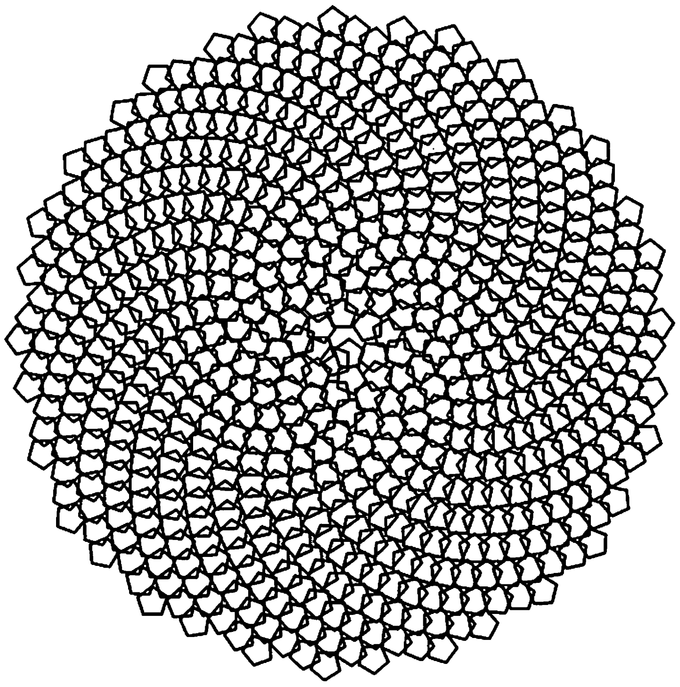
GDDM™

Base Programming Reference Volume 2

SC33-0332-1

IBM



Front Cover Pattern: Electronic Sunflower

The pattern on the front and back cover was produced using this GDDM program.

```
          INTEGER TYPE, VAL, COUNT, N, M
          REAL A1, A2, K1, K2, R1, R2, X, Y
          REAL XCEN, YCEN, XS, YS
          K1=5.3333
          K2=1.1
          R1=2
          XCEN=50
          YCEN=50
          CALL FSINIT
          CALL GSPS(1.0,1.0)
          K2=1.1*SQRT(2.4/K1)
          A2=0
          DO 40 M=1, 600
            A2=A2+K1
            R2=K2*(A2**.5)
            XS=R2*COS(A2)+XCEN
            YS=R2*SIN(A2)+YCEN
            DO 30 N=0, 5
              A1=2.*3.142*(FLOAT(N)/5.)+A2
              X=R1*COS(A1)+XS
              Y=R1*SIN(A1)+YS
              IF (N) 20,10,20
   10         CALL GSMOVE(X,Y)
   20         CALL GSLINE(X,Y)
   30       CONTINUE
   40     CONTINUE
          CALL ASREAD (TYPE,VAL,COUNT)
          CALL FSTERM
          END
```

SC33-0332-1
File No. S370/4300-40

G D D M

Base Programming Reference

GDDM/MVS 5665-356
GDDM/VM 5664-200
GDDM/VSE 5666-328
GDDM/VMXA 5684-007

Version 2 Release 2

Licensed Programs

Volume 2 of 2

IBM

Second Edition (September 1988)

This edition (Volume 2) applies to Version 2 Release 2 of the IBM GDDM™ (Graphical Data Display Manager) Series of licensed programs. The programs and their numbers are:

GDDM/VM	5664-200
GDDM/MVS	5665-356
GDDM/VSE	5666-328
GDDM/VMXA	5684-007.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems or equipment, refer to the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current. If you need to order additional copies of this edition of this book after any further revision has been published by IBM, use the temporary order number SQ33-0332.

Changes and additions to the text and illustrations are indicated by revision bars (vertical lines) to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program can be used; any functionally equivalent program can be used instead.

Publications are not stocked at the addresses given below. Requests for IBM publications should be made to your representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed either to:

International Business Machines Corporation, Department 6R1H, 180 Kost Road,
Mechanicsburg, Pennsylvania 17055, U.S.A.

or to:

IBM United Kingdom Laboratories Limited, Information Development,
Mail Point 95, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

™ Trademark of IBM Corporation.

