of the current segment.

FORMAT

```
GM_$SEGMENT_INQ_CURRENT (name, name_length, segment_id,
                         n_instances, status)
```

OUTPUT PARAMETERS

name
The pathname of the segment, in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

segment_id
The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

n_instances
The number of instances of the segment. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The returned segment number must be used for creating references to this segment within other segments.

GM_$SEGMENT_INQ_ID

Returns the segment identification and the number of instances of the named segment.

FORMAT

```
GM_$SEGMENT_INQ_ID (name, name_length, segment_id, n_instances, status)
```

INPUT PARAMETERS

name
The pathname of the segment. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

segment_id
The identification number assigned to the segment of specified name, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

In creating instances of (references to) this segment within other segments, you must use the returned segment identification number.

n_instances
The number of instances of the segment. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

GM_$SEGMENT_INQ_ID is complementary to GM_$SEGMENT_INQ_NAME.

Only the current file is searched to identify the segment number and the number of instances of the named segment.

GM_$SEGMENT_INQ_NAME

Returns the name of the segment with the specified segment identification number.

FORMAT

```
GM_$SEGMENT_INQ_NAME (seg_id, name, name_length, n_instances, status)
```

INPUT PARAMETERS

segment_id
The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

name
The pathname of the segment, in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

n_instances
The number of instances of the segment. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

GM_$SEGMENT_INQ_NAME is complementary to GM_$SEGMENT_INQ_ID.

GM_$SEGMENT_INQ_PICKABLE

Returns the pickable value of the specified segment.

FORMAT

```
GM_$SEGMENT_INQ_PICKABLE (segment_id, pickable, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the pickable value is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

pickable
The pickable value of the specified segment. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$SEGMENT_INQ_TEMPORARY

Returns whether the specified segment is temporary or not.

FORMAT

```
GM_$SEGMENT_INQ_TEMPORARY (segment_id, temporary, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the temporary/permanent status is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

temporary
A Boolean (logical) value that indicates whether the segment is temporary. A value of true indicates that the segment is temporary; false indicates that the segment is permanent.

Temporary segments are deleted when the file is closed.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$SEGMENT_INQ_VISIBLE

Returns the visible value of the specified segment.

FORMAT

```
GM_$SEGMENT_INQ_VISIBLE (segment_id, visible, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the visible value is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

visible
The visible value of the specified segment. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$SEGMENT_OPEN

Reopens an existing segment.

FORMAT

```
GM_$SEGMENT_OPEN (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to open, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

You must close the current segment before opening another segment.

Use GM_$SEGMENT_INQ_CURRENT to get the identification number of the current segment.

Currently, you cannot open a segment in a file that is open for read access. To open a segment, the file containing the segment must be open in write or read-write access mode. Otherwise, this error message is returned: gm_$illegal_value.

GM_$SEGMENT_RENAME

Renames an existing segment.

FORMAT

```
GM_$SEGMENT_RENAME (segment_id, name, name_length, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to rename, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The segment number remains the same when you rename the segment.

name
The new name of the segment in NAME_$PNAME_T format. This is an array of up to 256 characters.

If another segment already has the new name, you receive an error message, and the old name is not changed.

name_length
The number of characters in the new name of the segment. This is a 2-byte integer.

Currently, the segment name is truncated to twelve characters.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Verification that each name is unique carries a performance penalty. Therefore, you have the option of not naming segments and using the segment identification number to reference segments. To create an unnamed segment, set the value for name to 0:

```
GM_$SEGMENT_CREATE ( '', 0, seg_id, status)
```

You can use GM_$SEGMENT_RENAME to give a name to an unnamed segment or to remove the name of a segment.

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

To find the segment_id of an existing segment for which you know the name, use GM_$SEGMENT_INQ_ID.

GM_$SEGMENT_SET_PICKABLE

Assigns a pickable value to the specified segment.

FORMAT

```
GM_$SEGMENT_SET_PICKABLE (segment_id, pickable, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the pickable value is to be changed, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

pickable
The pickable value for the specified segment. This is a 4-byte integer.

When a segment is created, its pickable value is initialized to 255.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$SEGMENT_SET_TEMPORARY

Makes the specified segment temporary or not. Temporary segments are deleted when the file is closed.

FORMAT

```
GM_$SEGMENT_SET_TEMPORARY (segment_id, temporary, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to make temporary, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

temporary
A Boolean value that indicates whether the segment is temporary. Set to true to make temporary; set to false to make permanent.

When a segment is created, it is made permanent (temporary = false).

A temporary segment is useful for picture data that you want to display now but not store for future use.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$SEGMENT_SET_VISIBLE

Assigns a visible value to the specified segment.

FORMAT

```
GM_$SEGMENT_SET_VISIBLE (segment_id, visible, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the visible value is to be changed, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

visible
The visible value for the specified segment. This is a 4-byte integer.

When a segment is created, its visible value is initialized to 255.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$SEGMENT_SET_VISIBLE lets you display a picture without certain segments.

GM_$TAG

Inserts a comment into the current segment.

FORMAT

```
GM_$TAG (string, string_length, status)
```

INPUT PARAMETERS

string
The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length
The length of the string. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_TAG to get the value stored for the current GM_$TAG command.

Descriptor tags in a stroke font file must be entered in the file in capital letters, for example, WIDTH.

GM_$TAG_LOCATE

Looks for the specified tag in the specified range of segments and returns the segment ID of the lowest numbered segment in which the tag is found.

FORMAT

```
GM_$TAG_LOCATE (string, string_length, min, max, segment_id, status)
```

INPUT PARAMETERS

string
The string to be searched for, in GM_$STRING_T format. This is an array of up to 120 characters.

The string to be matched is passed through the pathname wildcard parser, as described in *DOMAIN System Command Reference* manual. To guarantee noninterference with the wildcard parser, you may place an escape character (@) between every byte of the string you wish to search for.

string_length
The length of the string to be searched for. This is a 2-byte integer.

min
The smallest segment number to search, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

max
The largest segment number to search, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

segment_id
The number of the segment in which the tag was found, in GM_$SEGMENT_ID_T format. This is 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

To find all occurrences of a tag, you must make successive calls to GM_$TAG_LOCATE.

GM_$TERMINATE

Terminates the graphics metafile package and closes the display.

FORMAT

```
GM_$TERMINATE (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

When GM terminates, it currently resets color 1 to whatever it was when GM was initialized. This is true of color nodes only. If you use GM in borrow mode, the entire color map is reset when GM terminates. (The resetting is not by GM, but by GPR.)

Any open files are closed. Revisions to these files are saved.

Any open segments are closed, and revisions are saved.

GM_$TEXT_2D[16,32,REAL]

Inserts a command into the current segment: write a text string.

FORMAT

```
GM_$TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)
```

INPUT PARAMETERS

point

The coordinates of the point at which to locate text. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$TEXT_2D16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$TEXT_2D32

GM_$POINTREAL
: A two-element array of real values for GM_$TEXT_2DREAL

See the GM_$ Data Types section for more information.

The text is placed as follows: The first character of the text string is placed at the location you specify. This means that the origin of this character, as defined in the font, is placed at the specified location. Usually, the origin is the lower left-hand corner, excluding descenders.

rotate

The angle at which this text string is to be written, in degrees. This is a real variable. Use 0.0 degrees to specify left to right text. Other values indicate clockwise rotation. For example, -90.0 degrees specifies bottom to top.

string

The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length

The length of the string. This is a 2-byte integer.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_TEXT_2D[16,32,REAL] to retrieve the parameters of a text command inserted by GM_$TEXT_2D[16,32,REAL].

Before supplying coordinate data to GM_$TEXT_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$TEXT_BACKGROUND_VALUE

Inserts a command into the current segment: change the background value used when writing text.

FORMAT

```
GM_$TEXT_BACKGROUND_VALUE (value, status)
```

INPUT PARAMETERS

value

The value to use for the text background. This is a 4-byte integer.

The default value is -2. This sets the text background value equal to the viewport background value.

The value -1 makes the background transparent: the text background value is equal to the current display pixel value.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_TEXT_BACKGROUND_VALUE to get the value stored for the current GM_$TEXT_BACKGROUND_VALUE command.

GM_$TEXT_SIZE

Inserts a command into the current segment : use a different text size from the same font family.

FORMAT

```
GM_$TEXT_SIZE (size, status)
```

INPUT PARAMETERS

size

The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value.

The default text size is 10.0.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_TEXT_SIZE to retrieve the parameters of a text command inserted by GM_$TEXT_SIZE.

GM_$TEXT_VALUE

Inserts a command into the current segment: set the value used when writing text.

FORMAT

```
GM_$TEXT_VALUE (value, status)
```

INPUT PARAMETERS

value
The value that specifies the new value to use when writing text. This is a 4-byte integer.

The default value is 1.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$INQ_TEXT_VALUE to get the value stored for the current GM_$TEXT_VALUE command.

GM_$VIEWPORT_CLEAR

Clears the current viewport.

FORMAT

```
GM_$VIEWPORT_CLEAR (value, status)
```

INPUT PARAMETERS

value
The value to which all pixels within the current viewport are to be set. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Only planes enabled by the current value of the plane mask are affected.

GM_$VIEWPORT_CREATE

Creates an additional viewport and makes it the current viewport.

FORMAT

```
GM_$VIEWPORT_CREATE (bounds, viewport_id, status)
```

INPUT PARAMETERS

bounds
The bounds of the new viewport, in GM_$BOUNDSREAL_T format. This is an array of four real values (xmin, ymin, xmin, ymax). See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is assigned by the GM package.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$INIT initializes the GM package and viewports, creates one viewport called viewport 1 which fills the display bitmap and makes it the selected viewport. Currently, viewports may not overlap, so you must change the bounds of viewport 1 before creating additional viewports. You must supply bounds for the new viewport, in bitmap coordinates. The GM package assigns a number to the viewport.

Use this procedure to change the original viewport to fill only the left half of the screen and create a second viewport in the center right of the screen:

```
bounds.xmin := 0.0; bounds.ymin := 0.0;
bounds.xmax := 0.5; bounds.ymax := 1.0;
GM_$VIEWPORT_SET_BOUNDS (bounds, status);
bounds.xmin := 0.6; bounds.ymin := 0.25;
bounds.xmax := 1.0; bounds.ymax := 0.75;
GM_$VIEWPORT_CREATE (bounds, viewport_id, status);
```

GM_$VIEWPORT_DELETE

Deletes a viewport.

FORMAT

```
GM_$VIEWPORT_DELETE (viewport_id, status)
```

INPUT PARAMETERS

viewport_id
The number assigned by the GM package to the viewport you wish to delete. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Because viewports currently may not overlap, you must delete all but one viewport if a single viewport is to be expanded to fill the entire GM bitmap.

GM_$VIEWPORT_INQ_BACKGROUND_VALUE

Returns the pixel value used for the background of the specified viewport.

FORMAT

```
GM_$VIEWPORT_INQ_BACKGROUND_VALUE (viewport_id, value, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

OUTPUT PARAMETERS

value
The value to use for the viewport background. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$VIEWPORT_INQ_BACKGROUND_VALU | E
```

Use GM_$VIEWPORT_SET_BACKGROUND_VALUE to change the background value of the specified viewport.

GM_$VIEWPORT_INQ_BORDER_SIZE

Returns the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

FORMAT

```
GM_$VIEWPORT_INQ_BORDER_SIZE (border_unit, border_size, status)
```

OUTPUT PARAMETERS

border_unit

The units for border size, in GM_$BORDER_UNIT_T format. This is a 2-byte integer. One of the following values is returned:

GM_$FRACTIONS
: Expresses edge width as fraction of the total 6M bitmap size.

GM_$PIXELS
: Default border type. Expresses edge width in pixels.

border_size

The size of the border, specified as left, bottom, right, top. This is an array of four real values (left, bottom, right, top).

The default border type is in pixels, and the default width is 1,1,1,1.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$VIEWPORT_INQ_BORDER_SIZE returns the size of the four edges of the current viewport. If border_unit = GM_$PIXELS, edge widths are expressed in pixels. If border_unit = GM_$FRACTIONS, edge widths are expressed as fractions of the total GM bitmap size.

Use GM_$VIEWPORT_SET_BORDER_SIZE to change the size of the border.

GM_$VIEWPORT_INQ_BOUNDS

Returns the bounds of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_BOUNDS (bounds, status)
```

OUTPUT PARAMETERS

bounds
The bounds of the current viewport, in GM_$BOUNDSREAL_T format. This is a four element array of real values (xmin, ymin, xmax, ymax). See the GM_$ Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$VIEWPORT_INQ_BOUNDS returns the bounds of the current viewport, as fractions of the total GM bitmap size.

GM_$VIEWPORT_INQ_CURRENT

Returns the number of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_CURRENT (viewport_id, status)
```

OUTPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is assigned by the GM package.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

If there is no current viewport, a GM_$NO_CURRENT_VIEWPORT error is returned.

GM_$VIEWPORT_INQ_GRIDS

Returns the number and type of grids for a viewport.

FORMAT

```
GM_$VIEWPORT_INQ_GRIDS (flags, index, cnt, grid, status)
```

INPUT PARAMETERS

maxcnt
Length of the grid array. This is a 2-byte integer.

OUTPUT PARAMETERS

flags
Type of grid, in GM_GRID_FLAGS_T format. This is a 2-byte integer. Any combination of the following predefined values can be returned:

GM_$GRID_VISIBLE
Specifies a visible grid.

GM_$GRID_SNAPTO
Specifies a snap grid. This may be visible or invisible.

sindex
Index of the snap grid. This is a 2-byte integer.

cnt
Number of grids. This is a 2-byte integer.

grid
Array of grids, in GM_$ARRAY_T format. This is an array of [1 .. gm_$max_grid] of GM_$GRID_T. See the GM Data Types section for more information.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$VIEWPORT_SET_GRIDS to establish a grid for a viewport.

GM_$VIEWPORT_INQ_REFRESH_STATE

Returns the refresh state of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_REFRESH_STATE (refresh_state, status)
```

OUTPUT PARAMETERS

refresh_state

The refresh state of the viewport, in GM_$VIEW_REFRESH_T format. This is a 2-byte integer. One of the following values is returned:

GM_$REFRESH_INHIBIT

In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH, or when the display is refreshed as the result of a DM command which causes the window to be redrawn. Thus, calling GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state.

GM_$REFRESH_WAIT

(Default) In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH or when the display is refreshed as the result of a DM command which causes the window to be redrawn.

GM_$REFRESH_PARTIAL

Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten. This provides faster interactive drawing. You should, however, periodically clean up the accumulating inaccuracies by calling GM_$VIEWPORT_REFRESH to redisplay the viewport.

GM_$REFRESH_UPDATE

Every time you change any command in the file, this viewport is completely corrected if it is the current viewport.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

The viewport refresh states are defined under the routine GM_$VIEWPORT_SET_REFRESH_STATE.

GM_$VIEWPORT_MOVE

Translates the current viewport, carrying the view with it.

FORMAT

```
GM_$VIEWPORT_MOVE (translate, status)
```

INPUT PARAMETERS

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values.

The translation is expressed as fractions of the display bitmap size.

Currently, values which would cause part of the viewport to be moved outside the GM bitmap result in an error.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$VIEWPORT_REFRESH

Refreshes the current viewport.

FORMAT

```
GM_$VIEWPORT_REFRESH (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL

Converts segment coordinates to pixel coordinates.

FORMAT

```
GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL (p, q, status)
```

OUTPUT PARAMETERS

p

Segment coordinates of any point in the viewport segment, in GM_$POINT16_T format. This is a two-element array of 2-byte integers, 4-byte integers, or real numbers depending on which form of the call is used.

q

Pixel coordinates of the screen location whose coordinates were entered in parameter "p". This parameter uses GM_$POINT16_T format. This is a two-element array of 2-byte integers, 4-byte integers, or real numbers depending on which form of the call is used.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

This routine allows the user to inquire about the pixel location of a particular point in a segment. For example, if a rectangle is drawn from (0,0) to (100,100), the user can obtain the screen location (pixel coordinates) of the point (0,0), or any other point in the segment.

This call is useful when drawing grids. For example, consider drawing the lines of a grid every 100 segment coordinates. You can use this call to determine where the points (0,0), and (100,0) will be drawn. This allows you to determine the density of your grid.

GM_$VIEWPORT_SELECT

Makes a viewport the current viewport.

FORMAT

```
GM_$VIEWPORT_SELECT (viewport_id, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

You must create the viewport before you can select it.

GM_$VIEWPORT_SET_BACKGROUND_VALUE

Sets the pixel value used for the background of the specified viewport.

FORMAT

```
GM_$VIEWPORT_SET_BACKGROUND_VALUE (viewport_id, value, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

value
The value to use for the viewport background. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$VIEWPORT_SET_BACKGROUND_VALU | E
```

Use GM_$VIEWPORT_INQ_BACKGROUND_VALUE to retrieve the background value of the specified viewport.

GM_$VIEWPORT_SET_BORDER_SIZE

Specifies the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

FORMAT

```
GM_$VIEWPORT_SET_BORDER_SIZE (border_unit, border_size, status)
```

INPUT PARAMETERS

border_unit

The units for border size, in GM_$BORDER_UNIT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$FRACTIONS
: Expresses edge width as fractions of the total GM bitmap size.

GM_$PIXELS
: Default border type. Expresses edge width in pixels.

border_size

The size of the border, specified as left, bottom, right, top. This is an array of four real values (left, bottom, right, top).

The default border type is in pixels, and the default width is 1,1,1,1.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Viewport borders are drawn with color value 1 for compatibility with monochrome nodes. For the same reason, the graphics metafile package sets the color map for color value 1 to white. With a color node, you may want to use the viewport background color to differentiate viewports from the overall display or the window background. Changing the color map to black is usually not practical because the cursor is also set to color value 1. An alternative is to create the viewport, set the border width to 0 pixels, and then refresh the viewport.

GM_$VIEWPORT_SET_BORDER_SIZE sets the size of the four edges of the current viewport. If border_unit = GM_$PIXELS, edge widths are expressed in pixels. If border_unit = GM_$FRACTIONS, edge widths are expressed as fractions of the total GM bitmap size.

Use GM_$VIEWPORT_INQ_BORDER_SIZE to retrieve the size of the border.

GM_$VIEWPORT_SET_BOUNDS

Changes the display bounds for the current viewport.

FORMAT

```
GM_$VIEWPORT_SET_BOUNDS (bounds, status)
```

INPUT PARAMETERS

bounds
The bounds of the new viewport, in GM_$BOUNDSREAL_T format. This is a four-element array of real values (xmin, ymin, xmax, ymax). See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

GM_$VIEWPORT_SET_BOUNDS sets the bounds of the current viewport. You must provide two diagonally opposite corners. Coordinates are expressed as fractions of the total display bitmap size: bottom left = (0.0, 0.0); top right = (1.0, 1.0).

Currently, viewports may not overlap.

Use this procedure to change the bounds of the current viewport to fill only the left half of the screen.

```
bounds.xmin := 0.0; bounds.ymin := 0.0;
bounds.xmax := 0.5; bounds.ymax := 1.0;
GM_$VIEWPORT_SET_BOUNDS (bounds, status);
```

GM_$VIEWPORT_SET_GRIDS

Specifies the number and type of grids for a viewport.

FORMAT

```
GM_$VIEWPORT_SET_GRIDS (flags, index, cnt, grid, status)
```

INPUT PARAMETERS

flags
Attributes of the snap grid, in GM_$GRID_FLAGS_T format. This is a 2-byte integer. Specify any combination of the following predefined values:

GM_$GRID_VISIBLE
Specifies a visible snap grid.

GM_$GRID_SNAPTO
Specifies a snap grid. This may be visible or invisible.

sindex
Index of the snap grid. This is a 2-byte integer.

cnt
Number of grids. This is a 2-byte integer between 1 and GM_$MAX_GRID.

grid
Array of grids, in GM_$ARRAY_T format. This is an array of [1 .. cnt] of GM_$GRID_T. See the GM Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM Data Types section for more information.

USAGE

Grids are input aids; they are not part of the metafile -- they are strictly attributes of a viewport.

If snapping is enabled for a particular viewport (for which a grid is also defined), then positions reported by GM_$EVENT_WAIT in that viewport will be snapped to the nearest grid point.

A given viewport may have an array of grids associated with it (up to 4 grids, currently). This allows the user to define major and minor grids, or even to define more complicated grids.

Use GM_$VIEWPORT_INQ_GRID to inquire grids for the current viewport.

The snap grid index indicates which of the grids in the array is to be used for snapping.

The flag array is relevant only to the grid given by the snap grid index.

To determine grid density use GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL.

A call to GM_$VIEWPORT_SET_GRIDS causes the viewport to be refreshed if the viewport is being displayed.

GM_$VIEWPORT_SET_REFRESH_STATE

Sets the refresh state of the current viewport.

FORMAT

```
GM_$VIEWPORT_SET_REFRESH_STATE (refresh_state, status )
```

INPUT PARAMETERS

refresh_state

The refresh state of the viewport, in GM_$VIEW_REFRESH_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$REFRESH_INHIBIT

In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH, or when the display is refreshed as the result of a DM command which causes the window to be redrawn. Thus, calling GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state.

GM_$REFRESH_WAIT

(Default) In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH or when the display is refreshed as the result of a DM command which causes the window to be redrawn.

GM_$REFRESH_PARTIAL

Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten. This provides faster interactive drawing. You should, however, periodically clean up the accumulating inaccuracies by calling GM_$VIEWPORT_REFRESH to redisplay the viewport.

Partial refresh does not always update the viewport accurately. For accuracy in incremental updating, use GM_$REFRESH_UPDATE. Extensive use of partial refresh may necessitate use of GM_$VIEWPORT_REFRESH.

GM_$REFRESH_UPDATE

Every time you change any command in the file, this viewport is completely corrected.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$VIEW_SCALE

Scales the view under the current viewport, keeping the specified point fixed.

FORMAT

```
GM_$VIEW_SCALE (scale, point, status)
```

INPUT PARAMETERS

scale

The value by which to multiply the view scale factor. This is a real value.

point

An (x,y) pair indicating the fixed point on screen, in GM_POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The point (point.x,point.y) on the screen (expressed in fraction-of-bitmap coordinates) is kept fixed during this rescaling.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use the following to rescale the screen by a scaling factor "scale" AND move the point (point.x,point.y) on the screen (in fraction-of-bitmap coordinates) to the center of the viewport, all in one operation:

```
{assumes scale not equal to 1.0}

GM_$VIEWPORT_INQ_BOUNDS (vbounds, status);
vcenter_x := 0.5 * (vbounds.xmax + vbounds.xmin);
vcenter_y := 0.5 * (vbounds.ymax + vbounds.ymin);
point1.x := (vcenter_x - point.x * scale)/(1.0 - scale);
point1.y := (vcenter_y - point.y * scale)/(1.0 - scale);
GM_$VIEW_SCALE(scale, point1,status);
```

GM_$VIEW_TRANSFORM

Rotates the view under the current viewport, keeping the specified point (in fraction-of-bitmap coordinates) fixed.