Preface (Volume 2)

This volume of the *GDDM Base Programming Reference* provides detailed support information for the IBM licensed program GDDM (Graphical Data Display Manager), Version 2.

This Volume is complementary to the introductory information and descriptions of the GDDM Base calls given in the *GDDM Base Programming Reference, Volume 1*.

For more information, see the Preface to *Volume 1*.

Book structure (Volume 2)

Volume 2 (this volume) of the *GDDM Base Programming Reference* contains:

| **Chapters 1 through 14**
These chapters describe the GDDM environment from a programming viewpoint. They give detailed information that may be needed for some programming tasks over and above the descriptions of the GDDM calls that are contained in *GDDM Base Programming Reference, Volume 1*.

| **Appendixes A through L**
These appendixes are reference sources for various aspects of the GDDM programming interface that are not described elsewhere.

Index

Book structure (Volume 1)

Volume 1 of the *GDDM Base Programming Reference* contains:

Chapter 1. Introduction to GDDM
This chapter is an introduction to GDDM, which briefly describes its main features.

Chapter 2. Concepts of GDDM and use of functions
This chapter describes the main concepts of GDDM and their uses.

Chapter 3. GDDM programming interface
This chapter describes conventions for writing GDDM programs.

Chapter 4. GDDM calls
This chapter is a complete list of all GDDM Base calls, with their syntax and function. The calls are listed in alphabetic order.

Glossary
This defines the abbreviations and terminology used in **both** Volumes of the *GDDM Base Programming Reference* manual.

Index

Table of contents (Volume 2)

Chapter 1. Customizing your program and its environment

You can customize various aspects of the GDDM and subsystem environment if you find that the defaults supplied with the product do not suit the needs of your application program exactly. You can modify GDDM to suit the needs of your installation, both hardware and software, the application programs that run in it, and the end users of your GDDM programs.

Before Version 1 Release 4, a GDDM installer or user had to change GDDM defaults by modifying and assembling an Environmental Defaults Module, the specific structure of which changed on each release. Since Version 1 Release 4, GDDM lets you create an External Defaults Module, the structure of which is such that a GDDM installer or user does not have to remodify or reassemble the module on subsequent releases. Also, in some operating environments, you can keep source-format defaults specifications on a locally-accessed file.

By using the information in this chapter and the supporting Chapters 2 through 6, you will be able to modify:

- Defaults that apply to the GDDM environment.

  You can change the defaults provided in GDDM. The term **default** covers a wide range of parameters that you might want to change to support the particular considerations for your installation, or for your program.

  They include, for example, naming conventions for files and data sets, buffer sizes and other performance-related factors, time, date, and number punctuation conventions, the language used in panels and error messages, and so on.

  If you have upgraded from a previous release of GDDM, you can continue to use the GDDM Environmental Defaults Module for your particular subsystem; they are as follows:

CICS/VS	ADMADFC
IMS/VS	ADMADFI
TSO	ADMADFT
VM/CMS	ADMADFV.

  Note that you can use the old method only for defaults that you specified in your earlier release of GDDM; newly supported defaults are available only through the new methods described in this manual.

  For details of the GDDM-supplied defaults that you can change, see Appendix A, "GDDM's default values" on page 127.

- Exits, either for individual users or for the whole installation.

  This allows a system program to trap specific events whenever an application program uses a GDDM or system resource. Such events include task-switching in TSO, intercepting some or all GDDM calls, and so on.

  For details of the GDDM user exits that you can specify, see "Specifying user exits" on page 104.

- Synonyms (called **nicknames**) that remove the need for specifying complex DSOPEN parameter structures.

  Nicknames help you to write application programs that are more device-independent than they might otherwise be. For example, you can write a program that sends a picture to a display screen, and, without having to change the source program or recompile it, you can use the program to send the picture to a file for later printing on a composed-page printer.

  For details of how to specify nicknames, see "Using nicknames to define device characteristics" on page 3.

GDDM provides an integrated method for changing these items. A **user default specification (UDS)** is the means by which you define a specific value, or set of values, for changing GDDM defaults, exits, and nicknames.

User default specifications

GDDM has two formats of user default specifications (UDSs):

- A source-format UDS
- An encoded (or assembled) UDS.

Both formats perform the same range of functions. However, the source-format UDS has to be interpreted by GDDM at run time; this involves an additional processing overhead in comparison with an encoded UDS. Because of this overhead, it is recommended that you use the source-format UDS only for changing defaults that apply to individual end users.

There are three types of UDS:

- ADMMDFT or DEFAULT (see "Changing GDDM's default values" on page 127)
- ADMMEXIT (see "Specifying user exits" on page 104)
- ADMMNICK or NICKNAME (see "Using nicknames to define device characteristics" on page 3).

Passing a source-format UDS to GDDM

Depending on the subsystem under which GDDM runs, you pass your source-format UDS to GDDM in the following ways:

- Under all subsystems, by means of the ESSUDS call, in which you specify the length and data area containing the UDS. The ESSUDS call is described in the *GDDM Base Programming Reference, Volume 1*.

- Except under IMS/VS, by means of an External Defaults File.

The value specified for a particular default in an ESSUDS call overrides any value specified for that default in an External Defaults File.

External defaults file

This section describes the format of an External Defaults File, which can contain many source-format UDSs. A GDDM External Defaults File must be F-format or V-format, with an LRECL of no greater than 256. The recommended format is F(80).

An External Defaults File cannot be used under IMS/VS. Under CICS/VS, it is intended to be used for problem determination purposes only; for details, see the *GDDM Diagnosis and Problem Determination Guide*.

Under TSO, you must allocate (using the ALLOC command) a corresponding file name, the GDDM default of which is ADMDEFS, to the sequential data set that represents the External Defaults File. This must be done before you call GDDM.

In the associated OS/VTAM Print Utility environment, you must allocate a corresponding ddname (the default is ADMDEFS) in the Print Utility JCL to the sequential data set that represents the External Defaults File.

Under VM/CMS, you must ensure that the External Defaults File exists with a suitable filename and filetype (the default is PROFILE ADMDEFS) on a currently accessed disk.

The records in a GDDM External Defaults File must be in one of the following forms:

```
[label] type  value          [optional comments]
```

```
[label] type  value-part1,   [optional comments]
              value-part2,   [optional comments]
                   .
                   .
              value-partn    [optional comments]
```

```
* comment text
```

The records must conform to Assembler-like coding conventions. The conventions are:

- The labels are optional. If specified, they must start in column 1 and must not be longer than 8 characters; they are ignored.
- The type must be preceded by at least one blank.
- The type and value parameters must be separated by at least one blank.
- The value parameter(s) must not contain embedded blanks.
- In a value parameter, a comma (,) followed by a blank or an end-of-record marker indicates that the value is continued on the next noncomment record. The continuation must be preceded by at least one blank. Any text that starts in column 1 is assumed to be part of a label.
- There is no limit on the number of continuation records allowed.
- In a value parameter, a blank or an end-of-record marker that is not preceded by a comma indicates the end of that source-UDS.
- Any text that follows a blank after a value parameter is assumed to be comment text, and is ignored.
- Comment records are optional; they require an asterisk (★) in column 1. Comment records are ignored in all circumstances.
- The source-format UDS can be entered in mixed case. Any lowercase characters are converted to uppercase before processing.

Converting a source-format UDS into an encoded UDS

It is possible (using the GDDM-supplied ADMMDFT, ADMMEXIT, and ADMMNICK macro instructions) to assemble a source-format UDS so that it is converted into the encoded version. You must use the ADMMDFT (not DEFAULT) or ADMMNICK (not NICKNAME) form if you are going to assemble the source-format UDS.