FORMAT

```
GM_$VIEW_TRANSFORM (rotate, point, status)
```

INPUT PARAMETERS

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

point

An (x,y) pair indicating the fixed point on the screen, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The point (point.x,point.y) on the screen (expressed in fraction-of-bitmap coordinates) is kept fixed during this transformation.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$VIEW_TRANSLATE

Translates the view under the current viewport.

FORMAT

```
GM_$VIEW_TRANSLATE (translate, status)
```

INPUT PARAMETERS

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The translation is specified in bitmap coordinates, that is, as fractions of the display bitmap.

A positive x translation moves the viewport to the right over the view, so that the picture on the display appears to move to the left. A positive y translation moves the viewport up over the view, so that the picture on the display appears to move down.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

GM_$VISIBLE_INQ_MASK

Returns the value of the visible mask.

FORMAT

```
GM_$VISIBLE_INQ_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The visible mask value. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$VISIBLE_SET_MASK to change the current value of the visible mask.

GM_$VISIBLE_INQ_THRESHOLD

Returns the value of the visible threshold.

FORMAT

```
GM_$VISIBLE_INQ_THRESHOLD (threshold, status)
```

OUTPUT PARAMETERS

threshold
The visible threshold value. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$VISIBLE_SET_THRESHOLD to change the current value of the visible threshold.

GM_$VISIBLE_SET_MASK

Sets the value of the visible mask.

FORMAT

```
GM_$VISIBLE_SET_MASK (mask, status)
```

INPUT PARAMETERS

mask

The visible mask value. This is a 4-byte integer.

The visible mask is initialized to 16#7FFFFFFF (all nonzero segments visible).

The visible mask is BIT-ANDed with the segment visible number. If the result is nonzero, the segment may be visible. Both the visible mask and visible threshold must be satisfied for a segment to be visible.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$VISIBLE_INQ_MASK to retrieve the current value of the visible mask.

GM_$VISIBLE_SET_THRESHOLD

Sets the visible threshold.

FORMAT

```
GM_$VISIBLE_SET_THRESHOLD (threshold, status)
```

INPUT PARAMETERS

threshold

The visible threshold value. This is a 4-byte integer.

The visible threshold is initialized to 1 (all nonzero segments visible).

If the segment visible number is greater than or equal to the visible threshold, the segment may be visible. Both the visible mask and visible threshold must be satisfied for a segment to be visible.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GM_$ Data Types section for more information.

USAGE

Use GM_$VISIBLE_INQ_THRESHOLD to retrieve the current value of the visible threshold.

ERRORS

GM_$ABLOCK_ID_INVALID
: The ablock identification number is not valid. Use the number assigned when you created the ablock.

GM_$ABLOCK_NOT_CREATED
: You must create an ablock before you can use it.

GM_$ACLASS_ID_INVALID
: The aclass identification number you used is not valid.

GM_$ALREADY_INITIALIZED
: You initialize the GM package only once during a session of using it.

GM_$ANOTHER_SEGMENT_IS_OPEN
: Only one segment may be open at a time.

GM_$ATTRIBUTE_VALUE_INVALID
: The attribute value is not valid.

GM_$BOUNDS_INVALID
: The bounds specified for displaying part of a segment or file do not satisfy the requirement that the minimum value be less than the maximum value.

GM_$CANT_DELETE_FONT_FAMILY_IN_USE
: You cannot delete a font family that is referenced by the current file.

GM_$CANT_DELETE_INSTANCED_SEGMENT
: You cannot delete a segment if it is instanced by other segments in the file.

GM_$COMMAND_TYPE_DOESNT_MATCH
: The current command does not match the type specified in inquire operation.

GM_$COORDINATE_CONVERSION_OVERFLOW
: You have supplied a value to a coordinate conversion routine, GM_$COORD..., which cannot be converted.

GM_$DATA_COERCE_NEEDED
: To use the data in the format you have supplied, you must convert it.

GM_$FILE_ID_INVALID
: The file identification number you used is not valid.

GM_$FILE_NAME_NOT_FOUND
: The file name you gave is not valid.

GM_$FONT_FAMILY_ID_INVALID
: When you reference a font family, you must use the font family id.

GM_$FONT_FAMILY_NAME_ALREADY_USED
: You may not rename a font family to have the same name as another font family.

GM_$FONT_FAMILY_NAME_NOT_FOUND
: You must include a font family before you can use it.

GM_$ILLEGAL_VALUE
: One of the input parameters that you supplied to the GM package has an illegal value.

GM_$ILLEGAL_SELF_INSTANCE
A segment may not instance itself, directly or indirectly.

GM_$INPUT_EVENT_TYPE_INVALID
You must use the event types associated with input routines.

GM_$INVALID_POLYLINE_OPTIONS
The only options for a polyline are open, closed, or closed and filled. When you specify filled, you must also specify closed.

GM_$NAME_LENGTH_INVALID
The limitation on the number of characters in a name is 12.

GM_$MODULE_CODE
GM module

GM_$NEGATIVE_CIRCLE_RADIUS
The radius of a circle must be a positive value.

GM_$NO_CURRENT_COMMAND
For editing procedures such as picking, you must have a current command.

GM_$NO_CURRENT_FILE
You must have a file open. If you have more than one file open and you close the current file, you must select another current file.

GM_$NO_CURRENT_SEGMENT
You must create or open a segment.

GM_$NO_CURRENT_VIEWPORT
You must have a current viewport. The current viewport is the last viewport created or selected.

GM_$NO_FONT_FAMILY_INCLUDED
You must include a font family before you can use it. See GM_$FONT_FAMILY_INCLUDE.

GM_$NO_GM_BITMAP_EXISTS
When you initialize the GM package, the bitmap size is not defined. The procedure GM_$INQ_BITMAP_SIZE cannot return a valid value until you define the size.

GM_$NO_PICK_MATCHES_FOUND
The command or segment that you searched for was not found.

GM_$NOTHING_DISPLAYED_IN_VIEWPORT
You must have displayed a segment in the specified or current viewport before calling this routine.

GM_$NOT_INITIALIZED
You must initialize the GM package before you can use it.

GM_$OPERATION_OK
Normal status

GM_$PICK_LIST_EMPTY
Only picked segments are included on the pick list. Use GM_$PICK_SEGMENT to list a segment.

GM_$PICK_LIST_NOT_INITIALIZED
To use a pick list, you must first initialize it.

GM_$PICK_LIST_TOO_LONG
The limitation on the number of segments is 32 in a pick list.

GM_$SEGMENT_ID_INVALID
The segment identification number you used is not valid.

GM_$SEGMENT_LOCKED_BY_PICK
You may not delete or edit a segment included in a list of picked segments.

GM_$SEGMENT_NAME_ALREADY_USED
Each segment name must be unique.

GM_$SEGMENT_NAME_NOT_FOUND
The segment name is not in the file.

GM_$TOO_MANY_ABLOCKS
The limitation on the number of ablocks is 40.

GM_$TOO_MANY_FILES
The number of files is limited to 16.

GM_$TOO_MANY_FONT_FAMILIES
The limitation on the number of font families is 8.

GM_$TOO_MANY_SEGMENTS
The limitation on the number of segments is 16384.

GM_$TOO_MANY_VIEWPORTS
The limitation on the number of viewports is 64.

GM_$VIEWPORT_BOUNDS_INVALID
Viewports may not overlap. Space outside of viewports is empty.

GM_$VIEWPORT_DOESNT_EXIST
You must create a viewport before you can use it.

GM_$VIEWPORT_ID_INVALID
You must use the viewport number assigned by GM_$VIEWPORT_CREATE.

GM_$WRONG_DISPLAY_MODE
Each display mode has its advantages and limitations. See GM_$INIT.

GM_$WRONG_FILE_ACCESS_MODE
You must use the access mode specified in GM_$FILE_CREATE.

GMF

This section describes the data types, the call syntax, and the error codes for the GMF programming calls. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

DATA TYPES

GMF_$OPOS_T

A 2-byte integer. Specifies the file opening positions. One of the following pre-defined values:

GMF_$OVERWRITE
Provides write access; truncates file to BOF if it already exists.

GMF_$APPEND
Provides write access if file exists.

GMF_$READ
Provides read access only.

GMF_$MEMORY_T

A 65535-element array of 131070-byte integers. An array of two-byte integers.

GMF_$MEMORY_PTR_T

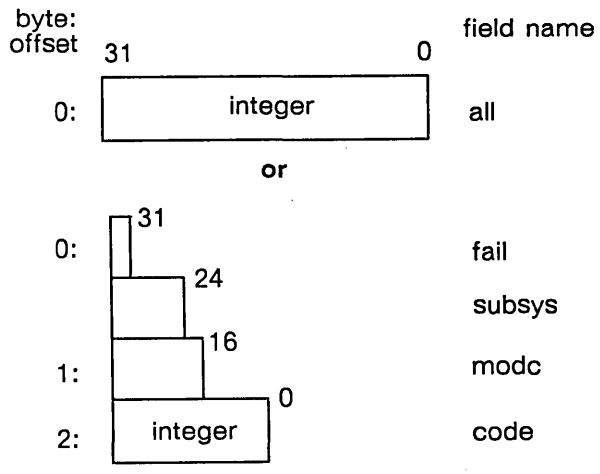
A 4-byte integer. A pointer to an array of type gmf_$memory_t.

STREAM_$ID_T

A 2-byte integer. Open stream identifier.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

GMF_$CLOSE

Closes a GMF.

FORMAT

```
GMF_$CLOSE (stream_id, status)
```

INPUT PARAMETERS

stream_id
The stream ID of the GMF to be closed, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream ID from the call to GMF_$OPEN that you used to open the GMF.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To open a GMF, use GMF_$OPEN.

GMF_$COPY_PLANE

Dumps a rectangular area of bits from virtual memory into a GMF.

FORMAT

```
GMF_$COPY_PLANE (stream_id, black_or_white, bpi, bit_pointer, x_dim, y_dim,
                 width, status)
```

INPUT PARAMETERS

stream_id

The stream ID of the GMF into which the image is to be stored, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream ID from the call to GMF_$OPEN that you used to open the GMF.

black_or_white

A Boolean variable. A value of TRUE means "1" bits are black and "0" bits are white. A value of FALSE means "1" bits are white and "0" bits are black. In the GMF, "1" bits are assumed to mean black. Thus if this parameter is false, the bits will be inverted as they are copied.

bpi

The number of bits per inch in the GMF. This information is stored in the GMF. It indicates the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the image's original size. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image. This is a 2-byte integer.

bit_pointer

A pointer to the upper left corner of the rectangular area to be stored, in GMF_$MEMORY_PTR_T format. This is a 4-byte integer. You obtain this value by calling the routine GPR_$INQ_BITMAP_POINTER.

x_dim

The x dimension of the rectangular area to be stored in the GMF. This is a 2-byte integer.

y_dim

The y dimension of the rectangular area to be stored in the GMF. This is a 2-byte integer.

width

The number of 16-bit words per scan line in the source bitmap. The value of this parameter is usually 64. The width must be at least 1/16 of the specified x-dim. For instance, if you are storing an area 400 bits wide in a GMF, the source bitmap must use at least 25 words to represent each scan line (row of dots). This is a 2-byte integer. You obtain this value by calling GPR_$INQ_BITMAP_POINTER.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To store an image in a GMF, you must have opened the GMF with the GMF_$OPEN call.

After storing an image in a GMF, close the GMF with the GMF_$CLOSE call.

The GMF_$COPY_PLANE call is a special case of the GMF_$COPY_SUBPLANE call.

GMF_$COPY_SUBPLANE

Dumps a rectangular area of bits from virtual memory into a GMF.

FORMAT

```
GMF_$COPY_SUBPLANE (stream_id, black_or_white, bpi, bit_pointer, x_dim, y_dim,
                    x_offset, y_offset, width, status)
```

INPUT PARAMETERS

stream_id
The stream ID of the GMF into which the image is to be stored, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream ID from the call to GMF_$OPEN that you used to open the GMF.

black_or_white
A Boolean variable. A value of TRUE means "1" bits are black and "0" bits are white. A value of FALSE means "1" bits are white and "0" bits are black. In the GMF, "1" bits are assumed to mean black. Thus if this parameter is false, the bits will be inverted as they are copied.

bpi
The number of bits per inch in the GMF. This information is stored in the GMF. It indicates the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the image's original size. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image. This is a 2-byte integer.

bit_pointer
A pointer to a bit which when offset by x_offset and y_offset gives the upper left corner of the rectangular area to be stored. This is a 4-byte integer. You obtain this value by calling the routine GPR_$INQ_BITMAP_POINTER.

x_dim
The x dimension of the rectangular area to be stored in the GMF. This is a 2-byte integer.

y_dim
The y dimension of the rectangular area to be stored in the GMF. This is a 2-byte integer.

x_offset
The x starting position of the rectangular area to be stored in the GMF relative to the bit whose address is given by bit_pointer. This is a 2-byte integer.

y_offset
The y starting position of the rectangular area to be stored in the GMF relative to the bit whose address is given by bit_pointer. This is a 2-byte integer.

width
The number of 16-bit words per scan line in the source bitmap. The value of this parameter is usually 64. The width must be at least 1/16 of the specified x-dim. For instance, if you are storing an area 400 bits wide in a GMF, the source bitmap must use at least 25 words to represent each scan line (row of dots). This is a 2-byte integer. You obtain this value by calling GPR_$INQ_BITMAP_POINTER.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To copy a plane into a GMF, you must have opened the GMF with the GMF_$OPEN call.

After copying a plane into a GMF, close the GMF with the GMF_$CLOSE call.

The GMF_$COPY_SUBPLANE call is a more general form of the GMF_$COPY_PLANE call.

GMF_$OPEN

Opens or creates a GMF.

FORMAT

```
GMF_$OPEN (name, name_length, start, stream_id, status)
```

INPUT PARAMETERS

name
Pathname, in NAME_$PNAME_T format.

name_length
The length of the name. This is a 2-byte integer.

start
Desired position in the file after open, in GMF_$OPOS_T format. This is a 2-byte integer. If you are opening the GMF to write data to it (to copy a plane or subplane into it), use one of these two constants:

GMF_$APPEND	sets the initial position to EOF.
GMF_$OVERWRITE	truncates the object to length 0 and sets the initial position to the beginning.

If you are opening the GMF to read data from it (restoring a plane), use this constant:

GMF_$READ	sets the initial position to the beginning without truncating the GMF.

If the specified GMF does not exist and you used GMF_$OPEN to create it, it does not matter what value this parameter has.

OUTPUT PARAMETERS

stream_id
The stream ID of the opened GMF, in STREAM_$ID_T format. This is a 2-byte integer. You use this value in subsequent GMF calls that refer to the opened GMF.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If the specified GMF does not exist, the call to GMF_$OPEN creates it.

You must call GMF_$OPEN before trying to read or write a GMF.

After opening a GMF with GMF_$OPEN, you must eventually close it by calling GMF_$CLOSE.

GMF_$RESTORE_PLANE

Copies an image back to the screen from a GMF.

FORMAT

```
GMF_$RESTORE_PLANE (stream_id, x_dim, y_dim, width, start, bpi, status)
```

INPUT PARAMETERS

stream_id
The stream ID of the GMF which is to supply the image, in STREAM_$ID_T format. This is a 2-byte integer. You obtain this parameter from the call to GMF_$OPEN you used to open the GMF.

x_dim
The x-dimension in bits of the display to which an image is to be restored. This is a 2-byte integer.

y_dim
The y-dimension in bits of the display to which an image is to be restored. This is a 2-byte integer.

width
The number of 16-bit words per scanline in the destination bitmap. This is a 2-byte integer.

start
The starting address in the destination bitmap. In Pascal this is a UNIV_PTR. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

bpi
Bits per inch as specified in GMF_$COPY_PLANE. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Before calling GMF_$RESTORE_PLANE, you must use GPR_$INIT to place the node in borrow-display mode.

The size of the area to be restored is the same as the size of the area you originally copied into the GMF. This information is contained in the GMF.

The area to be restored is determined by the bit-pointer specified in the GMF_$RESTORE_PLANE call and the size data in the GMF. If this area runs off the right side or the bottom of the screen, the GMF manager restores only the portion of the stored image that fits on the screen.

To restore a plane from a GMF, you must have opened the GMF with the GMF_$OPEN call.

After restoring a plane from a GMF, you should close the GMF with the GMF_$CLOSE call.

ERRORS

GMF_$BAD_BPI
: Bits/inch parameter is negative.

GMF_$BAD_POS
: Opening position parameter is illegal.

GMF_$BAD_WPL
: 16 bit words/line parameter is too small for x dim.

GMF_$BAD_X_DIM
: X-dimension parameter is not positive.

GMF_$BAD_Y_DIM
: Y-dimension parameter is not positive.

GMF_$NOT_GMF
: Opened file not a GMF metafile.

STATUS_$OK
: Successful completion.

GPR

This section describes the data types, the call syntax, and the error codes for the GPR programming calls. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

CONSTANTS

MNEMONIC	Value	Explanation
GPR_$BACKGROUND	-2	pixel value for window background
GPR_$BLACK	0	color value for black
GPR_$BLUE	16#0000FF	color value for blue
GPR_$BMF_MAJOR_VERSION	1	major identifier for a bitmap file
GPR_$BMF_MINOR_VERSION	1	minor identifier for a bitmap file
GPR_$CYAN	16#00FFFF	color value for cyan (blue + green)
GPR_$DEFAULT_LIST_SIZE	10	
GPR_$GREEN	16#00FF00	color value for green
GPR_$HIGHEST_PLANE	7	max plane number in a bitmap
GPR_$MAGENTA	16#FF00FF	color value for magenta (red + blue)
GPR_$MAX_BMF_GROUP	0	max group in external bitmaps
GPR_$MAX_X_SIZE	8192	max bits in bitmap x dimension
GPR_$MAX_Y_SIZE	8192	max bits in bitmap y dimension
GPR_$NIL_ATTRIBUTE_DESC	0	descriptor of nonexistent attributes
GPR_$NIL_BITMAP_DESC	0	descriptor of a nonexistent bitmap
GPR_$RED	16#FF0000	color value for red
GPR_$STRING_SIZE	256	number of chars in a gpr string
GPR_$TRANSPARENT	-1	pixel value for transparent (no change)
GPR_$WHITE	16#FFFFFF	color value for white
GPR_$YELLOW	16#FFFF00	color value for yellow (red + green)
GPR_$ROP_ZEROS	0	
GPR_$ROP_SRC_AND_DST	1	

GPR_$ROP_SRC_AND_NOT_DST	2
GPR_$ROP_SR	3
GPR_$ROP_NOT_SRC_AND_DST	4
GPR_$ROP_DST	5
GPR_$ROP_SRC_XOR_DST	6
GPR_$ROP_SRC_OR_DST	7
GPR_$ROP_NOT_SRC_AND_NOT_DST	8
GPR_$ROP_SRC_EQUIV_DS	9
GPR_$ROP_NOT_DST	10
GPR_$ROP_SRC_OR_NOT_DST	11
GPR_$ROP_NOT_SRC	12
GPR_$ROP_NOT_SRC_OR_DST	13
GPR_$ROP_NOT_SRC_OR_NOT_DS	14
GPR_$ROP_ONES	15

DATA TYPES

GPR_$ACCESS_MODE_T

A 2-byte integer. The ways to access an external bitmap. One of the following predefined values:

GPR_$CREATE
Create a file on disk.

GPR_$UPDATE
Update a file on disk.

GPR_$WRITE
Write to a file on disk.

GPR_$READONLY
Read a file on disk.

GPR_$ATTRIBUTE_DESC_T

A 4-byte integer. Identifies an attribute block.

GPR_$BITMAP_DESC_T

A 4-byte integer. Identifies a bitmap.

GPR_$BMF_GROUP_HEADER_T

The group header description for an external bitmap. The diagram below illustrates the GPR_$BMF_GROUP_HEADER_T data type:

predefined type	byte: offset		field name
	0:	integer	n_sects
	2:	integer	pixel_size
	4:	integer	allocated_size
	6:	integer	bytes_per_line
	8:	integer	bytes_per_sect
	10:	integer	
	12:	integer	storage_offset
	14:	integer	

Field Description:

n_sects
The number of sections in a group.

pixel_size
The number of bits per pixel in each section of a group.

allocated_size

bytes_per_line
The number of bytes in one row of a bitmap.

bytes_per_sect
The number of bytes_per_line multiplied by the height of the bitmap. This value must be rounded up to a page boundary, or for small bitmaps rounded up to the next largest binary submultiple of a page.

storage_offset
A pointer to the group storage area.

GPR_$BMF_GROUP_HEADER_ARRAY_T

A gpr_$max_bmf_group-element array of gpr_bmf_group_header_t record structures. The diagram below illustrates a single element:

predefined type	byte: offset	field name
integer	0:	n_sects
integer	2:	pixel_size
integer	4:	allocated_size
integer	6:	bytes_per_line
integer	8:	bytes_per_sect
integer	10:	
integer	12:	storage_offset
integer	14:	

Field Description:

n_sects
The number of sections in a group.

pixel_size
The number of bits per pixel in each section of a group.

allocated_size

bytes_per_line
The number of bytes in one row of a bitmap.

bytes_per_sect
The number of bytes_per_line multiplied by the height of the bitmap. This value must be rounded up to a page boundary, or for small bitmaps rounded up to the next largest binary submultiple of a page.

storage_offset
A pointer to the group storage area.

GPR_$COLOR_T

A 4-byte integer. Defines a color.

GPR_$COLOR_VECTOR_T

A 256-element array of 4-byte integers. Stores multiple color values. Arrays of this type are used as input parameters of color values to be inserted into consecutive slots of a color map. They are also used as output parameters to store color values when inquiries are performed on color maps.

GPR_$COORDINATE_ARRAY_T

A 10-element array of 2-byte integers. Specifies several coordinates in a bitmap. Generally, x coordinates are passed in one array and y coordinates are passed in another array.

GPR_$COORDINATE_T

A 2-byte integer. Specfies one coordinate in a bitmap.

GPR_$DIRECTION_T

A 2-byte integer. Specifies the direction of movement from one text character position to another in a bitmap. One of the following predefined values:

GPR_$UP

GPR_$DOWN

GPR_$LEFT

GPR_$RIGHT

GPR_$DISP_CHAR_T

Stores display characteristics. The diagram below illustrates the gpr_$disp_char_t data type:

predefined type	byte: offset		field name
gpr_$controller_type_t	0:	integer	controller_type
gpr_$accelerator_type_t	2:	integer	accelerator_type
	4:	integer	x_window_origin
	6:	integer	y_window_origin
	8:	integer	x_window_size
	10:	integer	y_window_size
	12:	integer	x_visible_size
	14:	integer	y_visible_size
	16:	integer	x_extension_size
	18:	integer	y_extension_size
	20:	integer	x_total_size
	22:	integer	y_total_size
	24:	integer	x_pixels_per_cm
	26:	integer	y_pixels_per_cm

predefined type	byte: offset		field name
	28:	integer	n_planes
	30;	integer	n_buffers
	32:	integer	delta_x_per_buffer
	34:	integer	delta_y_per_buffer
	36:	integer	delta_planes_per_buffer
gpr_$overlap_set_t	38:	integer	mem_overlaps
	40:	integer	x_zoom_max
	42;	integer	y_zoom_min
	44:	integer	video_refresh_rate
	46:	integer	n_primaries
	48:	integer	lut_width_per_primary
gpr_$format_set_t	50:	integer	avail_formats
gpr_$access_set_t	52:	integer	avail_access
	54:	integer	access_address_space

Field Description:

CONTROLLER_TYPE

A 2-byte integer. The type of graphics hardware controller. One of the following predefined values:

GPR_$CTL_NONE
none or not applicable.

GPR_$CTL_MONO_1
DN100/400/420/460

GPR_$CTL_MONO_2
DN300/320/330

GPR_$CTL_COLOR_1
DN600/550/560

GPR_$CTL_COLOR_2
580/590

GPR_$CTL_COLOR_3
DN570

GPR_$CTL_COLOR_4
DN3000 color.

```
For gpr_$no_display mode, gpr_$ctl_none is returned.
Note that code which makes use of these values may not automatically
extend to new node types, since as new controllers are released,
they will be given new values, and this list will be extended.
```

ACCELERATOR_TYPE

A 2-byte integer. The type of graphics hardware processing accelerator for the node. Only one of the following values is returned. One of the following predefined values:

GPR_$ACCEL_NONE
none or not applicable.

NOTE:

```
Code which makes use of these values may not
automatically extend to new node types, since as
new controllers are released, they will be given
new values, and this list will be extended.

For gpr_$no_display mode, gpr_$accel_none is
returned.
```

x_window_origin
X origin of the frame or window in frame and direct mode respectively. For borrow mode and no-display mode the origin is (0,0).

y_window_origin
Y origin of the frame or window in frame and direct mode respectively. For borrow mode and no-display mode the origin is (0,0).

x_window_size
X dimension of the frame or window in frame and direct mode respectively. For borrow mode this is the x dimension of the screen. For no-display mode this is the x dimension of the maximum legal bitmap.

y_window_size
Y dimension of the frame or window in frame and direct mode respectively. For borrow mode this is the x dimension of the screen. For no-display mode this is the y dimension of the maximum legal bitmap.

x_visible_size
X dimension of the visible area of the screen for frame, direct, and borrow modes. For no-display mode this is the x dimension of the maximum legal bitmap size.

y_visible_size
X dimension of the visible area of the screen for frame, direct, and borrow modes. For no-display mode this is the x dimension of the maximum legal bitmap size.

x_extension_size
The maximum x dimension of the bitmap after having been extended by GPR_$SET_BITMAP_DIMENSIONS. For frame, direct and no-display modes, this size is the same as X_VISIBLE_SIZE. For borrow-mode, this size may be bigger if the device has more display memory past the edges of the visible area.

y_extension_size
The maximum y dimension of the bitmap after having been extended by GPR_$SET_BITMAP_DIMENSIONS. For frame, direct and no-display modes, this size is the same as Y_VISIBLE_SIZE. For borrow-mode, this size may be bigger if the device has more display memory past the edges of the visible area.

x_total_size
X dimension of total bitmap memory. In particular, this is the number of addressable pixel positions, in a linear pixel addressing space, between the first pixel of a scan line and the first pixel of the next scan line. This value may be larger than x_extension_size. For no-display mode this value is the x dimension of the mamimum legal bitmap.

y_total_size
Y dimension of total bitmap memory. This value may be larger than y_extension_size. For no-display mode this value is the y dimension of the mamimum legal bitmap.

x_pixels_per_cm
The number of physical pixels per centimeter on the screen in the x dimension. For no-display mode, this value is set to zero.

y_pixels_per_cm
The number of physical pixels per centimeter on the screen in the y dimension. For no-display mode, this value is set to zero.

n_planes
The maximum number of planes of bitmap memory available on the device. For no-display mode, this parameter is the maximum legal bitmap depth.

n_buffers
The number of displayable refresh buffers available on the device, in borrow mode. In frame, direct, and no-display modes, this parameter is set to one.

delta_x_per_buffer
The "distance" in x, in pixel addresses between refresh buffers on a device with more than one buffer, in borrow mode. For frame, direct and no-display modes, and for devices with only one buffer, this parameter is set to zero.

delta_y_per_buffer
The "distance" in y, in pixel addresses between refresh buffers on a device with more than one buffer, in borrow mode. For frame, direct and no-display modes, and for devices with only one buffer, this parameter is set to zero.

delta_planes_per_buffer
This parameter gives the "distance" in pixel depth between refresh buffers on a device with more than one buffer, in borrow mode. Currently no such device capability is supported, but it may be in the future. For frame, direct and no-display modes, and for devices with only one buffer, this parameter is set to zero.

MEM_OVERLAPS

A 2-byte integer. This parameter gives the kinds of overlap situations that can exist between refresh buffer memory that may be used for different purposes in the device. Sometimes a deviced comes with extra refresh buffer memory beyond what is used to hold the screen image. There are several recognized purposes for particular parts of such memory, and sometimes some memory locations may be available for more than one purpose. If so, the program using this memory will have to take care not to use the same memory for two different purposes at the same time. In order to decide whether this is a possibility, the program can inspect this parameter. For frame, direct and no-display modes, this parameter is set to the null set. Any combination of the following predefined values:

GPR_$HDM_WITH_BITM_EXT
Hidden display memory (HDM), used for loaded text fonts and HDM bitmaps, overlaps with the area into which a bitmap can be extended by use of the GPR_$SET_BITMAP _DIMENSIONS call.

GPR_$HDM_WITH_BUFFERS
HDM overlaps with extra displayable refresh buffers.

GPR_$BITM_EXT_WITH_BUFFERS
The bitmap extension area overlaps with displayable refresh buffers.

x_zoom_max
The maximum pixel-replication zoom factor for x on a device in borrow mode. For frame, direct and no-display modes, and for devices which do not support pixel-replication zoom, these parameters are set to 1.

y_zoom_max
The maximum pixel-replication zoom factor for y on a device in borrow mode. For

frmae, direct and no-display modes, and for devices which do not support pixel-replication zoom, these parameters are set to 1.

video_refresh_rate
The refresh rate of the screen in Hertz. For no-display mode, this value is set to zero.

n_primaries
The number of independent primary colors supported by the video for the device. For color devices, this value is three; for monochrome devices it is one. For no-display mode, this value is set to zero.

lut_width_per_primary
The value gives the number of bits of precision available in each column of a video lookup table (color map) for representing the intensity of a primary color in an overall color value. If a primary color can only be on or off, this value is one. If it can have 16 intensities, this value will be four. If it can have 256 intensities, this value will be eight. For no-display mode, this parameter is set to zero.

AVAIL_FORMATS — A 2-byte integer. The set of available interactive or imaging formats available on the device. Any combination of the following predefined values:

GPR_$INTERACTIVE
Interactive format

GPR_$IMAGING_1024X1024X8
8-bit pixel format on a two-board configuration

GPR_$IMAGING_512X512X24
24-bit pixel format on a three-board configuration

AVAIL_ACCESS — A 2-byte integer. This parameter gives the possible legal pixel cell sizes, in bits, which are available to a program making direct read or write access to the refresh buffer. Currently, the only supported pixel cell size is one bit. This means that the refresh buffers can only be accessed by plane. In the future, other pixel cell sizes may be supported. Any combination of the following predefined values:

GPR_$ALLOC_1
One bit per pixel cell

GPR_$ALLOC_2
Two bits per pixel cell

GPR_$ALLOC_4
Four bits per pixel cell

GPR_$ALLOC_8
One byte per pixel cell

GPR_$ALLOC_16
Two bytes per pixel cell

GPR_$ALLOC_32
Four bytes per pixel cell

access_address_space
This parameter gives the amount of address space available for making direct access to the refresh buffer of the device, in units of 1K-byte pages. For example, if the address space is of a size sufficient to cover 1024 scan lines, each of 1024 bits, its extent will be 128K bytes, thus the value of this parameter will be 128.

GPR_$DISPLAY_CONFIG_T

A 2-byte integer. Specifies the hardware configuration. One of the following predefined values:

GPR_$BW_800X1024
A portrait black and white display.

GPR_$BW_1024X800
A landscape black and white display.

GPR_$COLOR_1024X1024X4
A four-plane color display.

GPR_$COLOR_1024X1024X8
An eight-plane color display.

GPR_$COLOR_1024X800X4
An four-plane color display.

GPR_$COLOR_1024X800X8
An eight-plane color display.

GPR_$COLOR_1280X1024X8
Two-board, eight-plane display.

GPR_$COLOR1_1024X800X8
Two-board, eight-plane display.

GPR_$COLOR2_1024X800X4
One-board, four-plane display.

GPR_$DISPLAY_MODE_T

A 2-byte integer. Specifies the mode of operation. One of the following predefined values:

GPR_$BORROW
Uses the entire screen.

GPR_$FRAME
Uses a frame of the Display Manager.

GPR_$NO_DISPLAY
Uses a main-memory bitmap.

GPR_$DIRECT
Uses a display-manager window.

GPR_$BORROW_NC
Uses the entire screen but does not clear the bitmap.

GPR_$EC_KEY_T

A 2-byte integer. GPR_$INPUT_EC is a predefined value.

GPR_$EVENT_T

A 2-byte integer. Specifies the type of input event. One of the following predefined values:

GPR_$KEYSTROKE
When keyboard character is typed.

GPR_$BUTTONS
When you press button on the mouse or bitpad puck.

GPR_$LOCATOR
When you move the mouse or bitpad puck or use the touchpad.

GPR_$ENTERED_WINDOW
When the cursor enters a window in which the GPR bitmap resides. Direct mode is required.

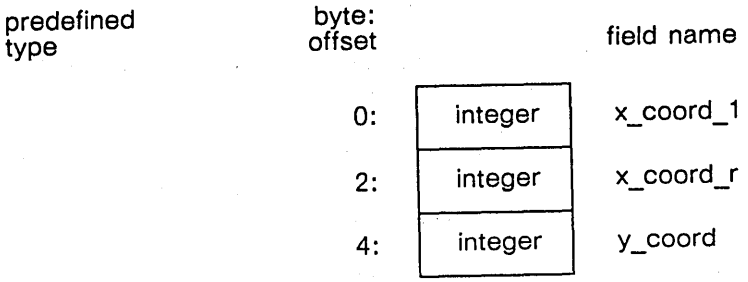
GPR_$LEFT_WINDOW
When the cursor leaves a window in which the GPR bitmap resides. Direct mode is required.

GPR_$LOCATOR_STOP
When you stop moving the mouse or bitpad puck, or stop using the touchpad.

GPR_$NO_EVENT

GPR_$HORIZ_SEG_T

Defines the left- and right-hand x coordinates and the y coordinate of a horizontal line segment. The diagram below illustrates the gpr_$horiz_seg_t data type:



Field Description:

x_coord_l
The left-hand x_coordinate of the line.

x_coord_r
The right-hand x_coordinate of the line.

y_coord
The y coordinate of the line.

GPR_$IMAGING_FORMAT_T

A 2-byte integer. Specifies an imaging or interactive display format. One of the following predefined values:

GPR_$INTERACTIVE
Specifies interactive format.

GPR_$IMAGING_1024X1024X8
Specifies 8-bit imaging format.

GPR_$IMAGING_512X512X24
Specifies 24-bit imaging format.

GPR_$KEYSET_T

An 8-element array of 4-byte integers. Specifies the set of characters that make up a keyset associated with the graphics input event types GPR_$KEYSTROKE and GPR_$BUTTONS. The maximum number of elements in a keyset is 256. Each element of the set is represented by one bit.

GPR_$LINE_PATTERN_T

A 4-element array of 2-byte integers. Specifies the line-pattern to use for line-drawing operation s

GPR_$LINESTYLE_T

A 2-byte integer. Specifies the linestyle for line-drawing operations One of the following predefined values:

GPR_$SOLID
Draw solid lines.

GPR_$DOTTED
Draw dotted lines.

GPR_$MASK_T

A 2-byte integer. Specifies a set of planes to be used in a plane mask.

GPR_$OBSCURED_OPT_T

A 2-byte integer. Specifies the action when a window is obscured. One of the following predefined values:

GPR_$OK_IF_OBS
Acquire the display even though the window is obscured.

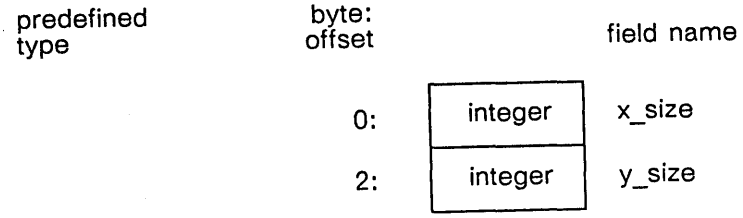
GPR_$ERROR_IF_OBS
Do not acquire the display; return an error message.

GPR_$POP_IF_OBS
Pop the window if it is obscured.

GPR_$BLOCK_IF_OBS
Do not acquire the display until the window is popped.

GPR_$OFFSET_T

Specifies the width and height of a window. The diagram below illustrates the gpr_$offset_t data type:



Field Description:

x_size
The width of the window in pixels.

y_size
The height of the window in pixels.

GPR_$PIXEL_ARRAY_T

A 131073-element array of 4-byte integers. Stores multiple pixel values.

GPR_$PIXEL_VALUE_T

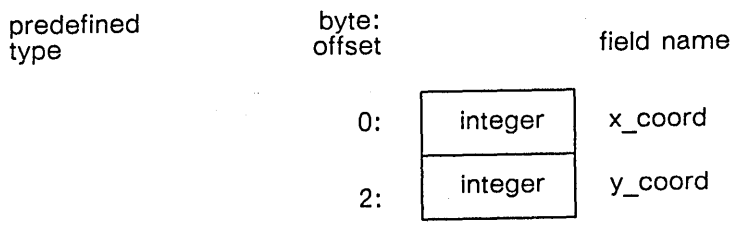
A 4-byte integer. Defines an index into a color map to identify the color of an individual pixel.

GPR_$PLANE_T

A 2-byte integer. Specifies the number of planes in a bitmap.

GPR_$POSITION_T

Specifies the x and y coordinates of a point in a bitmap. The diagram below illustrates the gpr_$position_t data type:



Field Description:

x_coord
The x_coordinate of the point in the bitmap.

y_coord
The y_coordinate of the point in the bitmap.

GPR_$RASTER_OP_ARRAY_T

A 8-element array of 2-byte integers. Stores multiple raster operation opcodes

GPR_$RASTER_OP_T

A 2-byte integer. Specifies raster operation opcodes.

GPR_$RHDM_PR_T

A 4-byte integer. A pointer to a procedure used for refresh-hidden display memory procedures.

GPR_$RWIN_PR_T

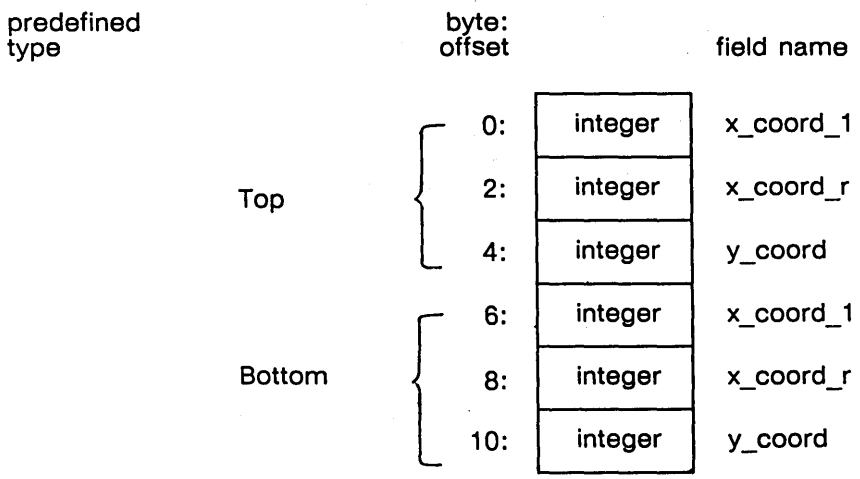
A 4-byte integer. A pointer to a procedure used for refresh-window procedures.

GPR_$STRING_T

An array of up to 256 characters. Stores up to 256 characters.

GPR_$TRAP_LIST_T

A 10-element array of gpr_$trap_t record structures. The diagram below illustrates a single element:



Field Description:

top.x_coord_l
The left-hand x_coordinate of the top line.

top.x_coord_r
The right-hand x_coordinate of the top line.

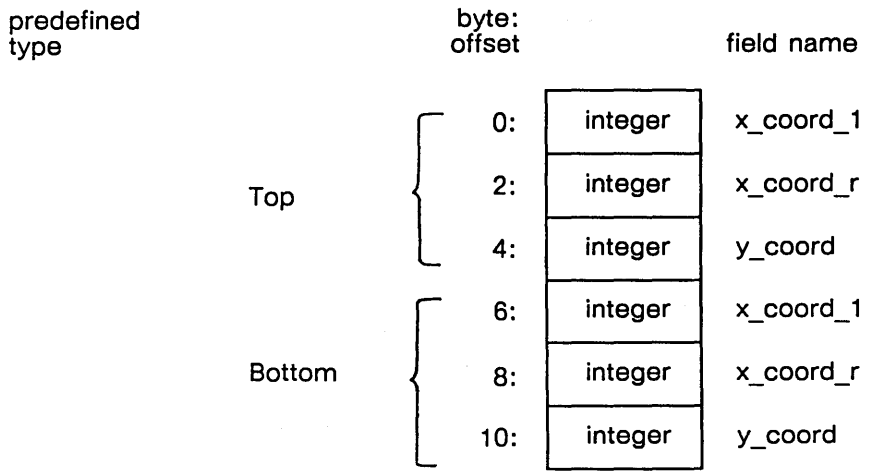
top.y_coord
The y_coordinate of the top line.
bot.x_coord_l
The left-hand x_coordinate of the bottom line.

bot.x_coord_r
The right-hand x_coordinate of the bottom line.

bot.y_coord
The y_coordinate of the bottom line.

GPR_$TRAP_T

Specifies the coordinates of the top and bottom line segments of a trapezoid. The diagram below illustrates the gpr_$trap_t data type:



Field Description:

top.x_coord_l
The left-hand x_coordinate of the top line.

top.x_coord_r
The right-hand x_coordinate of the top line.

top.y_coord
The y_coordinate of the top line.

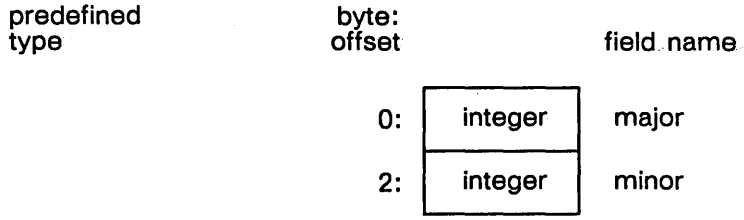
bot.x_coord_l
The left-hand x_coordinate of the bottom line.

bot.x_coord_r
The right-hand x_coordinate of the bottom line.

bot.y_coord
The y_coordinate of the bottom line.

GPR_$VERSION_T

The version number of an external bitmap header. The diagram below illustrates the gpr_$version_t data type:



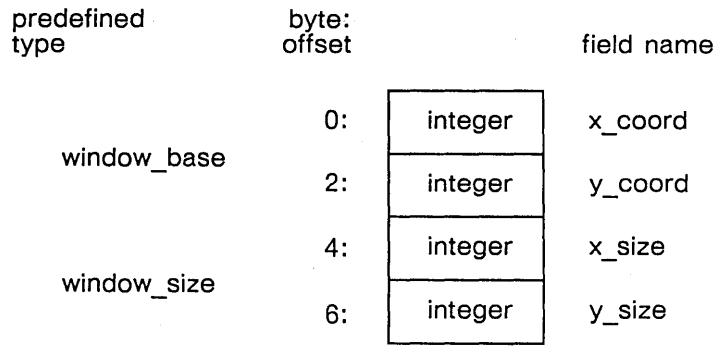
Field Description:

major
The major version number.

minor
The minor version number.

GPR_$WINDOW_LIST_T

A 10-element array of gpr_$window_t record structures. The diagram below illustrates a single element:



Field Description:

window_base.x_coord
The x coordinate of the top left-hand corner of the window.

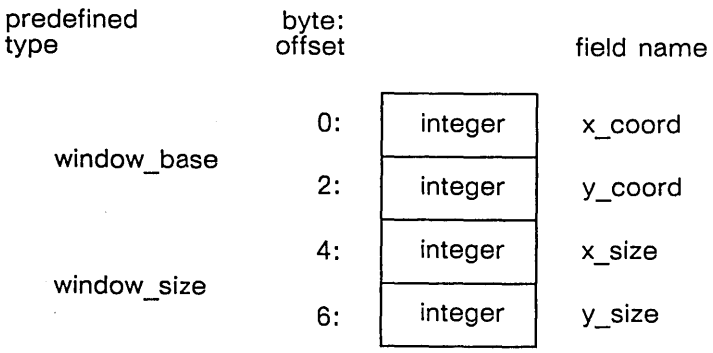
window_base.y_coord
The y coordinate of the top left-hand corner of the window.

window_size.x_size
The width of the widow in pixels.

window_size.y_size
The height of the window in pixels.

GPR_$WINDOW_T

Defines a rectanglar section of a bitmap. X_coord and y_coord specify the coordinates of the top left-hand corner of a rectangle. X_size and y_size specify the width and height of the rectangle. The diagram below illustrates the gpr_$window_t data type:



Field Description:

window_base.x_coord
The x coordinate of the top left-hand corner of the window.

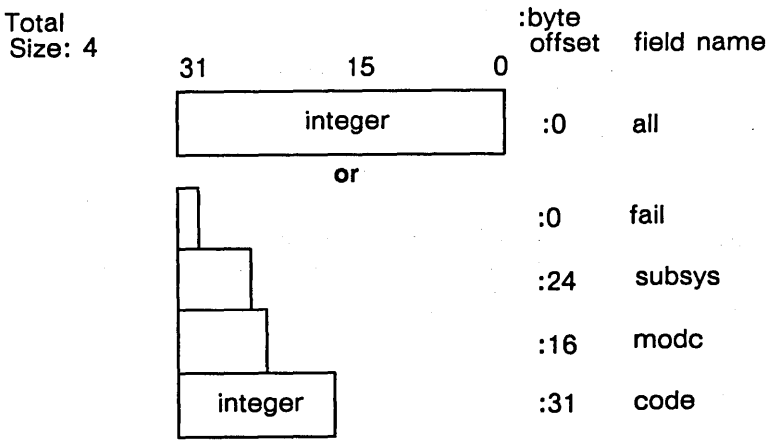
window_base.y_coord
The y coordinate of the top left-hand corner of the window.

window_size.x_size
The width of the widow in pixels.

window_size.y_size
The height of the window in pixels.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

GPR_$ACQUIRE_DISPLAY

Establishes exclusive access to the display hardware and the display driver.

FORMAT