When assembling source-format UDSs to produce encoded UDSs, ADMMDFT START and ADMMDFT END macro invocations can be used to generate the associated length field required in the construction of an External Defaults Module.

Passing an encoded UDS to GDDM

There are three ways of passing an encoded UDS to GDDM for processing:

1. In an External Defaults Module
2. In the SPINIT call
3. In the ESEUDS call.

The value specified for a particular default in an External Defaults File overrides any value specified for that default in an External Defaults Module. Similarly, the value specified for a particular default in a SPINIT call overrides any value specified for that default in an External Defaults File. Finally, a value specified for a particular default in an ESEUDS or ESSUDS call overrides any value specified for that default in a SPINIT call.

The SPINIT call is described in "Using the system programmer interface" on page 103; the ESEUDS call is described in the *GDDM Base Programming Reference, Volume 1*.

External defaults module

In all subsystems, a GDDM installer (and, possibly, an end user) can create a GDDM External Defaults Module by assembling a set of source UDSs, using GDDM-supplied ADMMDFT, ADMMEXIT, or ADMMNICK macros. The resultant module contains a 4-byte length field, followed by a list of encoded-UDSs.

The source of a GDDM External Defaults Module must contain a set of source-UDSs in the same format as the External Defaults File (as described under "External defaults file" on page 2). The set of source-format UDSs must be delimited by ADMMDFT START and ADMMDFT END macro invocations. Also, the source must conform to Assembler-language macro-coding conventions. For example, assuming that no ICTL instruction is used:

- Continuations must begin in column 16.
- Continuations must be flagged by using a nonblank character in column 72.
- The source-format UDSs must be entered in uppercase.

The source of an External Defaults Module must be in the following form:

```
ADMADFx CSECT                  (See the note below)

        ADMMDFT START

[label] type  value            [optional comments]

               .
               .

[label] type  value            [optional comments]

[label] type  value            [optional comments]

        ADMMDFT END

        END
```

Note: ADMADFx is the name of the External Defaults Module appropriate to the user's subsystem; that is,

ADMADFC under CICS/VS,
| ADMADFD under VSE/Batch,
ADMADFI under IMS/VS,
ADMADFT under TSO, and
ADMADFV under VM/CMS.

Using nicknames to define device characteristics

Another type of user default specification (UDS) is a **nickname**. Nicknames provide a way of defining all the characteristics of a device in a table, and then referencing that device on a DSOPEN call just by using the nickname.

Nicknames can be used in this way to extend the range of devices supported by current applications, often without requiring any modification to the applications.

Nicknames also extend the range of devices supported by the GDDM Print Utilities, by providing a mechanism for passing complex device definitions to the asynchronously-called utilities.

Nicknames also enable complex devices to be predefined by the installation programmer, thus simplifying the tasks of the application programmer and the end user. The application programmer and end user retain the ability to override such predefinitions.

For examples of how to use nicknames, see the *GDDM Application Programming Guide, Volume 1.*

The same mechanisms are available for specifying nicknames as can be used for defaults; that is, they can be derived from:

- The External Defaults Module
- The External Defaults File
- A SPINIT call
- An ESEUDS or ESSUDS call.

A nickname can either be in source or encoded format. A source nickname UDS can be defined:

- In an external defaults file
- As an argument to the ESSUDS call.

An encoded nickname UDS can be defined:

- In an external defaults module
- As an argument to the SPINIT call
- As an argument to the ESEUDS call.

The ESEUDS and ESSUDS calls are described in the *GDDM Base Programming Reference, Volume 1.*

The SPINIT call is described in "Using the system programmer interface" on page 103.

Source format of a nickname UDS

The source-format syntax of a nickname UDS is as follows:

```
[label] ADMMNICK  [APPEND|REPLACE,]
        or        [FAM=family,]
        NICKNAME  [NAME=name-list,]
                  [TOFAM=to-family,]
                  [TONAME=to-name-list,]
                  [DEVTOK=device-token,]
                  [PROCOPT=procopt-list]
```

Any number of nickname UDSs can be defined. More than one nickname UDS can be defined for the same family, device name, or both of these.