```
unobscured := GPR_$ACQUIRE_DISPLAY (status)
```

RETURN VALUE

unobscured
A Boolean value that indicates whether or not the window is obscured (false = obscured). This parameter is always true unless the option GPR_$OK_IF_OBS was specified to GPR_$SET_OBSCURED_OPT.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

While the display is acquired, the Display Manager cannot run. Hence, it cannot respond to pad calls or to stream calls to input or transcript pads. If you need to call any of these routines, you must release the display to do so.

Since no other display output can occur while the display is acquired, it is not a good idea to acquire the display for long periods of time. The acquire routine automatically times out after a default period of one minute; programs can change this time-out with the routine GPR_$SET_ACQ_TIME_OUT.

Although this call is needed only in direct mode, it can be called from any of the other display modes, where it performs no operation and returns the status code GPR_$NOT_IN_DIRECT_MODE.

If the display is already acquired when this call is made, a count of calls is incremented such that pairs of acquire/release display calls can be nested.

GPR_$ADDITIVE_BLT

Adds a single plane of any bitmap to the current bitmap.

FORMAT

```
GPR_$ADDITIVE_BLT (source_bitmap_desc, source_window, source_plane,
                   dest_origin, status)
```

INPUT PARAMETERS

source_bitmap_desc
Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source_window
Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Types section for more information.

source_plane
The identifier of the source plane to add, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the source bitmap's highest plane.

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Both the source and destination bitmaps can be in either display memory or main memory.

The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

If a rectangle is transferred by a BLT to a display manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$ALLOCATE_ATTRIBUTE_BLOCK

Allocates a data structure that contains a set of default bitmap attribute settings, and returns the descriptor for the data structure.

FORMAT

```
GPR_$ALLOCATE_ATTRIBUTE_BLOCK (attrib_block_desc, status)
```

OUTPUT PARAMETERS

attrib_block_desc
Attribute block descriptor, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To associate an attribute block with the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.

To deallocate an attribute block, use GPR_$DEALLOCATE_ATTRIBUTE_BLOCK.

GPR_$ALLOCATE_BITMAP

Allocates a bitmap in main memory and returns a bitmap descriptor.

FORMAT

```
GPR_$ALLOCATE_BITMAP (size, hi_plane_id, attrib_block_desc, bitmap_desc, status)
```

INPUT PARAMETERS

size

Bitmap width and height, in GPR_$OFFSET_T format. Possible values for width and height are 1 - 8192. This data type is four 4 long. See the GPR Data Types section for more information.

hi_plane_id

Identifier of the highest plane which the bitmap will use, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are 0 - 7.

attrib_block_desc

Descriptor of the attribute block which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap_desc

Descriptor of the allocated bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.

A program can not use a bitmap after it is deallocated.

To establish an allocated bitmap as the current bitmap, use GPR_$SET_BITMAP.

GPR_$ALLOCATE_BITMAP_NC

Allocates a bitmap in main memory without setting all the pixels in the bitmap to zero, and returns a bitmap descriptor.

FORMAT

```
GPR_$ALLOCATE_BITMAP_NC (size,hi_plane_id,attrib_block_desc,bitmap_desc,status)
```

INPUT PARAMETERS

size
Bitmap width and height, in GPR_$OFFSET_T format. This data type is 4 bytes long. The maximum size for a main-memory bitmap is 8192 x 8192. See the GPR Data Types section for more information.

hi_plane_id
Identifier of the highest plane which the bitmap will use, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are 0 - 7.

attrib_block_desc
Descriptor of the attribute block which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap_desc
Descriptor of the allocated bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.

A program can not use a bitmap after it is deallocated.

To establish an allocated bitmap as the current bitmap, use GPR_$SET_BITMAP

GPR_$ALLOCATE_BITMAP sets all pixels in the bitmap to zero; this routine does not. As a result, GPR_$ALLOCATE_BITMAP_NC executes faster, but the initial contents of the bitmap are unpredictable.

GPR_$ALLOCATE_HDM_BITMAP

Allocates a bitmap in hidden display memory.

FORMAT

```
GPR_$ALLOCATE_HDM_BITMAP (size, hi_plane_id, attrib_block_desc, bitmap_desc,
                          status)
```

INPUT PARAMETERS

size
The width and height of the bitmap, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

hi_plane_id
The identifier of the highest plane of the bitmap, in GPR_$PLANE_T format. This is a 2-byte integer.

attrib_block_desc
The descriptor of the bitmap's attribute block, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap_desc
The descriptor of the bitmap in hidden display memeory, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$ALLOCATE_HDM_BITMAP allocates a GPR bitmap in hidden display memory for programs in borrow-display or direct mode. In frame mode, hidden display memory bitmaps cannot be used.

In direct mode you must acquire the display before calling GPR_$ALLOCATE_HDM_BITMAP.

The maximum size allowed for hidden display memory bitmaps is 224 bits by 224 bits.

Use GPR_$DEALLOCATE_BITMAP to deallocate a hidden display bitmap.

GPR_$ARC_3P

Draws an arc from the current position through two other specified points.

FORMAT

```
GPR_$ARC_3P (point_2, point_3, status)
```

INPUT PARAMETERS

point_2
The second point on the arc, in GPR_$POSITION_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

point_3
The third point on the arc, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the arc is drawn.

After the arc is drawn, point_3 becomes the current position.

An error is returned if any of the three points are equal.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$ATTRIBUTE_BLOCK

Returns the descriptor of the attribute block associated with the given bitmap.

FORMAT

```
attrib_block_desc = GPR_$ATTRIBUTE_BLOCK (bitmap_desc, status)
```

RETURN VALUE

attrib_block_desc
Descriptor of the attribute block used for the given bitmap, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

INPUT PARAMETERS

bitmap-desc
Descriptor of the bitmap that is using the requested attribute block, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set an attribute block as the block for the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.

GPR_$BIT_BLT

Performs a bit block transfer from a single plane of any bitmap to a single plane of the current bitmap.

FORMAT

```
GPR_$BIT_BLT (source_bitmap_desc, source_window, source_plane,
              dest_origin, dest_plane, status)
```

INPUT PARAMETERS

source_bitmap_desc
Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source_window
Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Types section for more information.

source_plane
Identifier of the single plane of the source bitmap to move, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the source bitmap's highest plane.

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

dest_plane
Identifier of the plane of the destination bitmap, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the destination bitmap's highest plane.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Both the source and destination bitmaps can be in either display memory or main memory.

The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

If a rectangle is transferred by a BLT to a Display Manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$CIRCLE

Draws a circle with the specified radius around the specified center point.

FORMAT

```
GPR_$CIRCLE(center, radius, status)
```

INPUT PARAMETERS

center
The center of the circle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

radius
The radius of the circle. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The coordinates you specify for the paramenter "center" are added to the corresponding coordinates of the origin for the current bitmap. The resultant coordinate position is the center of the circle.

GPR_$CIRCLE does not change the current position.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$CIRCLE_FILLED

Draws and fills a circle with the specified radius around the specified center point.

FORMAT

```
GPR_$CIRCLE_FILLED (center, radius, status)
```

INPUT PARAMETERS

center
The center of the circle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

radius
The radius of the circle. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The coordinates you specify for the paramenter "center" are added to the corresponding coordinates of the origin for the current bitmap. The resultant coordinate position is the center of the circle.

GPR_$CIRCLE_FILLED does not change the current position.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$CLEAR

Sets all pixels in the current bitmap to the given color/intensity value.

FORMAT

```
GPR_$CLEAR (index, status)
```

INPUT PARAMETERS

index

New color map index specifying a color/intensity value for all pixels in the current bitmap, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

```
0 - 1   for monochromatic displays
0 - 15  for color displays in 4-bit pixel format
0 - 255 for color displays in 8-bit or 24-bit pixel format
-2      for all displays.
```

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A special case occurs if the specified index is -2. A value of -2 specifies clearing the bitmap to the current background color/intensity value. For memory bitmaps and borrowed displays, the background color/intensity index is zero. For Display Manager frames, the background color/intensity value is the same as that used for the window background color.

For monochromatic displays, only the low-order bit of the color value is considered, because bitmaps currently have only one plane. For color displays in 4-bit pixel mode, only the four lowest-order bits of the color value are considered because these displays have four planes.

You can use GPR_$SET_COLOR_MAP to establish the correspondence between color map indexes and color/intensity values. This means that you can use GPR_$SET_COLOR_MAP to assign the pixel value 0 to bright intensity, and then use GPR_$CLEAR either to make the screen bright by passing the pixel value 0, or make the screen dark by passing the value 1. This routine is subject to the restrictions of the current clipping window and plane mask.

GPR_$CLOSE_FILL_PGON

Closes and fills the currently open polygon.

FORMAT

```
GPR_$CLOSE_FILL_PGON (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$CLOSE_FILL_PGON closes and fills the series of polygon boundaries created with the routines GPR_$START_PGON and GPR_$PGON_POLYLINE.

GPR_$CLOSE_FILL_PGON does not use the current raster operation setting.

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

GPR_$CLOSE_RETURN_PGON

Closes the currently open polygon and returns the list of trapezoids within its interior.

FORMAT

```
GPR_$CLOSE_RETURN_PGON (list_size, trapezoid_list, trapezoid_number, status)
```

INPUT PARAMETERS

list_size
The maximum number of trapezoids that the routine is to return. This is a 2-byte integer.

OUTPUT PARAMETERS

trapezoid_list
The trapezoids returned. This is a GPR_$TRAP_LIST_T array of up to 10 elements. See GPR Data Types section for more information.

trapezoid_number
The number of trapezoids that exist within the polygon interior. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$CLOSE_RETURN_PGON returns a list of trapezoids within a polygon interior that the graphics program can draw at a later time with the routine GPR_$MULTITRAPEZOID.

The trapezoid_number parameter is always the total number of trapezoids composing the polygon interior. If this number is greater than the list-size parameter, some trapezoids were left out of the trapezoid_list for lack of space.

GPR_$CLOSE_RETURN_PGON_TRI

Closes the currently open polygon and returns a list of triangles within its interior.

FORMAT

```
GPR_$CLOSE_RETURN_PGON_TRI (list_size, t_list, n_triangles, status)
```

INPUT PARAMETERS

list_size
Maximum number of triangles that the routine is to return.

OUTPUT PARAMETERS

t_list
Triangles returned. This is a GPR_$TRIANGLE_LIST_T array. See the GPR Data Types section for more information.

n_triangles
Number of triangles that exist within the polygon interior. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$CLOSE_RETURN_PGON_TRI returns a list of triangles within a polygon interior that the graphics program can fill at a later time with the routine GPR_$MULTITRIANGLE.

GPR_$CLOSE_RETURN_PGON_TRI returns a list of triangles when a polygon has been defined using GPR_$START_PGON and GPR_$PGON_POLYLINE with the decomposition technique set to gpr_$non_overlapping_tris.

The n_triangles parameter is always the total number of triangles composing the polygon interior. If this number is greater than the list_size parameter, some triangles were left out of the t_list for lack of space.

GPR_$COLOR_ZOOM

Sets the zoom scale factor for a color display.

FORMAT

```
GPR_$COLOR_ZOOM (xfactor, yfactor, status)
```

INPUT PARAMETERS

xfactor
A 2-byte integer that denotes the scale factor for the x-coordinate, in the range 1 through 16.

yfactor
A 2-byte integer that denotes the scale factor for the y-coordinate, in the range 1 through 16.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If the x value is not equal to 1, then the y value must be not equal to 1.

GPR_$COLOR_ZOOM uses the integer zoom feature of the color hardware.

GPR_$COLOR_ZOOM works only in borrow-display mode.

GPR_$COLOR_ZOOM always zooms from the upper-left corner of the display.

GPR_$COLOR_ZOOM returns an error on models DN570/570A and DN3000 if any values other than xfactor = 1, yfactor = 1 are entered.

DN580s allow the xfactor and yfactor to be 2.

GPR_$COND_EVENT_WAIT

Returns information about the occurrence of any event without entering a wait state.

FORMAT

```
unobscured := GPR_$COND_EVENT_WAIT (event_type, event_data, position, status)
```

RETURN VALUE

unobscured
A Boolean value that indicates whether or not the window is obscured; a false value means that the window is obscured. This value is always true unless the program has called GPR_$SET_OBSCURED_OPT and specified an option of GPR_$OK_IF_OBS.

OUTPUT PARAMETERS

event_type
The type of event that occurred, in GPR_$EVENT_T format. This is a 2-byte integer. One of the following values is returned:

GPR_$KEYSTROKE	Input from a keyboard
GPR_$BUTTONS	Input from mouse or bitpad puck buttons
GPR_$LOCATOR	Input from a touchpad or mouse
GPR_$ENTERED_WINDOW	Cursor has entered window
GPR_$LEFT_WINDOW	Cursor has left window
GPR_$LOCATOR_STOP	Input from a locator has stopped
GPR_$NO_EVENT	No event has occurred

event_data
The keystroke or button character associated with the event, or the character that identifies the window associated with an entered window event. This parameter is not modified for other events.

position
The position on the screen or within the window at which graphics input occurred, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

When called, this routine returns immediately and reports information about any event that has occurred. Typically, this routine is called following return from an EC2_$WAIT call involving the eventcount returned by GPR_$GET_EC. The routine allows the program to obtain information about an event without having to suspend all of its activities.

Unless locator data has been processed since the last event was reported, "position" will be the last position given to GPR_$SET_CURSOR_POSITION.

If locator data is received during this call, and GPR_$LOCATOR events are not enabled, the GPR software will display the arrow cursor and will set the keyboard cursor position.

Although this call never waits, it may release the display if it receives an unenabled event that needs to be handled by the Display Manager.

The input routines report button events as ASCII characters. "Down" transitions range from "a" to "d"; "up" transitions range from "A" to "D". The three mouse keys start with (a/A) on the left side. As with keystroke events, button events can be selectively enabled by specifying a button keyset.

GPR_$DEALLOCATE_ATTRIBUTE_BLOCK

Deallocates an attribute block allocated by GPR_$ALLOCATE_ATTRIBUTE_BLOCK.

FORMAT

```
GPR_$DEALLOCATE_ATTRIBUTE_BLOCK (attrib_block_desc, status)
```

INPUT PARAMETERS

attrib_block_desc
The descriptor of the attribute block to deallocate, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To allocate an attribute block, use GPR_$ALLOCATE_ATTRIBUTE_BLOCK.

To associate an attribute block with the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.

GPR_$DEALLOCATE_BITMAP

Deallocates an allocated bitmap.

FORMAT

```
GPR_$DEALLOCATE_BITMAP (bitmap_desc, status)
```

INPUT PARAMETERS

bitmap_desc
Descriptor of the bitmap to deallocate, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To allocate a bitmap, use GPR_$ALLOCATE_BITMAP, GPR_$OPEN_BITMAP_FILE, or GPR_$ALLOCATE_HDM_BITMAP.

GPR_$DISABLE_INPUT

Disables a previously enabled event type.

FORMAT

```
GPR_$DISABLE_INPUT (event_type, status)
```

INPUT PARAMETERS

event_type
The type of event to be disabled, in GPR_$EVENT_T format. This is a 2-byte integer Specify only one of the following events:

GPR_$KEYSTROKE
Input from a keyboard.

GPR_$BUTTONS
Input from mouse or bitpad puck buttons.

GPR_$LOCATOR
Input from a touchpad or mouse.

GPR_$ENTERED_WINDOW
Cursor has entered window.

GPR_$LEFT_WINDOW
Cursor has left window.

GPR_$LOCATOR_STOP
Input from a locator has stopped.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This routine will release and reacquire the display.

Following this call, no events of the given event type will be returned by GPR_$EVENT_WAIT or GPR_$COND_EVENT_WAIT.

In borrow-display mode, disabled events received by the GPR software will be ignored.

In direct mode or frame mode, disabled keystroke or button events are processed by the Display Manager.

When locator events are disabled, the GPR software will display the arrow cursor and will set the keyboard cursor position when locator data is received.

GPR_$DRAW_BOX

Draws an unfilled box based on the coordinates of two opposing corners.

FORMAT

```
GPR_$DRAW_BOX (X1, Y1, X2, Y2, status)
```

INPUT PARAMETERS

X1

The x coordinate of the top left-hand corner of the box. This is a 2-byte integer.

Y1

The y coordinate of the top left-hand corner of the box. This is a 2-byte integer.

X2

The x coordinate of the bottom right-hand corner of the box. This is a 2-byte integer.

Y2

The y coordinate of the bottom right-hand corner of the box. This is a 2-byte integer.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the top left-hand and bottom right-hand corners of the box.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$ENABLE_DIRECT_ACCESS

Ensures completion of display hardware operations before the program uses the pointer to access display memory.

FORMAT

```
GPR_$ENABLE_DIRECT_ACCESS (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If a program uses the GPR_$INQ_BITMAP_POINTER to get the address of display memory for a monochromatic or color display, it should call GPR_$ENABLE_DIRECT_ACCESS after any calls that change the display and before using the pointer returned from the GPR_$INQ_BITMAP_POINTER.

GPR_$ENABLE_INPUT

Enables an event type and a selected set of keys.

FORMAT

```
GPR_$ENABLE_INPUT (event_type, key_set, status)
```

INPUT PARAMETERS

event_type

The type of event to be enabled, in GPR_$EVENT_T format. The types of events are:

```
GPR_$KEYSTROKE          Input from a keyboard
GPR_$BUTTONS            Input from mouse or bitpad puck buttons
GPR_$LOCATOR            Input from a touchpad or mouse
GPR_$ENTERED_WINDOW     Cursor has entered window
GPR_$LEFT_WINDOW        Cursor has left window
GPR_$LOCATOR_STOP       Input from a locator has stopped
```

key_set

The set of specifically enabled characters when the event class is in GPR_$KEYSET_T format. In Pascal, this is a set of characters. In FORTRAN and C this can be implemented as an eight element array of 4-byte integers. This parameter is specified for event types of GPR_$KEYSTROKE and GPR_$BUTTONS. See GPR Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is four bytes long. See the GPR Data Types section for more information.

USAGE

This routine will release and reacquire the display.

This routine specifies the type of event and event input for which GPR_$EVENT_WAIT is to wait.

This routine applies to the current bitmap. However, enabled input events are stored in attribute blocks (not with bitmaps) in much the same way as attributes are. When a program changes attribute blocks for a bitmap during a graphics session, the input events you enabled are lost unless you enable those events for the new attribute block.

Programs must call this routine once for each event type to be enabled.

No event types are enabled by default.

The keyset must correspond to the specified event type. For example, use ['#'..'~'] (in Pascal) to enable all normal printing graphics. Use [chr(0)..chr(127)] to enable the entire ASCII character set. Except in borrow-display mode, it is a good idea to leave at least the CMD and NEXT_WINDOW keys out of the keyset so that the user can access other Display Manager windows.

The insert file /SYS/INS/KBD.INS.PAS contains definitions for the non-ASCII keyboard keys in the range 128 - 255.

Events and keyset data not enabled with this routine will be handled by the Display Manager in frame or direct mode and discarded in borrow-display mode.

When locator events are disabled, the GPR software will display the arrow cursor and will set the keyboard cursor position when locator data is received.

GPR_$EVENT_WAIT

Waits for an event.

FORMAT

```
unobscured := GPR_$EVENT_WAIT (event_type, event_data, position, status)
```

RETURN VALUE

unobscured
A Boolean value that indicates whether or not the window is obscured; a false value means that the window is obscured. This value is always true unless the program has called GPR_$SET_OBSCURED_OPT and specified an option of GPR_$OK_IF_OBS.

OUTPUT PARAMETERS

event_type
The type of event that occurred, in GPR_$EVENT_T format. This is a 2-byte integer. One of the following predefined values is returned:

```
GPR_$KEYSTROKE          Input from a keyboard
GPR_$BUTTONS            Input from mouse or bitpad puck buttons
GPR_$LOCATOR            Input from a touchpad or mouse
GPR_$ENTERED_WINDOW     Cursor has entered window
GPR_$LEFT_WINDOW        Cursor has left window
GPR_$LOCATOR_STOP       Input from a locator has stopped
GPR_$NO_EVENT           No event has occurred
```

event_data
The keystroke or button character associated with the event, or the character that identifies the window associated with an entered window event. This parameter is not modified for other events.

position
The position on the screen or within the window at which graphics input occurred, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This routine suspends process execution until the occurrence of an event type enabled with the GPR_$ENABLE_INPUT. If the event type is keystroke or button, this routine reports only characters in the enabled keyset. Input routines report button events as ASCII characters.

In direct mode, time-out values do not apply to calls to GPR_$EVENT_WAIT; that is, GPR_$EVENT_WAIT waits indefinitely.

The input routines report button events as ASCII characters. "Down" transitions range from "a" to "d"; "up" transitions range from "A" to "D". The three mouse keys start with (a/A) on the left side. As with keystroke events, button events can be selectively enabled by specifying a button keyset.

Unless locator data has been processed since the last event was reported, "position" will be the last position given to GPR_$SET_CURSOR_POSITION.

If locator data is received during this call, and GPR_$LOCATOR events are not enabled, the GPR software will display the arrow cursor and will set the keyboard cursor position.

GPR_$EVENT_WAIT returns an error if the display has not previously been acquired.

This routine will implicitly release the display when the current process is waiting for an event to occur, or when an event that has not been enabled occurs and that event must be handled by the Display Manager.

GPR_$FORCE_RELEASE

Releases the display regardless of how many times it has previously been acquired.

FORMAT

```
GPR_$FORCE_RELEASE (acquire_count, status)
```

OUTPUT PARAMETERS

acquire_count
The number of times the display has been acquired. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This call releases the display regardless of how many times GPR_$ACQUIRE_DISPLAY has been called.

GPR_$GET_EC

Returns the eventcount associated with a graphic event.

FORMAT

```
GPR_$GET_EC (gpr_key, eventcount_pointer, status)
```

INPUT PARAMETERS

gpr_key
The key that specifies which eventcount to obtain, in GPR_$EC_KEY_T format. Currently, this key is always GPR_$INPUT_EC.

OUTPUT PARAMETERS

eventcount_pointer
A pointer to the eventcount for graphics input, in EC2_$PTR_T format.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

USAGE

GPR_$GET_EC returns the eventcount pointer for the graphics input eventcount, which is advanced whenever graphics input may be available.

When this eventcount is advanced, it does not guarantee that GPR_$COND_EVENT_WAIT will return an event, or that GPR_$EVENT_WAIT will not wait. The advance is merely an optimization of a simple polling loop that suspends execution of the process until an event might be available.

GPR_$INIT

Initializes the graphics primitives package.

FORMAT

```
GPR_$INIT (op_mode, unit, size, hi_plane_id, init_bitmap_desc, status)
```

INPUT PARAMETERS

op_mode

One of four modes of operation. Graphics primitives routines can operate in two borrow-display modes, within a Display Manager window, within a frame of a Display Manager pad, or without using the display. Use GPR_$DISPLAY_MODE_T format for this parameter. This is a 2-byte integer. Possible values for this parameter are:

GPR_$BORROW
: Program borrows the full screen and the keyboard from the Display Manager and uses the display driver directly through GPR software.

GPR_$BORROW_NC
: Same as GPR_$BORROW except that all the pixels are not set to zero. (screen is not cleared.)

GPR_$DIRECT
: Program borrows a window from the Display Manager instead of borrowing the whole display.

GPR_$FRAME
: Program executes within a frame of a Display Manager Pad.

GPR_$NO_DISPLAY
: GPR allocates a bitmap in main memory. No graphics is displayed on the screen.

unit

This parameter has three possible meanings, as follows:

1. The display unit, if the graphics routines are to operate in a borrowed display. This is a 2-byte integer. Currently, the only valid display unit number for borrow-display mode is 1.

2. The stream identifier for the pad, if the graphics routines are to operate in frame or direct mode. Use STREAM_$ID_T format. This is a 2-byte integer.

3. Any value, such as zero, if the graphics routines do not use the display.

size

The size of the initial bitmap (and the size of the frame, in frame mode), in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Type section for more information. Possible values are listed below.

	X	Y
Borrow-display or direct mode (limits are reduced to display or window size if necessary):	1 to 1024	1 to 1024
Display Manager Frame:	1 - 32767	1 - 32767
Main Memory Bitmap:	1 - 8192	1 - 8192

hi_plane_id
Identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are:

```
For display memory bitmaps:

0     for monochromatic displays
0 - 3 for color displays in two-board configuration
0 - 7 for color displays in three-board configuration


For main memory bitmaps:

0 - 7 for all displays
```

OUTPUT PARAMETERS

init_bitmap_desc
Descriptor of the initial bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer that uniquely identifies the bitmap.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

USAGE

To use multiple windows, you must call GPR_$INIT for each window.

GPR_$BORROW_NC allows you to allocate a bitmap in display memory without setting all the pixels to zero.

In GPR_$NO_DISPLAY mode, the program can manipulate only main memory bitmaps.

If a program executes in borrow-display mode or direct mode, the size of the initial bitmap can be equal to or smaller than the display. If the program executes in a frame of a Display Manager pad, "size" specifies the size of both the frame and the initial bitmap. (In frame mode, the frame and the bitmap must be the same size.) If the program does not use the display, GPR_$INIT creates a bitmap in main memory. The program specifies the size of this bitmap.

To use imaging formats, a program must be initialized in borrow-display mode.

GPR_$INQ_BITMAP

Returns the descriptor of the current bitmap.

FORMAT

```
GPR_$INQ_BITMAP (bitmap_desc, status)
```

OUTPUT PARAMETERS

bitmap_desc
The descriptor of the current bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

USAGE

To establish a bitmap as the current bitmap, use GPR_$SET_BITMAP.

GPR_$INQ_BITMAP_DIMENSIONS

Returns the size and number of planes of a bitmap.

FORMAT

```
GPR_$INQ_BITMAP_DIMENSIONS (bitmap_desc, size, hi_plane_id, status)
```

INPUT PARAMETERS

bitmap_desc
The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

size
Width and height of the bitmap, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

hi_plane_id
The identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A program can use the information returned by this call to retrieve the actual bitmap size. This could be useful, for example, if the program specified a bitmap size that was too large for the display, causing a reduction in bitmap size.

GPR_$INQ_BITMAP_POINTER

Returns a pointer to bitmap storage in virtual address space. Also returns offset in memory from beginning of one scan line to the next.

FORMAT

```
GPR_$INQ_BITMAP_POINTER (bitmap_desc, storage_ptr, storage_line_width, status)
```

INPUT PARAMETERS

bitmap_desc
Descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

storage_ptr
Start address of bitmap in virtual address space. This is a 4-byte integer.

storage_line_width
Number of 16-bit words in virtual memory between the beginning of one of the bitmap's scan lines and the next. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A program can use the information returned by this call to access individual bits.

Each scan line (horizontal line of a bitmap) starts on a word boundary. The parameter storage_line_width gives the offset in memory from the beginning of one scan line to the beginning of the next, in units of 16-bit words.

When a program uses the parameter storage_ptr to access the screen bitmap on a monochrome system, pixels which are white have the value 1 and pixels that are black have the value 0, regardless of any calls to GPR_$SET_COLOR_MAP. Also, if the cursor is active, the cursor pattern appears in the bitmap.

A program cannot use this routine on a bitmap which is a Display Manager frame.

GPR_$INQ_BITMAP_POSITION

Returns the position of the upper left corner of the specified bitmap. This is normally the screen position; although, it does have some significance for main memory bitmaps.

FORMAT

```
GPR_$INQ_BITMAP_POSITION(bitmap_desc,origin,status);
```

INPUT PARAMETERS

bitmap_desc
The descriptor of the bitmap in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

origin
The position of the upper left-hand corner of the bitmap in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This call is not meaningful if the bitmap is a display manager pad (i.e., a frame mode bitmap).

GPR_$INQ_BM_BIT_OFFSET

Returns the bit offset that corresponds to the left edge of a bitmap in virtual address space.

FORMAT

```
GPR_$INQ_BM_BIT_OFFSET (bitmap_desc, offset, status)
```

INPUT PARAMETERS

bitmap_desc
The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

offset
The number of bits between a 16-bit word boundary and the left edge of the specified bitmap. This is a 2-byte integer in the range 0 - 15.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Each scan line (horizontal line of a bitmap) starts on a word boundary. For all scan lines, this routine returns the number of bits in the most significant part of the first word that are not part of the specified bitmap.

Currently, the offset will be zero for any bitmap other than a direct-mode window.

GPR_$INQ_CHARACTER_WIDTH

Returns the width of the specified character in the specified font.

FORMAT

```
GPR_$INQ_CHARACTER_WIDTH (font_id, character, width, status)
```

INPUT PARAMETERS

font_id
Identifier of the text font. This is a 2-byte integer.

character
The specified character. This is a character variable.

OUTPUT PARAMETERS

width
The width parameter of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a character's width, use GPR_$SET_CHARACTER_WIDTH.

The initial character widths are defined in the font file.

This routine returns the character width in the local copy of the font. Initially, this is a copy of the font file; but the local copy may have been changed. Change in the local copy does not affect the font file or the use of the font by other processes.

GPR_$INQ_COLOR_MAP

Returns the current color map values.

FORMAT

```
GPR_$INQ_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of the first color value entry, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer.

OUTPUT PARAMETERS

values
Color value entries, in GPR_$COLOR_VECTOR_T format. This is a 256-element array of 4-byte integers.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set the color map, use GPR_$SET_COLOR_MAP.

GPR_$INQ_CONFIG

Returns the current display configuration.

FORMAT

```
GPR_$INQ_CONFIG (config, status)
```

OUTPUT PARAMETERS

config

Display configuration, in GPR_$DISPLAY_CONFIG_T format. This is a 2-byte integer. One of the following predefined values is returned:

Returned Value	Display Type
GPR_$BW_800x1024	monochromatic portrait
GPR_$BW_1024x800	monochromatic landscape
GPR_$COLOR_1024x1024x4	color 1024 x 1024 (DN6xx) 2-board config
GPR_$COLOR_1024x1024x8	color 1024 x 1024 (DN6xx) 3-board config
GPR_$COLOR_1024x800x4	color 1024 x 800 (DN5xx) 2-board config
GPR_$COLOR_1024x800x8	color 1024 x 800 (DN5xx) 3-board config
GPR_$COLOR1_1024X800X8	color 1024 x 800 (DN570) 2-board config
GPR_$COLOR_1280X1024X8	color 1280 x 1024 (DN580) 2-board config
GPR_$COLOR2_1024X800X4	color 1024 x 800 (DN3000) 1-board config

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$INQ_CONFIG can be used before GPR_$INIT. This is useful to determine the number of possible planes in bitmaps on color displays before initializing GPR.

GPR_$INQ_CONSTRAINTS

Returns the clipping window and plane mask used for the current bitmap.

FORMAT

```
GPR_$INQ_CONSTRAINTS (window, active, plane_mask, status)
```

OUTPUT PARAMETERS

window
The clipping window, in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Type section for more information.

active
Boolean (logical) value which specifies whether the clip window is enabled. If the value is false, the clip window is disabled; if the value is true, the clip window is enabled.

plane_mask
The plane mask, which specifies the active bitmap plane(s), in GPR_$MASK_T format. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To establish a new clipping window for the current bitmap, use GPR_$SET_CLIP_WINDOW.

To enable the new clipping window, use GPR_$SET_CLIPPING_ACTIVE.

To establish a plane mask, use GPR_$SET_PLANE_MASK.

GPR_$INQ_COORDINATE_ORIGIN

Returns the x- and y-offsets added to all x- and y-coordinates used as input to move, drawing, and BLT operations on the current bitmap.

FORMAT

```
GPR_$INQ_COORDINATE_ORIGIN (origin, status)
```

OUTPUT PARAMETERS

origin
The current coordinate origin for the bitmap, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new coordinate origin, use GPR_$SET_COORDINATE_ORIGIN.

GPR_$INQ_CP

Returns the current position in the current bitmap.

FORMAT

```
GPR_$INQ_CP (x, y, status)
```

OUTPUT PARAMETERS

x

The x-coordinate of the current position, in GPR_$COORDINATE_T format. This is a 2-byte integer.

y

The y-coordinate of the current position, in GPR_$COORDINATE_T format. This is a 2-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$INQ_CP can be used to verify that the current position is at the desired location. If it is not, use GPR_$MOVE to move the current position without drawing a line.

GPR_$INQ_CURSOR

Returns information about the cursor.

FORMAT

```
GPR_$INQ_CURSOR (curs_pat, curs_raster_op, active, position, origin,status)
```

OUTPUT PARAMETERS

cursor_pat
Identifier of the cursor pattern bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

cursor_raster_op
Cursor raster operation code, in GPR_$RASTER_OP_ARRAY_T format. This is an eight-element array of 2-byte integers. The default value is three. (The operation assigns all source values to the new destination).

active
A Boolean (logical) value which indicates whether the cursor is displayed. The parameter is set to true if the cursor is displayed; it is set to false if the cursor is not displayed.

position
The cursor's current position on the screen, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

origin
The pixel currently set as the cursor origin, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Cursor position: If a program calls this routine when in borrow-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a display manager pad, the x- and y-coordinates are relative to the top left corner of the frame.

To alter the cursor, use one of the following:

```
GPR_$SET_CURSOR_PATTERN
GPR_$SET_CURSOR_ACTIVE
GPR_$SET_CURSOR_POSITION
GPR_$SET_CURSOR_ORIGIN
```

Currently, a program can not alter the cursor raster operation.

GPR_$INQ_DECOMP_TECHNIQUE

Returns the mode which controls the algorithm used to decompose and rasterize polygons.

FORMAT

```
GPR_$INQ_DECOMP_TECHNIQUE(decomp_technique,status)
```

OUTPUT PARAMETERS

decomp_technique

Returns a mode which controls the algorithm used to decompose polygons into trapezoids in GPR_$DECOMP_TECHNIQUE_T format. This is a 2-byte integer. Only one of the following predefined values is returned:

GPR_$FAST_TRAPS
: This is the default value on DN3XX/4XXs, DN550/560s, and DN6XXs which indicates that the faster but imprecise algorithm is to be used. This is the only algorithm that existed prior to SR9.

GPR_$PRECISE_TRAPS
: This value indicates that a slower but more precise version of the decomposition algorithm is to be used.

GPR_$NON_OVERLAPPING_TRIS
: This is the default value on DN570/580s and DN3000s which indicates that a triangle decomposition algorithm is to be used.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$INQ_DECOMP_TECHNIQUE returns a mode setting, not an attribute.

GPR_$INQ_DISP_CHARACTERISTICS

Allows the application program to obtain a variety of information about the nature of the actual display device or external bitmap if the program is operating in no-display mode.

FORMAT

```
GPR_$INQ_DISP_CHARACTERISTICS(op,unit_or_pad,disp_len,disp,disp_len_ret,status)
```

INPUT PARAMETERS

op

One of four modes of operation. Graphics primitives routines can operate in two borrow-display modes, within a Display Manager window, within a frame of a Display Manager pad, or without using the display. Use GPR_$DISPLAY_MODE_T format for this parameter. This is a 2-byte integer. Possible values for this parameter are:

GPR_$BORROW
: Returns information about to a borrowed display.

GPR_$BORROW_NC
: Returns information about to a borrowed display.

GPR_$DIRECT Returns information about to a direct-mode window.

GPR_$FRAME Returns information about to a frame of a Display Manager Pad.

GPR_$NO_DISPLAY
: Returns infomation about to a main-memory bitmap.

unit_or_pad

This parameter has three possible meanings, as follows:

1. The display unit, if the graphics routines are to operate in a borrowed display. This is a 2-byte integer. Currently, the only valid display unit number for borrow-display mode is 1.

2. The stream identifier for the pad, if the graphics routines are to operate in frame or direct mode. Use STREAM_$ID_T format. This is a 2-byte integer.

3. For gpr_$no_display this parameter is ignored.

disp_len

Size of the buffer (the DISP parameter described below) provided by the calling program, which will contain the returned display or device information in bytes. For example, if the buffer is ten 16-bit words in length, the program gives 20 as the value of this parameter. No checking is (or can be) done to verify that this length is correct, so unpredictable results are obtained if the program gives a size that is larger than the actual size of the buffer. This parameter allows the calling program to request that less than the full set of characteristics be returned. It also allows the program to continue to function correctly if the list of returned characteristics is extended in the future.

OUTPUT PARAMETERS

disp

Returned display device characteristics in GPR_$DISP_CHAR_T format. This is an array of up 56 bytes. See the GPR data types section for more information.

disp_len_ret

Actual number of bytes of data returned in the "disp" parameter. This is a 2-byte integer. It will always be less than or equal to the "disp_len" input parameter value. Presently, the length of the full set of characteristics is 28 16-bit words, or 56 bytes, so 56 is the current maximum possible value for this parameter.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Prior to SR9.2, programs using GPR could only obtain a value that identified a particular display type, for example, a monochrome display, 1024 by 800 pixels. Programs then derived the particular display characteristics from this value. As a result, a program that wanted to determine display characteristics had to assign a value to each device type that it might want to obtain. Each time we added new display types, user programs had to be modified to identify the new display types.

GPR_$INQ_DISP_CHARACTERISTICS eliminates the need for user programs to include values that identify display device characteristics. This call returns all of a node's display characteristics as a data item in the "disp" parameter. If you use this call, you will not need to extend your programs to support any future display types.

You can call GPR_$INQ_DISP_CHARACTERISTICS at any time, regardless of whether or not GPR has been initialized. If you have initialized GPR, calling this routine has no effect on the current bitmap or its attributes.

When the program calls GPR_$INQ_DISP_CHARACTERISTICS, the values it specifies in the first two parameters are the same as the values it specifies to GPR_$INIT. These parameters identify the display mode and unit or stream to the call, which can then return specific information about the window or bitmap to be used, as well as general information about the display device. The application program must supply a buffer variable, typically of a record type in Pascal, a structure type in C, or an array type in FORTRAN, in which the data can be returned.

In the future, we may extend the list of data items that this call returns as we release new display devices with new characteristics. However, programs written to use the existing set of characteristics will continue to operate correctly.

GPR_$INQ_DRAW_VALUE

Returns the color/intensity value used for drawing lines.

FORMAT

```
GPR_$INQ_DRAW_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel Format
0 - 255	For color displays in 8-bit or 24-bit pixel Format
-1	For all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new draw value, use GPR_$SET_DRAW_VALUE.

GPR_$INQ_FILL_BACKGROUND_VALUE

Returns the color/intensity value of the background used for tile fills.

FORMAT

```
GPR_$INQ_FILL_BACKGROUND_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the tile fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new background value, use GPR_$SET_FILL_BACKGROUND_VALUE.

GPR_$INQ_FILL_PATTERN

Returns the fill pattern for the current bitmap.

FORMAT

```
GPR_$INQ_FILL_PATTERN(pattern, scale, status)
```

OUTPUT PARAMETERS

pattern
The descriptor of the bitmap containing the fill pattern, in GPR_$BITMAP_DESC_T format.

scale
The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern in both the x and y directions. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new fill pattern for the current bitmap, use GPR_$SET_FILL_PATTERN.

Currently, the tile pattern must be stored in a bitmap that is 32 x 32 pixels. The scale factor must be one. Any other pattern size or scale value results in an error.

With a one-plane bitmap as the pattern, the pixel values used are those set by GPR_$SET_FILL_VALUE and GPR_$SET_FILL_BACKGROUND_VALUE. Pixels corresponding to "1" bits of the pattern are drawn in the fill value: pixels corresponding to "0" bits of the pattern are drawn in the fill background value.

GPR_$INQ_FILL_VALUE

Returns the color/intensity value used to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$INQ_FILL_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity fill value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new fill value, use GPR_$SET_FILL_VALUE.

GPR_$INQ_HORIZONTAL_SPACING

Returns the parameter for the width of spacing between displayed characters for the specified font.

FORMAT

```
GPR_$INQ_HORIZONTAL_SPACING (font_id, horizontal_spacing, status)
```

INPUT PARAMETERS

font_id
Identifier of the text font. This is a 2-byte integer.

OUTPUT PARAMETERS

horizontal_spacing
The parameter for horizontal spacing of the specified font. This is a 2-byte integer. Possible values are in the range -127 - 127.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Use GPR_$SET_HORIZONTAL_SPACING to set the width of spacing for a font.

The initial width of horizontal spacing is defined in the font file.

This routine returns the horizontal spacing in the local copy of the font. Initially, this is a copy of the font file; however, the local copy may have been changed. Change in the local copy does not affect the font file or the use of the font by other processes.

GPR_$INQ_IMAGING_FORMAT

Returns the current imaging format.

FORMAT

```
GPR_$INQ_IMAGING_FORMAT (format, status)
```

OUTPUT PARAMETERS

format

Imaging format in GPR_$IMAGING_FORMAT_T configuration. This is a 2-byte integer. If you are using an interactive format, the returned value is GPR_$INTERACTIVE. If you are using the imaging 8-bit pixel format on a two-board configuration, the returned value is GPR_$IMAGING_1024x1024x8. If you are using the imaging 24-bit pixel format, the returned value is GPR_$IMAGING_512x512x24.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

GPR_$INQ_LINESTYLE

Returns information about the current line-style.

FORMAT

```
GPR_$INQ_LINESTYLE (style, scale, status)
```

OUTPUT PARAMETERS

style

The style of line, in GPR_$LINESTYLE_T format. This is a 2-byte integer. One of the following predefined values is returned:

GPR_$SOLID For solid lines

GPR_$DOTTED
For dotted lines.

scale

The scale factor for dashes if the style parameter is GPR_$DOTTED. This is a 2-byte integer.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

When the line-style attribute is GPR_$DOTTED, lines are drawn in dashes. The scale factor determines the number of pixels in each dash and in each space between the dashes.

To set the line-style attribute, use GPR_$SET_LINESTYLE.

GPR_$INQ_LINE_PATTERN

Returns the pattern used in drawing lines.

FORMAT

```
GPR_$INQ_LINE_PATTERN (repeat, pattern, length, status)
```

OUTPUT PARAMETERS

repeat
The replication factor for each bit in the pattern. This is a 2-byte integer.

pattern
The bit pattern, left justified, in GPR_$LINE_PATTERN_T format. This is a four-element array of 2-byte integers.

length
The length of the pattern in bits. This is a 2-byte integer in the range of 0 - 64.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$INQ_LINE_PATTERN returns the current line pattern set explicitly with GPR_$SET_LINE_PATTERN or set implicitly with GPR_$SET_LINESTYLE.

Use GPR_$SET_LINE_PATTERN to specify a new line pattern. You can also use GPR_$SET_LINESTYLE to set a line pattern within the limits of the parameter GRP_$DOTTED.

GPR_$INQ_RASTER_OPS

Returns the raster operations for the current bitmap.

FORMAT

```
GPR_$INQ_RASTER_OPS (raster_op, status)
```

OUTPUT PARAMETERS

raster_op
Raster operation codes, in GPR_$RASTER_OP_ARRAY_T format. This is an eight-element array of 2-byte integers. Each element corresponds to the raster operation for a single plane of the bitmap. Possible raster op values are zero through fifteen.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new raster operation for the current bitmap, use GPR_$SET_RASTER_OP.

GPR_$INQ_REFRESH_ENTRY

Returns two pointers: one to the procedure which refreshes the window; one to the procedure which refreshes hidden display memory.

FORMAT

```
GPR_$SET_REFRESH_ENTRY (window_procedure, disp_mem_procedure, status)
```

OUTPUT PARAMETERS

window_procedure
Entry point for the application-supplied procedure that refreshes the Display Manager window, in GPR_$RWIN_PR_T format. This is a pointer to a procedure.

disp_mem_procedure
Entry point for the application-supplied procedure that refreshes the application's hidden display memory, in GPR_$RHDM_PR_T format. This is a pointer to a procedure.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The returned routines apply to the current bitmap and current attribute block.

Applications can also direct the Display Manager to refresh the window automatically; see the routine GPR_$SET_AUTO_REFRESH.

GPR_$INQ_SPACE_SIZE

Returns the width of the space to be displayed when a character requested is not in the specified font.

FORMAT

```
GPR_$INQ_SPACE_SIZE (font_id, space_size, status)
```

INPUT PARAMETERS

font_id
Identifier of the text font. This is a 2-byte integer.

OUTPUT PARAMETERS

space_size
The space size of the specified font. This is a 2-byte integer. Possible values are in the range -127 to 127.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a font's space size, use GPR_$SET_SPACE_SIZE.

The initial space size is defined in the font file.

The space size is the number of pixels to skip in the horizontal direction when a character not included in the font is written.

GPR_$INQ_TEXT

Returns the text font and text path used for the current bitmap.

FORMAT

```
GPR_$INQ_TEXT (font_id, direction, status)
```

OUTPUT PARAMETERS

font_id
Identifier of the text font used for the current bitmap. This is a 2-byte integer.

direction
The direction of movement from one text character position to the next in the current bitmap, in GPR_$DIRECTION_T format. This is a 2-byte integer. One of the following predefined values is returned:

```
GPR_$UP,
GPR_$DOWN,
GPR_$LEFT,
GPR_$RIGHT
```

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set a new text font for the current bitmap, use GPR_$SET_TEXT_FONT.

To change the direction of text, use GPR_$SET_TEXT_PATH.

GPR_$INQ_TEXT_EXTENT

Returns the x- and y-offsets a string spans when written by GPR_$TEXT.