The nickname parameters are described below. The terms "family", "device token", "name-list", and "procopt" are explained in the description of the DSOPEN call in the *GDDM Base Programming Reference, Volume 1.*

label
: Optional (ignored — it is not part of the UDS).

APPEND|REPLACE
: Specifies whether this specification is to be added to or replaces an existing specification. The default is APPEND.

FAM = family
: A nonnegative integer. The default is FAM = 0.

  The value for "family" does not have to be a valid DSOPEN family number. Only the final "target-family" must be valid.

NAME = name-list
: A name-list in the form of a list of "name-parts," each being a string of from 0 through 8 nonblank characters. A blank "name-part" is represented by a string of 0 characters.

These are valid name-lists:

`NAME=name1,`	one nonblank name-part
`NAME=(name1),`	one nonblank name-part
`NAME=(),`	one blank name-part
`NAME=(name1,name2,name3),`	three nonblank name-parts
`NAME=(name1,,name3),`	two nonblank name-parts and one blank name-part.

The name-list can be null, that is, it is entered as `NAME=,`. This is the default. In this case, if any more nickname parameters are entered, they must be entered without any intervening blanks; all text after a blank is taken as comment text and is ignored.

A name-part in the NAME parameter can also contain a leading or trailing "?" generic character, or both of these. Such a character is considered to match any combination of characters in the same position as the "?." Thus:

`'?abc'`	matches any name-part ending with 'abc'
`'abc?'`	matches any name-part starting with 'abc'
`'?abc?'`	matches any name-part containing 'abc'.

Embedded "?" characters are not allowed, and are diagnosed as being in error.

It is not necessary for "name-list" to be a valid DSOPEN device name-list. Only the final "target-name-list" must be valid.

TOFAM = to-family
An integer (0 or greater). 0 is the default.

TONAME = to-name-list
A name-list in the same form as the NAME parameter (except that "?" generic characters are not allowed). The default is a null to-name-list.

DEVTOK = device-token
A string of 0 through 8 nonblank characters. The default is a null string.

The DEVTOK parameter enables an explicit (non-"★", nonblank) device token to be used, **when** the application program has specified a DSOPEN device token of "★" (or blank). An explicit (that is, non-"★", nonblank) device token specified in the DSOPEN call cannot be overridden.

PROCOPT = procopt-list
A procopt-list in the form of a list of "procopt-specifications" (procopt-specs), thus:

```
PROCOPT=((procopt-spec),(procopt-spec),....)
```

Each procopt-spec is a **keyword** identifying a specific DSOPEN processing option followed by a number of **arguments** valid for that processing option, thus:

```
PROCOPT=((keyword,argument,argument),
         (keyword,argument), ....)
```

The default is a null procopt-list.

Note that the following processing options accept a variable number of arguments:

4	PRINTCTL
18	STAGE2ID
20	ORIGINID
23	SPECDEV
1002	CPSPOOL
1003	CPTAG

Also, the following variable-length processing options are "mergeable", as described below:

4	PRINTCTL
20	ORIGINID

Full details of all DSOPEN processing options currently available are given in Appendix B, "Processing option groups and name-lists" on page 149.

Nickname scanning and matching

GDDM maintains a "nickname-list" containing all the nickname UDSs that have been defined, in the following order:

1. Those defined in an External Defaults Module
2. Those defined in an External Defaults File
3. Those defined by means of the SPINIT call
4. Those defined by means of ESEUDS or ESSUDS calls (in the order in which the calls are made).

When an application program issues a DSOPEN call, GDDM constructs a "source DSOPEN parameter list" that contains

```
"source-family"
"source-name-list"
"source-device-token"
"source-procopt-list"
```

and a "target DSOPEN parameter list" that contains

```
"target-family"
"target-name-list"
"target-device-token"
"target-procopt-list".
```

GDDM initializes both these parameter lists to the DSOPEN call parameters specified by the application program. (The DSOPEN "device-id" parameter is not affected by nickname processing.)