FORMAT

```
GPR_$INQ_TEXT_EXTENT (string, string_length, size, status)
```

INPUT PARAMETERS

string
A string, in GPR_$STRING_T format. This is a 256 element chararacter array.

string_length
Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

size
Width and height of the area the written string will occupy, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

When the text path is GPR_$RIGHT or GPR_$LEFT, the width is the x-offset. When the text path is GPR_$UP or GPR_$DOWN, the height is the y-offset.

To change the direction of text, use GPR_$SET_TEXT_PATH.

Figure GPR-1 shows two examples of the extent of text in relation to offsets. For horizontal text, use GPR_$RIGHT with GPR_$SET_TEXT_PATH. For rotated text, use GPR_$UP with GPR_$SET_TEXT_PATH.

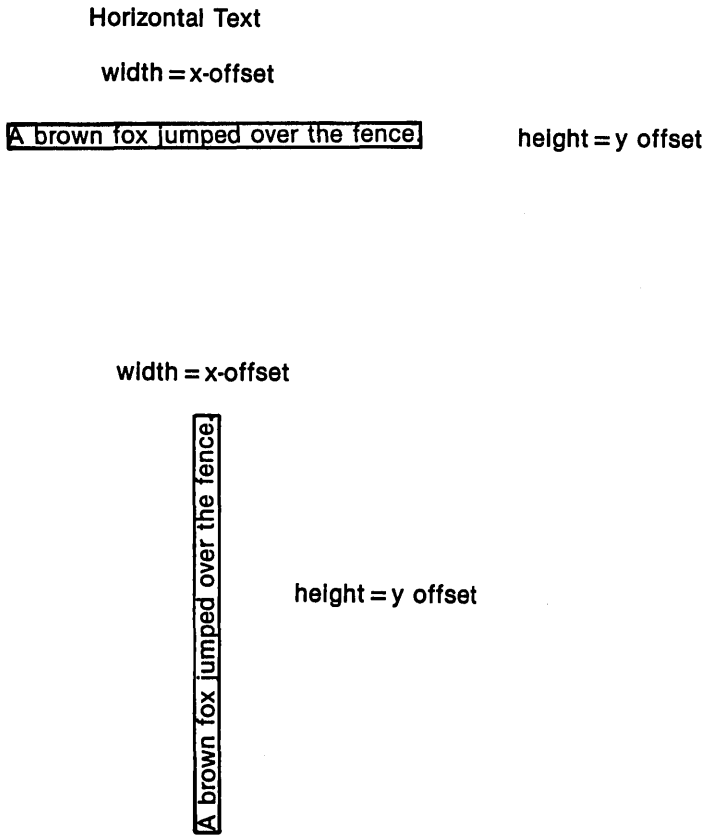


Figure GPR-1. Height and Width for Horizontal and Rotated Text

GPR_$INQ_TEXT_OFFSET

Returns the x- and y-offsets from the top left pixel of a string to to the origin of the string's first character. This routine also returns the x- or y-offset to the pixel which is the new current position after the text is written with GPR_$TEXT.

FORMAT

```
GPR_$INQ_TEXT_OFFSET (string, string_length, start, xy_end, status)
```

INPUT PARAMETERS

string
A string, in GPR_$STRING_T format. This is a 256-element character array.

string_length
Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

start
X- and Y-offsets from the top left pixel of the string to the origin of its first character, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

xy_end
The X- or Y-offset from the top left pixel of the string to the pixel that will be the new current position after the string is written with GPR_$TEXT. This is the X-offset when the text path is specified as GPR_$RIGHT or GPR_$LEFT. This is The Y-offset when the text path is specified as GPR_$UP or GPR_$DOWN. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A program can use the information derived from the "start" output parameter to set the current position to the character origin, rather than the top left corner of the string, before writing the string with GPR_$TEXT.

When the text path is GPR_$RIGHT or GPR_$LEFT, the offset is to the x-axis. When the text path is GPR_$UP or GPR_$DOWN, the offset is to the y-axis.

See GPR_$SET_TEXT_PATH for use of GPR_$RIGHT, GPR_$LEFT, GPR_$UP, and GPR_$DOWN.

Figure GPR-2 shows an example of text offsets, after using GPR_$RIGHT and GPR_$UP with GPR_$SET_TEXT_PATH.

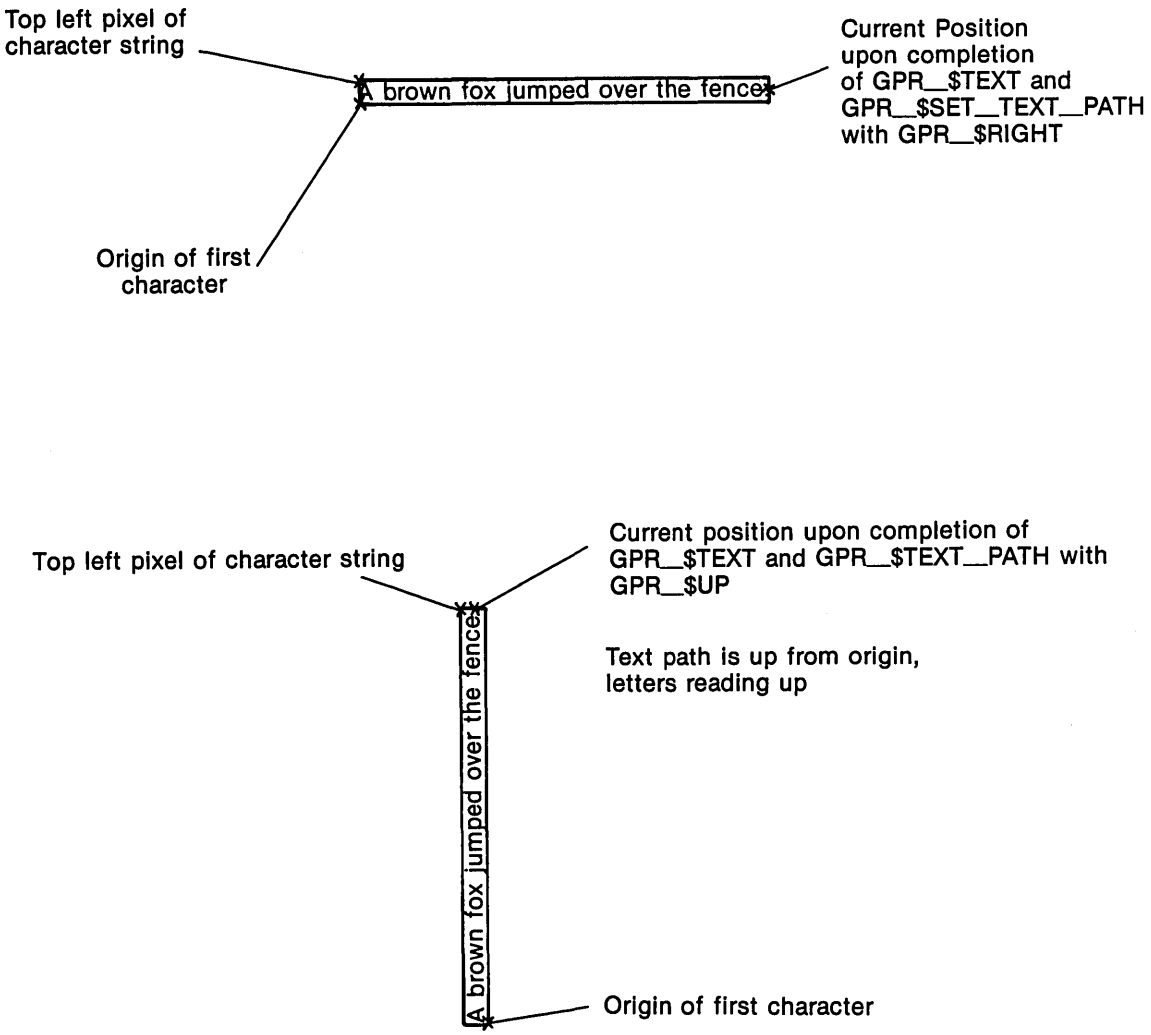


Figure GPR-2. Text Offsets

GPR_$INQ_TEXT_PATH

Returns the direction for writing a line of text.

FORMAT

```
GPR_$INQ_TEXT_PATH (direction, status)
```

OUTPUT PARAMETERS

direction

Direction for writing text, in GPR_$DIRECTION_T format. This is a 2-byte integer. One of the following predefined values is returned: GPR_$UP, GPR_$DOWN, GPR_$LEFT, GPR_$RIGHT

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To set the current text path, use GPR_$SET_TEXT_PATH.

GPR_$INQ_TEXT_VALUES

Returns the text color/intensity value and the text background color/intensity value used in the current bitmap.

FORMAT

```
GPR_$INQ_TEXT_VALUES (text_value, text_bkgd_value, status)
```

OUTPUT PARAMETERS

text_value
A color map index that indicates the text color/intensity value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

text_bkgd_value
A color map index that indicates the text background color/intensity value, in GPR_$PIXEL_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To establish the text color/intensity value, use GPR_$SET_TEXT_VALUE. To establish the text background color/intensity value, use GPR_$SET_TEXT_BACKGROUND_VALUE.

GPR_$INQ_VIS_LIST

Returns a list of the visible sections of an obscured window.

FORMAT

```
GPR_$INQ_VIS_LIST (slots_available, slots_total, vis_list, status)
```

INPUT PARAMETERS

slots_available
Size of the array of visible window sections. This is a 2-byte integer, which is the maximum number of visible rectangles that can be returned. If you want to list all existing sections, you must specify a number that is greater than or equal to the number returned in the slots_total argument (see output parameters).

OUTPUT PARAMETERS

slots_total
Number of existing visible rectangles. This is a 2-byte integer. If this value is greater than the slots_available parameter, then only the number of rectangles specified in slots_available is returned.

vis_list
List of visible window sections. This is an array in GPR_$WINDOW_T format. This data type is eight bytes long. See the GPR Data Types section for more information.

There is no set limit to the number of visible regions that may be returned.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If the display has been acquired but the target window is obscured, programs can call GPR_$INQ_VIS_LIST to locate any visible sections of the window.

If the target window is visible, this routine returns a base of (0,0) and the size of the entire window.

If the window is obscured, the application should call GPR_$SET_CLIP_WINDOW once for each rectangle returned by GPR_$INQ_VIS_LIST before making calls to drawing routines. Clipping is to rectangles only. The GPR software will not perform clipping automatically.

GPR_$INQ_VIS_LIST implicitly releases and reacquires the display in order to communicate with the Display Manager.

GPR_$INQ_WINDOW_ID

Returns the character that identifies the current bitmap's window.

FORMAT

```
GPR_$INQ_WINDOW_ID (character, status)
```

OUTPUT PARAMETERS

character
The character that identifies the current bitmap's window.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This character is returned by GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT when they return GPR_$ENTERED_WINDOW events. The character indicates which window was entered.

The character "A" is the default value of the window identification for all windows.

GPR_$LINE

Draws a line from the current position to the end point supplied. The current position is updated to the end point.

FORMAT

```
GPR_$LINE (x,y, status)
```

INPUT PARAMETERS

x

The x-coordinate, which designates the end point of the line and then becomes the current x-coordinate. Use GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within the bitmap limits, unless clipping is enabled.

y

The y-coordinate, which designates the end point of the line and then becomes the current y-coordinate. Use GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within the bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the line drawn.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

After the line has been drawn, its end point becomes the current position.

To set a new position without drawing a line, use GPR_$MOVE.

GPR_$LOAD_FONT_FILE

Loads a font from a file into the display's font storage area.

FORMAT

```
GPR_$LOAD_FONT_FILE (pathname, pathname_length, font_id, status)
```

INPUT PARAMETERS

pathname
Pathname of the file containing the font, in NAME_$PNAME_T format. This is a character string. Additional information on fonts can be found in the Command Reference manual.

pathname_length
Number of characters in font file pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

font_id
Font identifier. This is a 2-byte integer. Available fonts are listed in the directory /sys/dm/fonts.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Use the font-id returned from this file as input for GPR_$SET_TEXT_FONT.

To unload fonts loaded with this routine, use GPR_$UNLOAD_FONT_FILE.

GPR_$MOVE

Sets the current position to the given position.

FORMAT

```
GPR_$MOVE (x, y, status)
```

INPUT PARAMETERS

x

The x-coordinate, which becomes the current x-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

y

The y-coordinate, which becomes the current y-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The current position is the starting point for many drawing and text operations.

GPR_$MOVE does not draw any lines.

The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the move operation.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$MULTILINE

Draws a series of disconnected lines.

FORMAT

```
GPR_$MULTILINE (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive coordinate positions in GPR_$COORDINATE_ARRAY_T format. This is an array of 2-byte integers. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive coordinate positions in GPR_$COORDINATE_ARRAY_T format. This is an array of 2-byte integers. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$MULTILINE alternately moves to a new position and draws lines: it moves to the first given position, draws a line from the first to the second given position, updates the current position, moves to the third position, etc. After the last line has been drawn or the last move has been made, the endpoint becomes the current position.

The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the multiline drawn.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$MULTITRAPEZOID

Draws and fills a list of trapezoids in the current bitmap.

FORMAT

```
GPR_$MULTITRAPEZOID (trapezoid_list, trapezoid_number, status)
```

INPUT PARAMETERS

trapezoid_list
Trapezoids to fill, in GPR_$TRAP_LIST_T format. This data type is 12 bytes long. See the GPR Data Types section for more information.

trapezoid_number
Number of trapezoids to fill. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$MULTITRAPEZOID fills in a list of trapezoids with the color/intensity value specified with GPR_$SET_FILL_VALUE.

To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

GPR_$MULTITRIANGLE

Draws and fills a list of triangles in the current bitmap.

FORMAT

```
GPR_$MULTITRIANGLE (t_list, n_triangles, status)
```

INPUT PARAMETERS

t_list
Triangles to fill in GPR_$TRIANGLE_LIST_T format. This data type is a variable size array where each element of the array contains 14 bytes. See the GPR Data Types section for more information.

n_triangles
Number of triangles to fill. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This call fills a list of triangles with the color/intensity value specified with GPR_$SET_FILL_VALUE.

To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

When entering coordinates for each triangle, you must set a winding number. The winding number must agree with filling criterion established with GPR_$SET_TRIANGLE_FILL_CRITERIA. For example, if the filling criterion is gpr_$parity, the winding number of triangles to be filled must equal 1. The default filling criterion is gpr_$parity.

Individual triangles can be assigned different winding numbers making it possible to fill specific triangles in the list using GPR_$SET_TRIANGLE_FILL_CRITERIA.

Filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris contain fewer pixels than filled areas rasterized with the decomposition technique set to either gpr_$fast_traps or gpr_$precise_traps.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either gpr_$fast_traps or gpr_$precise_traps overlap.

GPR_$OPEN_BITMAP_FILE

Opens a file for external storage of a bitmap.

FORMAT

```
GPR_$OPEN_BITMAP_FILE (access, filename, name_size, version, size, groups,
                       group_header, attributes, bitmap, created, status)
```

INPUT PARAMETERS

access

One of four ways to access external bitmap objects, in GPR_$ACCESS_MODE_T format. This is a 2-byte integer. Specify only one of the following values:

GPR_$CREATE
: Allocates a new file on disk for storage of a graphic image.

GPR_$UPDATE
: Allows you to modify a previously created file or create a new one.

GPR_$WRITE Allows you to write to an existing file.

GPR_$READONLY
: Allows you to read a previously created file.

filename

The pathname of the bitmap file, in NAME_$PNAME_T format.

name_size

The length of the file name. This is a 2-byte integer.

INPUT/OUTPUT PARAMETERS

version

The version number on the header of the external bitmap file, in GPR_$VERSION_T format. This is a two-element array of two 2-byte integers: a major version number and a minor version number. Currently, version is not used and is always returned as major version 1, minor version 1.

size

Bitmap width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is bitmap width, in raster units; the second element is the bitmap height, in raster units. Possible values for x are 1-4096; possible values for y are 1-4096.

groups

The number of groups in external bitmaps. This is a 2-byte integer. Possible values are 1..(GPR_$MAX_BMF_GROUP +1). Currently, a bitmap can contain only 1 group.

group_header

Description of the external bitmap, in GPR_$BMF_GROUP_HEADER_ARRAY_T format. This is an array [0..GPR_$MAX_BMF_GROUP] of GPR_$BMF_GROUP_HEADER_T. A description of the fields in a group header and the possible values are listed below.

N_SECTS — The number of sections in the group. Currently, there must be 1 section for each plane of a bitmap. N_SECTS is a 2-byte integer which can have a value in the range 1 - 8.

PIXEL_SIZE — The number of bits per pixel in each section of a group. Since each section currently can contain only 1 plane of a bitmap, this value must be 1. PIXEL_SIZE is a 2-byte integer.

ALLOCATED_SIZE
2-byte integer Currently, this value must be 1, but you can specify this value as 0 and GPR will perform the necessary calculations.

BYTES_PER_LINE
The number of bytes in one row of one plane of the bitmap. BYTES_PER_LINE is a 2-byte integer. The value must be a multiple of 4, but can be specified as 0 and GPR will perform the necessary calculations.

BYTES_PER_SECT
The number of BYTES_PER_LINE multiplied by the height of the bitmap. This value must then be either rounded up to a page boundary, or for small bitmaps rounded up to the next largest binary submultiple of a page, for example, one-half, one-fourth, or one-eighth. One page equals 1024 bytes. BYTES_PER_SECT is a 4-byte integer. This value can be specified as 0 and GPR will perform the necessary calculations.

STORAGE_OFFSET
UNIV_PTR format

INPUT PARAMETERS

attribs

The attributes which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap

Descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

created

Boolean (logical) value which specifies whether the bitmap file was created. If the value is true, the file was created.

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Currently, a section is equivalent to one plane of a bitmap. N_SECTS may include up to eight bit planes.

For ALLOCATED_SIZE, BYTES_PER_LINE and BYTES_PER_SECT, you can specify values as 0, and the GPR package will calculate and return the appropriate values.

BYTES_PER_SECT is not necessarily a multiple of BYTES_PER_LINE. This means that GPR will leave unused space at the end of one section to satisfy alignment constraints. The result is that the next section starts on an alignment boundary, which is normally a page boundary.

The access parameter specifies one of four ways to use external bitmaps. As shown in the table below, the value given for this parameter determines whether four other parameters are input (IN) or output (OUT). The values for these parameters are used to validate your input with GPR_$CREATE and GPR_$UPDATE.

	GPR_$CREATE	GPR_$UPDATE file exists no	GPR_$UPDATE file exists yes	GPR_$WRITE	GPR_$READONLY
version, size, groups, group-headers	IN	IN	OUT	OUT	OUT

GPR_$CREATE indicates that you want a new external bitmap file. GPR_$UPDATE means that you want to create a new file or overwrite an existing one.

When you specify GPR_CREATE as the access parameter and you specify a file name that already exists, the file is superseded only if it is a bitmap file. If the file is not a bitmap file, you get the error message NAME_$ALREADY_EXISTS.

Attributes are not stored with the bitmap. You assign attributes when you open the bitmap file. See the routines GPR_$ALLOCATE_ATTRIBUTE_BLOCK and GPR_$ALLOCATE_BITMAP.

Figure GPR-3 is a global view of one group.

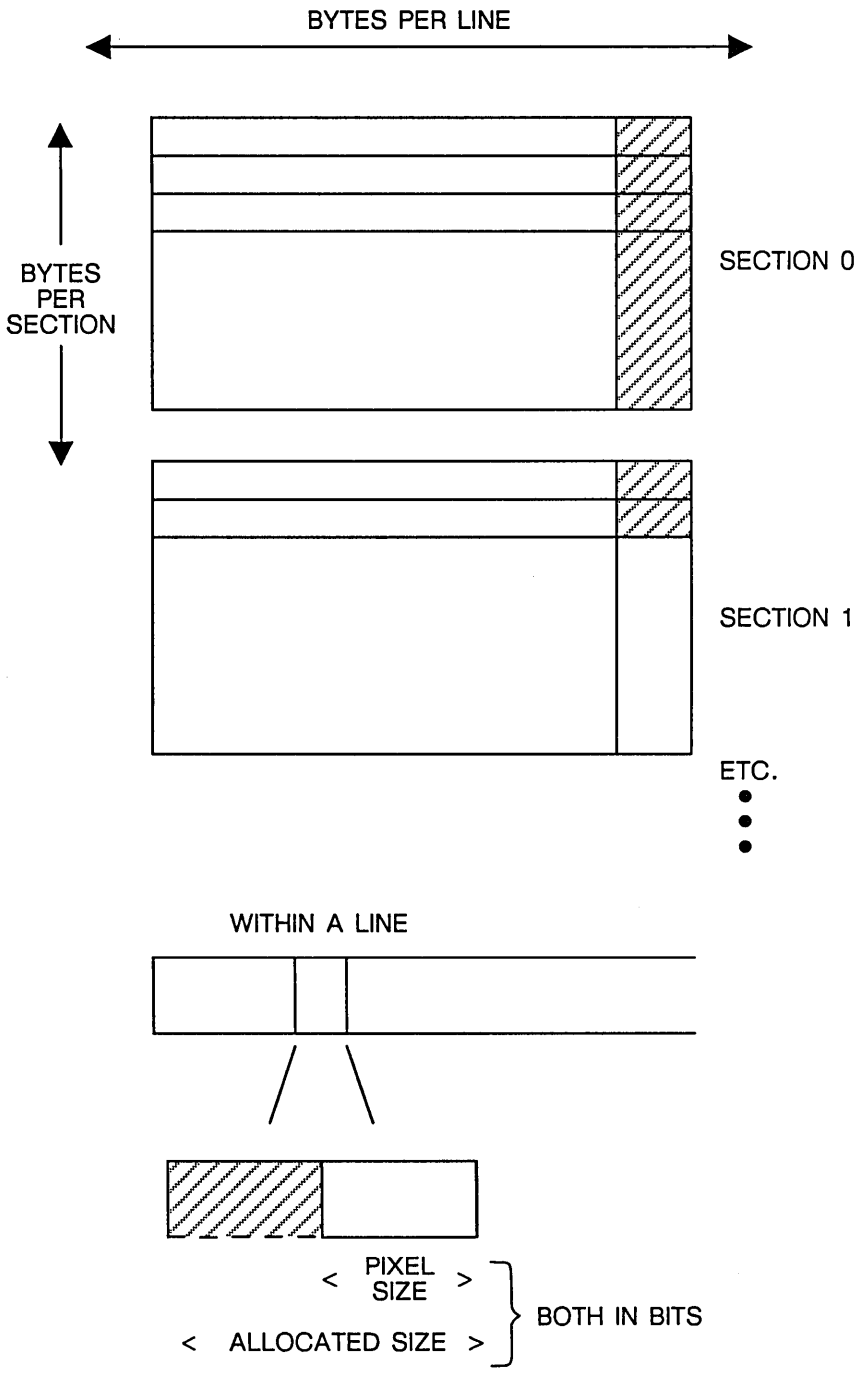


Figure GPR-3. View of One Group

GPR_$PGON_DECOMP_TECHNIQUE

Sets a mode which controls the algorithm used to decompose polygons into trapezoids.

FORMAT

```
GPR_$PGON_DECOMP_TECHNIQUE(decomp_technique,status)
```

INPUT PARAMETERS

decomp_technique

Sets a mode that controls the algorithm used to decompose polygons into trapezoids in GPR_$DECOMP_TECHNIQUE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GPR_$FAST_TRAPS

This is the default value on DN3XX, DN4XX, DN550/560, DN600/660 which indicates that the faster but imprecise algorithm is to be used. This is the only algorithm that existed prior to SR9.

GPR_$PRECISE_TRAPS

This value indicates that a slower but more precise version of the trapezoid decomposition algorithm is to be used.

GPR_$NON_OVERLAPPING_TRIS

This is the default value on the following models: DN570/570A/580 and DN3000.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$PGON_DECOMP_TECHNIQUE establishes a mode setting, not an attribute. Setting the decomposition technique applies to all polygons drawn during a particular session of GPR (within a GPR_$INIT and GPR_$TERMINATE), not just the polygons drawn in the current bitmap.

Polygons without self-crossing and "normal" self-crossing polygons work with the GPR_$FAST_TRAPS setting. Polygons with multiple self-crossings and/or vertices in close proximity may not be filled correctly with the GPR_$FAST_TRAPS setting. Fill these polygons using the GPR_$PRECISE_TRAPS setting.

GPR_$PGON_POLYLINE

Defines a series of line segments forming part of a polygon boundary.

FORMAT

```
GPR_$PGON_POLYLINE (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions. GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive positions. GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$PGON_POLYLINE defines a series of line segments that comprise part of a polygon to be filled in by either (1) GPR_$CLOSE_FILL_PGON or by (2) GPR_$CLOSE_RETURN_PGON and GPR_$MULTITRAPEZOID. The lines are not drawn on the screen until the polygon is filled in by either routines (1) or (2) above. To draw a series of connected lines without filling the resulting figure, use GPR_$POLYLINE.

GPR_$PGON_POLYLINE must be called only when the line segments of a polygon are being defined. See the routine GPR_$START_PGON for more information.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$PIXEL_BLT

Performs a pixel block transfer from any bitmap to the current bitmap.

FORMAT

```
GPR_$PIXEL_BLT (source_bitmap_desc, source_window, dest_origin, status)
```

INPUT PARAMETERS

source_bitmap_desc
Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source_window
Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Types section for more information.

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Use GPR_$SET_BITMAP to establish the current bitmap for this routine.

Both the source and destination bitmaps can be in either display memory or main memory.

The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

If a rectangle is transferred by a BLT to a Display Manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$POLYLINE

Draws a series of connected lines: drawing begins at the current position, draws to the first given coordinate position, then sets the current position to the first given position. This is repeated for all given positions.

FORMAT

```
GPR_$POLYLINE (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the polyline drawn.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$READ_PIXELS

Reads the pixel values from a window of the current bitmap and stores the values in a pixel array.

FORMAT

```
GPR_$READ_PIXELS (source_window, pixel_array, status)
```

INPUT PARAMETERS

source_window
Rectangular section of the current bitmap from which to read pixel values (color/intensity), in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

pixel_array
An array of the pixel values (color/intensity) in GPR_$PIXEL_ARRAY_T format. This is a 131,073-element array of 4-byte integers.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The pixel values from the source window of the current bitmap are stored in the pixel array in row-major order, one in each 4-byte integer.

To write pixel values from an array to the current bitmap, use GPR_$WRITE_PIXELS.

A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

A program cannot read pixels values in imaging formats.

GPR_$RECTANGLE

Draws and fills a rectangle.

FORMAT

```
GPR_$RECTANGLE (rectangle, status)
```

INPUT PARAMETERS

rectangle
The rectangle in the current bitmap to be filled in. Rectangle is in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Type section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$RECTANGLE fills in a rectangle with the color/intensity value specified with GPR_$SET_FILL_VALUE. To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

To draw an unfilled rectangle use GPR_$DRAW_BOX or GPR_$POLYLINE.

GPR_$RELEASE_DISPLAY

Decrements a counter associated with the number of times a display has been acquired.

FORMAT

```
GPR_$RELEASE_DISPLAY (status)
```

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$RELEASE_DISPLAY decrements a counter whose value reflects the number of times the display has been acquired. If the counter value reaches zero, the routine releases the display, allowing other processes, including the Display Manager, to use the display.

GPR_$RELEASE_DISPLAY automatically releases the keyboard when it releases the display.

Programs that call GPR_$EVENT_WAIT may not need to call GPR_$RELEASE_DISPLAY, since GPR_$EVENT_WAIT releases the display implicitly whenever the process waits for input.

GPR_$REMAP_COLOR_MEMORY

Defines the plane in color display memory for which a pointer will be returned when using GPR_$INQ_BITMAP_POINTER. This allows a single plane of color display memory to be accessed directly.

FORMAT

```
GPR_$REMAP_COLOR_MEMORY (plane, status)
```

INPUT PARAMETERS

plane

The plane in color display memory in GPR_$PLANE_T. This is a 2-byte integer. A pointer can be returned to the plane using GPR_$INQ_BITMAP_POINTER. Valid values are 0 - 7.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

When accessing color display memory directly (i.e. by dereferencing the pointer returned by GPR_$INQ_BITMAP_POINTER), the program can access only one plane at a time. This is unlike access to multi-plane memory bitmaps, in which the first scan line of a plane immediately follows the last scan line of the previous plane in virtual memory, or access to bitmaps stored in bitmap files where bytes_per_section specifies the address difference between planes. Therefore, a program must use GPR_$REMAP_COLOR_MEMORY to establish which plane of color display memory will be accessible through the "storage_ptr" returned by GPR_$INQ_BITMAP_POINTER.

GPR_$REMAP_COLOR_MEMORY_1

Defines the plane in hidden color display memory for which a pointer is returned when GPR_INQ_BITMAP_POINTER is used. This allows direct access to a single plane of color display memory.

FORMAT

```
GPR_$REMAP_COLOR_MEMORY_1 (plane, status)
```

INPUT PARAMETERS

plane

The plane in hidden color display memory in GPR_$PLANE_T. This is a 2-byte integer. A pointer can be returned to the plane using GPR_$INQ_BITMAP_POINTER. Valid values are 0 - 7.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$REMAP_COLOR_MEMORY_1 allows access to the normally hidden frame 1 of color display memory. GPR_$REMAP_COLOR_MEMORY allows access to frame 0.

GPR_$REMAP_COLOR_MEMORY_1 returns an error on the following machine models: DN570/570A/580 and DN3000.

GPR_$REPLICATE_FONT

Creates and loads a modifiable copy of a font.

FORMAT

```
GPR_$REPLICATE_FONT (font_id, repli_font_id, status)
```

INPUT PARAMETERS

font_id
Identifier of the original text font. This is a 2-byte integer.

OUTPUT PARAMETERS

repl_font_id
Identifier of the copied text font. This is a 2-byte integer.

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

GPR_$SELECT_COLOR_FRAME

Selects whether frame 0 or frame 1 of color display memory is visible.

FORMAT

```
GPR_$SELECT_COLOR_FRAME (frame, status)
```

INPUT PARAMETERS

frame

This is a 2-byte integer. Denotes which frame is to be visible. Possible values are zero or one. Normally, frame 0 is visible.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SELECT_COLOR_FRAME returns an error if any value other than 0 is entered on the following models: DN570/570A/580 and DN3000.

GPR_$SET_ACQ_TIME_OUT

Establishes the length of time the display will be acquired.

FORMAT

```
GPR_$SET_ACQ_TIME_OUT (timeout, status)
```

INPUT PARAMETERS

timeout
The maximum real time, in TIME_$CLOCK_T format, for which the program can acquire the display.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If the program has not released the display when the time-out expires and another process (for example, the Display Manager) needs the display, an acquire time-out fault (SMD_$ACQUIRE_TIMEOUT) is generated in the user process. The acquire time-out fault is a warning fault that the program can intercept with a cleanup handler or static fault handler. If the program does not release the display within a few seconds of the acquire timeout fault, a second fault occurs (with the status code FAULT_$QUIT) and the program loses control of the display.

If this routine is not called, the default time-out value is one minute.

GPR_$SET_ATTRIBUTE_BLOCK

Associates an attribute block with the current bitmap.

FORMAT

```
GPR_$SET_ATTRIBUTE_BLOCK (attrib_block_desc, status)
```

INPUT PARAMETERS

attrib_block_desc
Descriptor of the attribute block, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To allocate and deallocate attribute blocks, use GPR_$ALLOCATE_ATTRIBUTE_BLOCK and GPR_$DEALLOCATE_ATTRIBUTE_BLOCK.

To request the descriptor of the current bitmap's attribute block, use GPR_$ATTRIBUTE_BLOCK.

This routine may release and reacquire the display if the events enabled in the current and new attribute blocks are different.

GPR_$SET_AUTO_REFRESH

Directs the Display Manager to refresh the window automatically.

FORMAT

```
GPR_$SET_AUTO_REFRESH (auto_refresh, status)
```

INPUT PARAMETERS

auto_refresh
A Boolean value that indicates whether or not the Display Manager will automatically refresh the application's window. A value of true means that auto-refresh is enabled; a value of false (the default) means that auto-refresh is disabled.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Automatic refresh of windows can affect system performance and reduce the amount of disk space available, especially if the application's windows are large.

As an alternative, the application program can also provide procedures that refresh the screen and hidden display. See the routine GPR_$SET_REFRESH_ENTRY.

GPR_$AUTO_REFRESH implicitly releases and reacquires the display in order to communicate with the Display Manager.

This routine applies to the current bitmap. When a program changes attribute blocks for a bitmap during a graphics session, the auto refresh flag is lost unless you set it for the new attribute block.

GPR_$SET_BITMAP

Establishes a bitmap as the current bitmap for subsequent operations.

FORMAT

```
GPR_$SET_BITMAP (bitmap_desc, status)
```

INPUT PARAMETERS

bitmap_desc
A unique bitmap descriptor, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The program can obtain the bitmap descriptor by using GPR_$INQ_BITMAP.

After a bitmap is established using GPR_$SET_BITMAP or GPR_$INIT, it is called the "current bitmap."

GPR_$SET_BITMAP_DIMENSIONS

Modifies the size and the number of planes of a bitmap.

FORMAT

```
GPR_$SET_BITMAP_DIMENSIONS (bitmap_desc, size, hi_plane_id, status)
```

INPUT PARAMETERS

bitmap_desc
The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

size
New width and height of the bitmap, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

hi_plane_id
The new identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A program can use this call to change the size of a bitmap after the bitmap has been created. This is useful if the program wishes to restrict itself to an upper-left subset of the original bitmap or to use hidden memory on a borrowed display.

In direct mode when you allocate a bitmap, you request a size. You may get a smaller size if the Display Manager window is smaller than the size you requested. These restrictions apply to resizing bitmaps. Any bitmap can be shrunk from its original dimensions in x, y or the highest plane. Once the bitmap has been shrunk, it can grow up to its requested size. The maximum allowed sizes for x, y and the highest plane for the various DOMAIN displays are given in the following table.

	max X	max Y	max high plane
Monochromatic display (either portrait or landscape)	1024	1024	0
Color display--Interactive format			
4-bit pixels	1024	2048	3
8-bit pixels	1024	2048	7

If a program uses hidden display memory, it must be careful not to modify areas that are being used to store fill constants or text fonts. The following areas may be used by these functions on the various DOMAIN displays.

Fill constants:

Both monochromatic displays: 800 <= X <= 1023 and Y = 1023.

Color displays: none.

Stand-alone font:

Monochromatic portrait display: 800 <= X <= 1023 and 0 <= Y <= 39.

Monochromatic landscape display: 800 <= X <= 1023 and 983 <= Y <= 1022.

Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

User text fonts: (only if text fonts are loaded)

Monochromatic portrait display: 800 <= X <= 1023 and 40 <= Y <= 1022, allocated from top to bottom.

Monochromatic landscape display: 0 <= X <= 1023 and 800 <= Y <= 1023, in columns 224 bits wide, allocated top to bottom and left to right.

Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

Note that these areas may move, grow or shrink in future DOMAIN software releases. Therefore, only limited use should be made of hidden display memory in conjunction with text or cursor operations.

GPR_$SET_CHARACTER_WIDTH

Specifies the width of the specified character in the specified font.

FORMAT

```
GPR_$SET_CHARACTER_WIDTH (font_id, character, width, status)
```

INPUT PARAMETERS

font_id
Identifier of the text font. This is a 2-byte integer.

character
The specified character. This is a character variable.

width
The width parameter of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve a character's width, use GPR_$INQ_CHARACTER_WIDTH.

The initial character widths are defined in the font file.

To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

GPR_$SET_CLIPPING_ACTIVE

Enables/disables a clipping window for the current bitmap.

FORMAT

```
GPR_$SET_CLIPPING_ACTIVE (active, status)
```

INPUT PARAMETERS

active
A Boolean (logical) value which specifies whether or not to enable the clipping window. Set this value to true to enable the clipping window; set it to false to disable the clipping window.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To specify a clipping window, use the routine GPR_$SET_CLIP_WINDOW.

Initially, in borrow-display, the clip window is disabled. In direct mode, the clip window is enabled and clipped to the size of the window. Clipping cannot be enabled in a bitmap corresponding to a Display Manager frame.

To inquire whether the clip window is enabled, use GPR_$INQ_CONSTRAINTS.

GPR_$SET_CLIP_WINDOW

Changes the clipping window for the current bitmap.

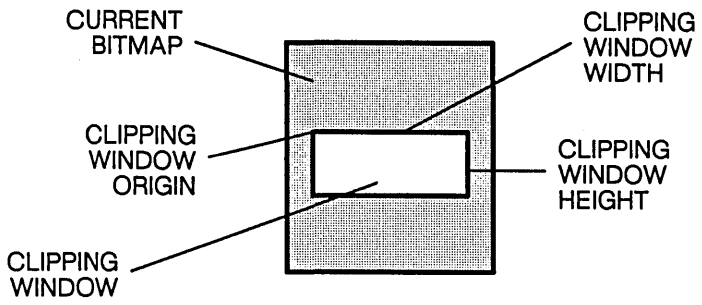
FORMAT

```
GPR_$SET_CLIP_WINDOW (window, status)
```

INPUT PARAMETERS

window

The new clipping window, in GPR_$WINDOW_T format. This data type is 8 bytes long. See GPR Data Types section for more information.



Clipping Window Origin, Width, Height

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The default clip window is the entire bitmap.

Pixels outside the clip window in the current bitmap are not modified by subsequent operations.

To enable the clip window, use GPR_$SET_CLIPPING_ACTIVE.

To request the dimensions of the current clip window, use GPR_$INQ_CONSTRAINTS.

This call is not allowed on the bitmap corresponding to the Display Manager frame.

GPR_$SET_COLOR_MAP

Establishes new values for the color map.

FORMAT

```
GPR_$SET_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-bit pixel format
1 - 256	For color displays in 8-bit or 24-bit pixel format

values
Color value entries, in GPR_$COLOR_VECTOR_T format. This is a 256-element array of 4-byte integers.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

For the monochromatic display, the default start-index is 0, n-entries is 2, and the values are GPR_$BLACK and GPR_$WHITE. Dark has the value GPR_$BLACK, and bright has the value GPR_$WHITE. A program can use this routine to redefine the pixel values corresponding to bright and dark intensity.

For the monochromatic display, if the program provides fewer than two values, or if the first two values are the same (both black or both white), the routine returns an error.

For the monochromatic display, the graphics primitives simulate a color map by modifying the contents of display memory.

In direct mode, you must acquire the display before establishing new values for the color map.

To retrieve the current color map, use GPR_$INQ_COLOR_MAP.

On a monochrome display in direct mode, the color map can not be modified.

GPR_$SET_COLOR_MAP

Establishes new values for the color map.

FORMAT

```
GPR_$SET_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-bit pixel format
1 - 256	For color displays in 8-bit or 24-bit pixel format

values
Color value entries, in GPR_$COLOR_VECTOR_T format. This is a 256-element array of 4-byte integers.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

For the monochromatic display, the default start-index is 0, n-entries is 2, and the values are GPR_$BLACK and GPR_$WHITE. Dark has the value GPR_$BLACK, and bright has the value GPR_$WHITE. A program can use this routine to redefine the pixel values corresponding to bright and dark intensity.

For the monochromatic display, if the program provides fewer than two values, or if the first two values are the same (both black or both white), the routine returns an error.

For the monochromatic display, the graphics primitives simulate a color map by modifying the contents of display memory.

In direct mode, you must acquire the display before establishing new values for the color map.

To retrieve the current color map, use GPR_$INQ_COLOR_MAP.

On a monochrome display in direct mode, the color map can not be modified.

GPR_$SET_COORDINATE_ORIGIN

Establishes x- and y-offsets to add to all x- and y-coordinates used for move, draw, text, fill, and BLT operations on the current bitmap.

FORMAT

```
GPR_$SET_COORDINATE_ORIGIN (origin, status)
```

INPUT PARAMETERS

origin
The new coordinate origin for the bitmap, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current coordinate origin, use GPR_$INQ_COORDINATE_ORIGIN.

The default coordinate origin is (0,0).

This routine may not be used on a bitmap corresponding to a Display Manager frame.

GPR_$SET_CURSOR_ACTIVE

Specifies whether the cursor is displayed.

FORMAT

```
GPR_$SET_CURSOR_ACTIVE (active, status)
```

INPUT PARAMETERS

active
Boolean (logical) value that specifies whether to display the cursor. Set the parameter to true to display the cursor; set it to false if you do not want to display the cursor.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Initially, the cursor is not displayed.

To inquire whether the cursor is currently displayed, use GPR_$INQ_CURSOR.

A program may call this routine only while operating in borrow-display or direct mode.

GPR_$SET_CURSOR_ORIGIN

Defines one of the cursor's pixels as the cursor origin.

FORMAT

```
GPR_$SET_CURSOR_ORIGIN (origin, status)
```

INPUT PARAMETERS

origin

The position of one cursor pixel (the origin) relative to the entire cursor, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

A program uses GPR_$SET_CURSOR_ORIGIN to designate one pixel in the cursor pattern as the cursor origin. Thereafter, when the cursor is moved, the pixel designated as the cursor origin moves to the screen coordinate designated as the cursor position.

The default cursor origin depends on the default cursor size, which depends on the size of the Display Manager's standard font.

To inquire about the current cursor origin, pattern, position and whether the cursor is enabled, use GPR_$INQ_CURSOR.

GPR_$SET_CURSOR_PATTERN

Loads a cursor pattern.

FORMAT

```
GPR_$SET_CURSOR_PATTERN (cursor_pattern, status)
```

INPUT PARAMETERS

cursor_pattern
The descriptor of the bitmap which contains the cursor pattern, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Initially, the cursor pattern is a rectangle, which varies in size according to the size of the Display Manager's standard font. A program can use GPR_$SET_CURSOR_PATTERN to redefine the cursor pattern. The bitmap that represents the cursor pattern consists of one plane, which is a maximum of 16x16 pixels in size.

To inquire about the current cursor pattern, use GPR_$INQ_CURSOR.

GPR_$SET_CURSOR_POSITION

Establishes a position on the screen for display of the cursor.

FORMAT

```
GPR_$SET_CURSOR_POSITION (position, status)
```

INPUT PARAMETERS

position

Screen coordinate position for display of the cursor, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

The first element is the cursor position's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the display in use, as follows:

	X	Y
Borrowed Display:		
Monochromatic Portrait:	0 - 799	0 - 1023
Monochromatic Landscape:	0 - 1023	0 - 799
Color:	0 - 1023	0 - 1023
Color : 550	0 - 1023	0 - 799
Display Manager Frame:	0 - 32767	0 - 32767

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Cursor position: If a program calls this routine when in borrow-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a Display Manager pad, the x- and y-coordinates are offsets from the top left corner of the frame.

If the coordinate position would cause any part of the cursor to be outside the screen or frame, the cursor moves only as far as the edge of the screen. The cursor is neither clipped nor made to disappear.

To request the current cursor position, use GPR_$INQ_CURSOR.

In a Display Manager frame, this routine moves the cursor only if the cursor is in the window viewing this frame when the call is issued. If not, a "next window" command which moves to that window will move the cursor to its new position.

GPR_$SET_DRAW_VALUE

Specifies the color/intensity value to use to draw lines.

FORMAT

```
GPR_$SET_DRAW_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current draw value, use GPR_$INQ_DRAW_VALUE.

The default draw value is 1.

For monochromatic displays, only the low-order bit of the draw value is considered, because monochromatic displays have only one plane.

For color displays in 4-bit pixel format, only the four lowest-order bits of the draw value are considered, because these displays have four planes.

GPR_$SET_FILL_BACKGROUND_VALUE

Specifies the color/intensity value used for drawing the background of tile fills.