Nickname scanning

GDDM then scans the "nickname-list" for any nickname UDSs whose FAM and NAME parameters **match** the "source-family" and "source-name-list" (in the "source DSOPEN parameter list"). GDDM updates the "target DSOPEN parameter list" using the TOFAM, TONAME, DEVTOK, and PROCOPT parameters of the matching nickname UDSs in the manner described in "Nickname matching" on page 5.

The resulting "target DSOPEN parameter list" is itself subject to more nickname processing. After each scan, GDDM reinitializes the "source DSOPEN parameter list" from the resulting "target DSOPEN parameter list". GDDM then rescans the "nickname-list" for any nickname UDSs that match the modified "source DSOPEN parameter list" and updates the "target DSOPEN parameter list" accordingly.

Any nickname UDSs that are found to match on a scan or a rescan for a DSOPEN are excluded from subsequent rescans for that DSOPEN. This applies even if the nickname UDSs were ignored because the REPLACE parameter was specified in a later nickname.

GDDM repeats the rescanning process until no more matching UDSs are found. The final "target DSOPEN parameter list" is then processed, as described in the description of the DSOPEN call in the *GDDM Base Programming Reference, Volume 1*.

Nickname matching

The DSOPEN parameter list specified by an application program might not match any nickname UDSs in the "nickname-list". In this case, the DSOPEN parameter list is processed directly.

The rules for matching are as follows:

- The FAM value must be 0 or the same as the current "source-family" value for the nickname to match the current "source DSOPEN parameter list".
- For the nickname to match the current "source DSOPEN parameter list", the name-list in the NAME parameter must either be null or must match the current "source-name-list".

  The two name-lists match when the corresponding name-parts are the same (after left-justification, translation to uppercase, and padding with blanks). In this respect, if the name-lists do not contain the same number of name-parts, the shorter name-list is extended with "★" name-parts for the purpose of comparison. For example:

  ```
      'FRED' and '(FRED)'           )
      'FRED' and '(FRED,*)'         ) match
      'FRED' and '(FRED,*,*)'       )
  but 'FRED' and '(FRED,ADMPRINT)' - do not match
  ```
- If APPEND is specified in a matching nickname, the effect of the nickname is merged with that of any preceding nickname in the current scan or rescan, according to the processing rules defined below.

  If REPLACE is specified in a matching nickname, it causes any preceding matching nickname in the current scan or rescan to be ignored. It does not cancel the effect of preceding scans or rescans.

If the nickname is found to match the current "source DSOPEN parameter list", GDDM updates the "target DSOPEN parameter list" as follows:

1. The TOFAM is examined.

   TOFAM = 0
   The "target-family" is not changed.

   TOFAM = nonzero
   The "target-family" is changed to the value to-family.
2. The TONAME is examined.

   TONAME = null
   The "target-name-list" is not changed.

   TONAME = not-null
   The "target-name-list" is changed to be the value of the "to-name-list".
3. The DEVTOK is examined.

   DEVTOK = null, (that is, a null device-token)
   The "target-device-token" is not changed.

   DEVTOK = not-null
   If the current "source-device-token" is "★" or null, the "target-device-token" is changed to be the value of the "device-token". Otherwise, the "target-device-token" is not changed.
4. The PROCOPT is examined.

   PROCOPT = , (that is, a null procopt-list)
   The "target-procopt-list" is not changed.

   PROCOPT = ((procopt-spec),....)
   The procopt-list is inserted into the "target-procopt-list" such that it follows any procopt-lists added so far during the current scan or rescan, but precedes any procopt-lists that were present at the start of the current scan or rescan.

   Note that, in a DSOPEN procopt-list, the latest procopt-specifications take priority; that is to say, where a procopt-list contains two or more procopt-specs for the same processing option, the latest will apply. (Exceptions to this rule are the "mergeable" PRINTCTL and ORIGINID processing options; see below.) This means that the PROCOPT parameter enables an explicit procopt-specification to be applied, if the application program did not specify the corresponding processing option group in the DSOPEN call. A processing option group specified in the DSOPEN call cannot be overridden.

   Note: In a DSOPEN procopt-list, any procopt-specifications that are not applicable to the DSOPEN device family and device name-list are ignored.

Encoded format of a nickname UDS

The encoded format of a nickname UDS is shown here.

The operation of an encoded nickname UDS is identical to that of a source-format nickname UDS. However, the following points should be noted:

- A null source-name-list is expressed by specifying 0 as the number of source-name-parts (N) and by omitting the source-name-parts entirely.
- A null device token is expressed by specifying it as all blanks or all X'00'.
- A null procopt-list is expressed by specifying 0 as the number of procopt-words (P) and by omitting the procopt-words entirely.
- A null target-name-list is expressed by specifying 0 as the number of target-name-parts (T) and by omitting the target-name-parts entirely.

Word 1	Length (in full-words): 2N+P+2T+10
2	UDS-code: 2001
3	Replace (0) or Append (1)
4	Source family
5	Number of source name-parts (N)
6	Source-name-part 1 (8 bytes) (padded with blanks, as necessary)
7	
8	Source-name-part 2 (8 bytes) (padded with blanks, as necessary)
9	
	. . .
2N+4	Source-name-part N (8 bytes) (padded with blanks, as necessary)
2N+5	
2N+6	Target family
2N+7	Device token (8 bytes) (padded with blanks, as necessary)
2N+8	
2N+9	Number of procopt words (P)
2N+10	Procopt-word 1
2N+11	Procopt-word 2
	. . .
2N+P+9	Procopt-word P
2N+P+10	Number of target names (T)
2N+P+11	Target-name-part 1 (8 bytes) (padded with blanks, as necessary)
2N+P+12	
2N+P+13	Target-name-part 2 (8 bytes) (padded with blanks, as necessary)
2N+P+14	
	. . .
2N+P+2T+9	Target-name-part T (8 bytes) (padded with blanks, as necessary)
2N+P+2T+10	

Chapter 2. Using GDDM under CICS/VS

This chapter describes the use of GDDM under the CICS/OS/VS and CICS/DOS/VS subsystems. It contains these sections:

- Overview
- Programming languages and restrictions
- Compiling and link-editing GDDM application programs
- Using the nonreentrant interface
- Using the system programmer interface by means of dynamic load
- CICS pseudoconversational applications
- Data sets and file processing
- Display terminal conventions
- Using GDDM with Basic Mapping Support
- CICS/VS GDDM default error exit
- Requesting transaction-independent services
- Using the resource audit trails
- Running application programs in VSE batch mode.

The print utility is described in Chapter 7, "The GDDM print utilities" on page 47.

Application programs, which must be written in the command-level (EXEC) interface, are treated as normal CICS/VS applications except that they must be link-edited with GDDM interface modules.

A working knowledge of CICS/VS is assumed throughout.

Programming languages and restrictions

GDDM can be used by CICS/VS command-level (EXEC) application programs written in PL/I, COBOL, or Assembler language.

COBOL restriction: COBOL programs run under CICS/VS must not use the STOP RUN statement.

Compiling and link-editing GDDM application programs

Examples of the JCL that can be used to compile and link-edit application programs written in COBOL, PL/I, and Assembler language are listed on pages 17 through 22 at the end of this chapter.

Compiling a PL/I program

If you use the GDDM-supplied declarations in your program, you must access the libraries containing them before compiling.

Link-editing a GDDM application program

An application program using GDDM under CICS/VS must be link-edited with CICS/VS command-level (EXEC) stubs in the usual way, as described in the *CICS/VS Installation and Operations Guide.* Unless the application program uses dynamic load facilities to access GDDM using the System Programmer Interface (see "Using the system programmer interface with dynamic load" on page 8), the program must also be link-edited with an appropriate GDDM interface module or modules.