FORMAT

```
GPR_$SET_FILL_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the fill background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current background value, use GPR_$INQ_FILL_BACKGROUND_VALUE.

The default fill background value is -2.

This routine defines the background fill value for 1-bit patterns. In all other fill patterns, the values set with this routine are ignored.

GPR_$SET_FILL_PATTERN

Specifies the fill pattern used for the current bitmap.

FORMAT

```
GPR_$SET_FILL_PATTERN (pattern, scale, status)
```

INPUT PARAMETERS

pattern
The descriptor of the bitmap containing the fill pattern, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer. See restriction below.

scale
The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. See restriction below.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Currently, the tile pattern must be stored in a bitmap that is 32x32 pixels by n planes. The scale factor must be one. Any other pattern size or scale value results in an error.

To retrieve the current fill pattern for the current bitmap, use GPR_$INQ_FILL_PATTERN.

With a one-plane bitmap as the pattern, the pixel values used are those set by GPR_$SET_FILL_VALUE and GPR_$SET_FILL_BACKGROUND_VALUE. Pixels corresponding to "1" bits of the pattern are drawn in the fill value: pixels corresponding to "0" bits of the pattern are drawn in the fill background value.

With a multiplane bitmap as the pattern, the pixel values used are those contained in the pattern bitmap.

To re-establish solid fills, set the fill pattern descriptor to GPR_$NIL_BITMAP_DESC.

GPR_$SET_FILL_VALUE

Specifies the color/intensity value to use to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$SET_FILL_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current fill color/intensity value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. The default fill value is 1. Valid values are:

0 - 1 for monochromatic displays 0 - 15 for color displays in 4-bit pixel format 0 - 255 for color displays in 8-bit or 24-bit pixel format

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

For monochromatic displays, only the low-order bit of the fill value is considered, because monochromatic displays have only one plane.

For color displays in 4-bit pixel format, only the four lowest-order bits of the fill value are considered, because these displays have four planes.

"Index" is a color map index, not a color value.

GPR_$SET_HORIZONTAL_SPACING

Specifies the parameter for horizontal spacing of the specified font.

FORMAT

```
GPR_$SET_HORIZONTAL_SPACING (font_id, horizontal_spacing, status)
```

INPUT PARAMETERS

font_id
The identifier of the text font. This is a 2-byte integer.

horizontal_spacing
The horizontal spacing parameter of the specified font. This is a 2-byte integer. Possible values are -127 - 127.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Use GPR_$INQ_HORIZONTAL_SPACING to retrieve a font's horizontal spacing.

The initial horizontal spacing is defined in the font file.

To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

Horizontal spacing is the space between each character in a string.

GPR_$SET_IMAGING_FORMAT

Sets the imaging format of the color display.

FORMAT

```
GPR_$SET_IMAGING_FORMAT (format, status)
```

INPUT PARAMETERS

format

Color format in GPR_$IMAGING_FORMAT_T. This is a two-byte integer. Valid values are:

GPR_$INTERACTIVE
: Either two- or three-board

GPR_$IMAGING_1024x1024x8
: Two-board only

GPR_$IMAGING_512x512x24
: Three-board only

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current imaging format, use GPR_$INQ_IMAGING_FORMAT.

To use GPR_$SET_IMAGING_FORMAT, you must be in borrow display mode and be using a color node.

Imaging formats support only limited GPR operations - displaying pixel data and changing the color map. Other functions return error messages.

1024x1024x8 imaging format is not supported on a three-board system because it offers no advantages over interactive formats.

GPR_$SET_IMAGING_FORMAT accepts only GPR_$INTERACTIVE on the following models: DN570/570A/580 and DN3000.

GPR_$SET_INPUT_SID

Specifies the input pad from which graphics input is to be taken.

FORMAT

```
GPR_$SET_INPUT_SID (stream_id, status)
```

INPUT PARAMETERS

stream_id
The stream-id that GPR software will use for input in frame mode, in STREAM_$ID_T format. The stream must be a Display Manager input pad.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Programs use this call only when they call input routines in frame mode (GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT).

If this routine is not called, the default stream ID is STREAM_$STDIN (a stream id of zero).

To work properly, the input pad must be the pad associated with the transcript pad passed to GPR_$INIT. STREAM_$STDIN is associated with STREAM_$STDOUT in this way in a normal Shell process window. Other process input pads derive their association from the PAD_$CREATE call that created them.

GPR_$SET_LINESTYLE

Sets the line-style attribute of the current bitmap.

FORMAT

```
GPR_$SET_LINESTYLE (style, scale, status)
```

INPUT PARAMETERS

style

The style of line, in GPR_$LINESTYLE_T format. This is a 2-byte integer. Specify only one of the following values:

GPR_$SOLID For solid lines,

GPR_$DOTTED
For dotted lines

scale

The scale factor for dashes if the style parameter is GPR_$DOTTED. This is a 2-byte integer.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

When the line-style attribute is GPR_$DOTTED, lines are drawn in dashes. The scale factor determines the number of pixels in each dash and in each space between the dashes.

For greater flexibility in setting line styles, use GPR_$SET_LINE_PATTERN.

Use GPR_$INQ_LINESTYLE to retrieve the current line-style attribute.

GPR_$SET_LINE_PATTERN

Specifies the pattern to use in drawing lines.

FORMAT

```
GPR_$SET_LINE_PATTERN (repeat_count, pattern, length, status)
```

INPUT PARAMETERS

repeat_count
The replication factor for each bit in the pattern. This is a 2-byte integer. Specifying a value of 0 results in a solid line.

pattern
The bit pattern, left justified, in GPR_$LINE_PATTERN_T format. This is a four-element array of 2-byte integers.

length
The length of the pattern in bits. This is a 2-byte integer in the range of 0 to 64. Specifying a value of 0 results in a solid line.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$LINE, GPR_$POLYLINE, GPR_$MULTILINE use the pattern/style most recently defined by either GPR_$SET_LINE_PATTERN or GPR_$SET_LINESTYLE. The actual bits in the integers define the line pattern. You should set the first bit in the pattern; otherwise, the vectors you draw will not show the beginning of the line correctly.

Specifying the value of 0 for either repeat or length results in a solid line.

You may also set a line pattern with GPR_$SET_LINESTYLE. The pattern is defined by the parameter GPR_$DOTTED.

Within each element of the bit pattern, the bits are used in order of decreasing significance. This starts with the most significant bit of entry 1 down to the least significant of entry 4.

Use GPR_$INQ_LINE_PATTERN to retrieve the current line pattern. This routine returns the pattern set explicitly with GPR_$SET_LINE_PATTERN or set implicitly with GPR_$SET_LINESTYLE.

GPR_$SET_OBSCURED_OPT

Establishes the action to be taken when a window to be acquired is obscured.

FORMAT

```
GPR_$SET_OBSCURED_OPT (if_obscured, status)
```

INPUT PARAMETERS

if_obscured

If the window to be acquired by GPR_$ACQUIRE_DISPLAY is obscured, this argument specifies, in GPR_$OBSCURED_OPT_T format, the action to be taken. This is a 2-byte integer. Specify only one of the following values:

GPR_$POP_IF_OBS
: Pop the window.

GPR_$ERR_IF_OBS
: Return an error and do not acquire the display.

GPR_$BLOCK_IF_OBS
: Block display acquisition until the window is popped.

GPR_$OK_IF_OBS
: Acquire the display even though the window is obscured.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

If this routine is not called, the action to be taken defaults to GPR_$ERR_IF_OBS.

These options apply whenever the display is acquired, either by GPR_$ACQUIRE_DISPLAY or implicitly by GPR_$EVENT_WAIT.

If the program specifies the option GPR_$ERR_IF_OBS, it must check the status code returned from GPR_$ACQUIRE_DISPLAY or GPR_$EVENT_WAIT before calling any drawing routines.

When a program specifies OK_$IF_OBS, the output is performed even when the window is obscured. This output may overwrite other Display Manager windows.

Use GPR_$INQ_VIS_LIST to retrieve a list of visible sections of an obscured window.

GPR_$SET_PLANE_MASK

Establishes a plane mask for subsequent write operations.

FORMAT

```
GPR_$SET_PLANE_MASK (mask, status)
```

INPUT PARAMETERS

mask
The plane mask, which specifies which planes to use, in GPR_$MASK_T format. This is a two-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The default mask specifies that all planes are used.

Operations occur only on the planes specified in the mask. A program can use this routine, for example, to perform raster operations on separate planes or groups of planes in the bitmap.

Using the mask, a program can partition the 8-bit pixels into subunits. For example, the program can use planes 0 - 3 for one picture and planes 4 - 7 for another. Thus, one bitmap may contain two color pictures. This does not, however, increase the number of colors available for one bitmap.

To retrieve the current plane mask, use GPR_$INQ_CONSTRAINTS.

GPR_$SET_RASTER_OP

Specifies a raster operation for both BLTs and lines.

FORMAT

```
GPR_$SET_RASTER_OP (plane_id, raster_op, status)
```

INPUT PARAMETERS

plane_id
Identifier of the bitmap plane involved in the raster operation, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are zero through the identifier of the bitmap's highest plane. See GPR Data Types section for more information.

raster_op
Raster operation code, in GPR_$RASTER_OP_T format. This is a 2-byte integer. Possible values are zero through fifteen.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Use GPR_$INQ_RASTER_OPS to retrieve the current BLT raster operation.

The initial raster operation is 3. This operation assigns all source bit values to new destination bit values.

The following is a list of the op codes and logical functions of the sixteen raster operations and a truth table of the raster operations.

Raster Operations and Their Functions

Op Code	Logical Function
0	Assign zero to all new destination values.
1	Assign source AND destination to new destination.
2	Assign source AND complement of destination to new destination.
3	Assign all source values to new destination.
4	Assign complement of source AND destination to new destination.
5	Assign all destination values to new destination.
6	Assign source EXCLUSIVE OR destination to new destination.
7	Assign source OR destination to new destination.
8	Assign complement of source AND complement of destination to new destination.
9	Assign source EQUIVALENCE destination to new destination.
10	Assign complement of destination to new destination.
11	Assign source OR complement of destination to new destination.
12	Assign complement of source to new destination.
13	Assign complement of source OR destination to new destination.
14	Assign complement of source OR complement of destination to new destination.
15	Assign 1 to all new destination values.

Raster Operations: Truth Table

Source Bit Value	Destination Bit Value	Resultant Bit Values for the following OP Codes: 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

GPR_$SET_REFRESH_ENTRY

Specifies the entry points of application-supplied procedures that refresh the displayed image in a direct window and hidden display memory.

FORMAT

```
GPR_$SET_REFRESH_ENTRY (window_procedure, disp_mem_procedure, status)
```

INPUT PARAMETERS

window_procedure
Entry point for the application-supplied procedure that refreshes the Display Manager window, in GPR_$RWIN_PR_T format. This is a pointer to a procedure.

disp_mem_procedure
Entry point for the application-supplied procedure that refreshes the application's hidden display memory, in GPR_$RHDM_PR_T format. This is a pointer to a procedure.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The Display Manager determines when the window needs to be redrawn based on the amount of activity the user generates on the screen. When a redrawing operation is necessary, the Display Manager calls the application-supplied procedure the next time that the application acquires the display. Two input parameters are passed to the window refresh procedure:

- unobscured -- When false, this Boolean value indicates that the window is obscured.
- position_changed -- When true, this Boolean value indicates that the window has moved or grown since the display was released.

The *Programming With General System Calls* describes the pointer data type.

See *Programming With DOMAIN Graphic Primitives* for an algorithm using procedure pointers.

GPR_$SET_SPACE_SIZE

Specifies the size of horizontal spacing for the specified font.

FORMAT

```
GPR_$SET_SPACE_SIZE (font_id, space_size, status)
```

INPUT PARAMETERS

font_id
Identifier of the text font. This is a 2-byte integer.

space_size
Space size is the number of pixels to skip in the horizontal direction when you include a character that is not in the font. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve a font's space size, use GPR_$INQ_SPACE_SIZE.

The initial character widths are defined in the font file.

To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

The space size is the number of pixels to skip in the horizontal direction when you write a character that is not in the font. Space size is not the size of the space character. To set the size of the space character use GPR_$SET_CHAR_WIDTH.

GPR_$SET_TEXT_BACKGROUND_VALUE

Specifies the color/intensity value to use for text background.

FORMAT

```
GPR_$SET_TEXT_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for the text background, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. This parameter is an index into a color map; it is not a color value. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the text background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the text background. For borrowed displays and memory bitmaps, this value is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current text background value, use GPR_$INQ_VALUES.

The default text background value is -2.

For monochromatic displays, only the low-order bit of the text background value is considered, because monochromatic displays have only one plane.

For color displays in 4-bit pixel mode, only the four lowest-order bits of the text background value are considered, because these displays have four planes.

GPR_$SET_TEXT_FONT

Establishes a new font for subsequent text operations.

FORMAT

```
GPR_$SET_TEXT_FONT (font_id, status)
```

INPUT PARAMETERS

font_id
Identifier of the new text font. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Obtain the font-id when loading a font with GPR_$LOAD_FONT_FILE.

To request the identifier of the current font, use GPR_$INQ_TEXT.

There is no default text font. A program must load and set the font.

Call GPR_$SET_TEXT_FONT for each main memory bitmap. Otherwise, an error is returned (invalid font id).

GPR_$SET_TEXT_PATH

Specifies the direction for writing a line of text.

FORMAT

```
GPR_$SET_TEXT_PATH (direction, status)
```

INPUT PARAMETERS

direction

The direction used for writing text, in GPR_$DIRECTION_T format. This is a 2-byte integer. Specify only one of the following values:

GPR_$UP

GPR_$DOWN

GPR_$LEFT

GPR_$RIGHT

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current text path, use GPR_$INQ_TEXT_PATH.

The initial text path is GPR_$RIGHT.

GPR_$SET_TEXT_VALUE

Specifies the color/intensity value to use for writing text.

FORMAT

```
GPR_$SET_TEXT_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for writing text, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. The valid values are listed below:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To retrieve the current text value, use GPR_$INQ_VALUES.

The default text value is 1 for borrowed displays, memory bitmaps, and Display Manager frames on monochromatic displays; 0 for Display Manager frames on color displays.

For monochromatic displays, only the low-order bit of the text value is considered, because monochromatic displays have only one plane.

For color displays in 4-bit pixel format, only the four lowest-order bits of the text value are considered, because these displays have four planes.

The color specification parameter is a color map index, not a color value.

GPR_$SET_WINDOW_ID

Establishes the character that identifies the current bitmap's window.

FORMAT

```
GPR_$SET_WINDOW_ID (character, status)
```

INPUT PARAMETERS

character
The character that identifies the current bitmaps's window. This is a character variable.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This character is returned by GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT when they return GPR_$ENTERED_WINDOW events. The character indicates which window was entered.

The character 'A' is the default value of the window identification for all windows.

You may assign the same character to more than one window. However, if you do so, you cannot distinguish input from the two windows.

GPR_$SPLINE_CUBIC_P

Draws a parametric cubic spline through the control points.

FORMAT

```
GPR_$SPLINE_CUBIC_P (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SPLINE_CUBIC_P draws a smooth curve starting from the current position, through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any two consecutive points are equal.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$SPLINE_CUBIC_X

Draws a cubic spline as a function of x through the control points.

FORMAT

```
GPR_$SPLINE_CUBIC_X (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SPLINE_CUBIC_X draws a smooth curve starting from the current position and through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any x-coordinate is less than or equal to a previous x-coordinate. The x-coordinate array must be sorted into increasing order.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$SPLINE_CUBIC_Y

Draws a cubic spline as a function of y through the control points.

FORMAT

```
GPR_$SPLINE_CUBIC_Y (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y

List of the y-coordinates of all the successive positions.
GPR_$COORDINATE_ARRAY_T, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

npositions

Number of coordinate positions. This is a 2-byte integer in the range 1 - 32767.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SPLINE_CUBIC_Y draws a smooth curve starting from the current position and through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any y-coordinate is less than or equal to a previous y-coordinate. The y-coordinate array must be sorted into increasing order.

When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$START_PGON

Defines the starting position of a polygon.

FORMAT

```
GPR_$START_PGON (x, y, status)
```

INPUT PARAMETERS

x

The x-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

y

The y-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$START_PGON defines the first point in a polygon boundary. This routine is used in conjunction with GPR_$PGON_POLYLINE to define a connected series of edges composing one closed loop of a polygon's boundary. To see the polygon, you must fill it using either GPR_$CLOSE_FILL_PGON or GPR_$CLOSE_RETURN_PGON and GPR_$MULTITRAPEZOID.

This routine closes any previously open loop of edges by connecting its last endpoint to its first endpoint with an edge. Then, the routine starts the new loop.

GPR_$TERMINATE

Terminates the graphics primitives package.

FORMAT

```
GPR_$TERMINATE (delete_display, status)
```

INPUT PARAMETERS

delete_display
A Boolean (logical) value which specifies whether to delete the frame of the Display Manager pad. If the program has operated in a Display Manager frame and needs to delete the frame at the end of a graphics session, set this value to true. If the program needs to close, but not delete the frame, set this value to false. If the program has not used a Display Manager frame, the value is ignored.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$TERMINATE deletes the frame regardless of the value of the delete-display argument in the following case. A BLT operation from a memory bitmap has been done to a Display Manager frame since the last time GPR_$CLEAR was called for the frame.

GPR_$TEXT

Writes text to the current bitmap, beginning at the current position.

FORMAT

```
GPR_$TEXT (string, string_length, status)
```

INPUT PARAMETERS

string
The string to write, in GPR_$STRING_T format. This is an array of up to 256 characters.

string_length
Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$TEXT always clips to the edge of the bitmap, regardless of whether clipping is enabled.

GPR_$TEXT writes the characters in the current font which correspond to the ASCII values of the characters in the specified string. If the font does not have a character which corresponds to a character in the string, GPR_$TEXT leaves a space. The size of the space is set by GPR_$SET_SPACE_SIZE.

Text is written at the current position. The origin of the first character of the character string is placed at the current position. Generally, the origin of the character is at the bottom left, excluding descenders of the character.

Upon completion of the GPR_$TEXT routine, the current position is updated to the coordinate position where a next character would be written. This is the case even if the string is partly or completely clipped. However, the current position always remains within the boundaries of the bitmap.

GPR_$TRAPEZOID

Draws and fills a trapezoid.

FORMAT

```
GPR_$TRAPEZOID (trapezoid, status)
```

INPUT PARAMETERS

trapezoid
Trapezoid in GPR_$TRAP_T format. This data type is 12 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$TRAPEZOID fills in a trapezoid with the color/intensity value specified with GPR_$SET_FILL_VALUE or the pattern set by GPR_$SET_FILL_PATTERN. To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

The GPR routines define a trapezoid as a quadrilateral with two horizontally parallel sides.

To draw an unfilled trapezoid use GPR_$POLYLINE.

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

GPR_$TRIANGLE

Draws and fills a triangle.

FORMAT

```
GPR_$TRIANGLE (vertex_1, vertex_2, vertex_3, status)
```

INPUT PARAMETERS

vertex_1
First vertex of the triangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

vertex_2
Second vertex of the triangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

vertex_3
Third vertex of the triangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$TRIANGLE fills in a triangle with the color/intensity value specified with GPR_$SET_FILL_VALUE or the fill pattern set by GPR_$SET_FILL_PATTERN.

To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

Filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris contain fewer pixels than filled areas rasterized with the decomposition technique set to either gpr_$fast_traps or gpr_$precise_traps.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either gpr_$fast_traps or gpr_$precise_traps overlap.

GPR_$UNLOAD_FONT_FILE

Unloads a font that has been loaded by GPR_$LOAD_FONT_FILE.

FORMAT

```
GPR_$UNLOAD_FONT_FILE (font_id, status)
```

INPUT PARAMETERS

font_id
Font identifier. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The font_id is returned when a program loads a file with the routine GPR_$LOAD_FONT_FILE.

GPR_$WAIT_FRAME

Waits for the current frame refresh cycle to end before executing operations that modify the color display.

FORMAT

```
GPR_$WAIT_FRAME (status)
```

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

This routine is for use on color displays only.

Operations that modify the color display include block transfers and drawing and text operations.

This routine is useful primarily for animation. It delays execution of display modifications until the scan beam has completely covered the screen.

A program can also use this routine to synchronize changes to the color map with the beginning of the frame.

GPR_$WRITE_PIXELS

Writes the pixel values from a pixel array into a window of the current bitmap.

FORMAT

```
GPR_$WRITE_PIXELS (pixel_array, destination_window, status)
```

INPUT PARAMETERS

pixel_array
A 131,073-element array of 4-byte integers in GPR_$PIXEL_ARRAY_T format from which to write pixel values (color/intensity).

destination_window
Rectangular section of the current bitmap into which to write the pixel values, in GPR_$WINDOW_T format. This data type is 8 bytes long. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

The pixel values in the pixel array, one in each 4-byte integer, are stored in the destination window of the bitmap in row-major order.

For monochromatic displays, only the low-order bit of each pixel value is significant.

For color displays in 4-bit pixel format, only the four lowest-order bits of each pixel value are considered because the bitmaps have four planes.

GPR_$WRITE_PIXELS overwrites the old contents of the bitmap.

To read pixel values from the current bitmap into an array, use GPR_$READ_PIXELS.

A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

ERRORS

GPR_$ALREADY_INITIALIZED
: Primitives are already initialized.

GPR_$ARRAY_NOT_SORTED
: Array must be in ascending order.

GPR_$BAD_ATTRIBUTE_BLOCK
: The attribute block descriptor is incorrect.

GPR_$BAD_BITMAP
: The bitmap descriptor is incorrect.

GPR_$BAD_FONT_FILE
: Font file is incorrect.

GPR_$BITMAP_IS_READ_ONLY
: Bitmap is read-only.

GPR_$CANT_DEALLOCATE
: You cannot deallocate this bitmap.

GPR_$CANT_MIX_MODES
: You cannot mix display modes, for example, borrow and direct.

GPR_$CHARACTER_NOT_IN_FONT
: Character is not in a font.

GPR_$COORD_OUT_OF_BOUNDS
: Coordinate value is out of bounds.

GPR_$DEST_OUT_OF_BOUNDS
: Destination window origin is out of bitmap bounds.

GPR_$DIMENSION_TOO_BIG
: The bitmap dimension is too big.

GPR_$DIMENSION_TOO_SMALL
: The bitmap dimension is too small.

GPR_$DISPLAY_NOT_ACQ
: Display has not been acquired.

GPR_$DUPLICATE_POINTS
: Duplicate points are illegal.

GPR_$FONT_TABLE_FULL
: Font table is full.

GPR_$FONT_IS_READ_ONLY

GPR_$ILLEGAL_FILL_PATTERN
: Illegal bitmap for a fill pattern.

GPR_$ILLEGAL_FILL_SCALE
: Fill pattern scale must be one.

GPR_$ILLEGAL_FOR_FRAME
: Operation is illegal for DM frame.

GPR_$ILLEGAL_FOR_PIXEL_BITMAP

GPR_$ILLEGAL_PATTERN_LENGTH

GPR_$ILLEGAL_PIXEL_VALUE
Pixel value range is illegal.

GPR_$ILLEGAL_SOFTWARE_VERSION

GPR_$ILLEGAL_TEXT_PATH

GPR_$ILLEGAL_WHEN_IMAGING
Operation is illegal in imaging format.

GPR_$INCORRECT_ALIGNMENT
Bitmap layout specifications do not satisfy GPR alignment constraints.

GPR_$INTERNAL_ERROR
This is an internal error.

GPR_$INVALID_COLOR_MAP
The color map is invalid.

GPR_$INVALID_FONT_ID
Font id is invalid.

GPR_$INVALID_IMAGING_FORMAT
Format is invalid for display hardware.

GPR_$INVALID_PLANE
The plane number is invalid.

GPR_$INVALID_RASTER_OP
The raster operation value is invalid.

GPR_$KBD_NOT_ACQ
Keyboard has not been acquired.

GPR_$MUST_BORROW_DISPLAY
You must borrow the display for this operation.

GPR_$MUST_RELEASE_DISPLAY
You must release the display for this operation.

GPR_$NO_ATTRIBUTES_DEFINED
No attributes are defined for the bitmap.

GPR_$NO_INPUT_ENABLED
No input events are enabled.

GPR_$NO_MORE_SPACE
No more bitmap space is available.

GPR_$NOT_IN_DIRECT_MODE
Display is not in direct mode.

GPR_$NOT_IN_POLYGON
No polygon is being defined.

GPR_$NOT_INITIALIZED
: Primitives are not initialized.

GPR_$SOURCE_OUT_OF_BOUNDS
: Source window origin is out of bitmap bounds.

GPR_$TOO_MANY_INPUT_WINDOWS

GPR_$UNABLE_TO_ROTATE_FONT

GPR_$WINDOW_OBSCURED
: Window is obscured.

GPR_$WINDOW_OUT_OF_BOUNDS
: Window origin is out of bitmap bounds.

GPR_$WRONG_DISPLAY_HARDWARE
: The display hardware is wrong for this operation.