Link-editing under CICS/OS/VS

Under CICS/OS/VS, the required interface module can be explicitly included in the link-edit process. Or, if the application program uses one of the other FSINIT entry points described in the *GDDM Base Programming Reference, Volume 1*, the interface module can be included by linkage editor automatic library call facilities. The following is a list of GDDM interface modules for CICS/OS/VS:

Interface	**Interface module**	**FSINIT alternative entry**
Nonreentrant	ADMASNC	FSINNC
Reentrant	ADMASRC	FSINRC
System programmer	ADMASPC or ADMASPKC (see note)	—

Note: ADMASPKC is an alias entry point for ADMASPC, and is provided for compatibility with GDDM-PGF Version 1 Release 1.

Link-editing under CICS/DOS/VS

Under CICS/DOS/VS, two GDDM interface modules are required, and they should be explicitly included in the link-edit process. The first interface module should be selected according to the form of interface used by the application program and the functions required, as follows:

Interface	**Interface module**	**Functions included**
Nonreentrant	ADMASNB	GDDM
Reentrant	ADMASRB	GDDM
Nonreentrant	ADMASNO	GDDM and GDDM-PGF
Reentrant	ADMASRO	GDDM and GDDM-PGF
System	ADMASP	GDDM and GDDM-PGF
programmer	ADMASP	GDDM and GDDM-PGF

The second GDDM interface module required is ADMASLC. It is used for all programs. This has an alias entry point of ADMASKC, which is provided for compatibility with GDDM-PGF Release 1.

In the absence of an explicit ENTRY statement, it is important to include the application program module before the relevant GDDM interface modules, to ensure that the application program entry point is correctly identified.

Thus, a CICS/DOS/VS PL/I application program using the reentrant interface to GDDM can be link-edited as follows:

```
// JOB jobname
// OPTION CATAL
   PHASE phase-name, *
   INCLUDE DFHPL1I
   INCLUDE pl/i-relocatable-module
   INCLUDE ADMASRB
   INCLUDE ADMASLC
// EXEC LNKEDT
/&
```

and a CICS/DOS/VS COBOL application program using the nonreentrant interface to GDDM can be compiled and link-edited as follows:

```
// JOB jobname
     . . . . .
     . . . . .
     Standard
  Translate Step
     . . . . .
     . . . . .
// OPTION CATAL
   PHASE phase-name, *
   INCLUDE DFHECI
// EXEC COBOL
   INCLUDE ADMASNB
   INCLUDE ADMASLC
// EXEC LNKEDT
/&
```

Using the nonreentrant interface of GDDM

GDDM provides a mechanism for using the nonreentrant interface form under CICS/VS while still allowing GDDM and its invoking application program to be quasi-reentrant. To do so, the application programmer should reserve an area of 8 bytes in the associated Transaction Work Area (TWA). This may require changes in the corresponding transaction definition in the CICS/VS Program Control Table (PCT). The programmer should then define an external control section (CSECT) named ADMUOFF, to be link-edited with the application program and the GDDM nonreentrant interface module. This should contain a full-word defining the offset in the TWA of the area reserved for GDDM's use.

Thus, for application programs that would not otherwise require a TWA, the following would be sufficient:

1. Define a TWA of length 8 bytes by specifying the corresponding option in the transaction definition in the CICS/VS Program Control Table.
2. Define an ADMUOFF CSECT containing a full-word of value zero, to be link-edited with the application program.

The ADMUOFF CSECT can be defined using standard Assembler language facilities. Thus:

```
ADMUOFF CSECT
INIT    DC   F'0'
        END
```

Or, high-level language constructs can be used, where such are available. In PL/I, the CSECT could be generated by a declaration of the form:

```
DECLARE ADMUOFF STATIC EXTERNAL FIXED BINARY (31)
        INITIAL(n);
```

GDDM uses the area reserved in the TWA to store an Application Anchor Block (AAB), in the format described for the reentrant interface in the *GDDM Base Programming Reference, Volume 1.* When the nonreentrant interface is invoked, GDDM verifies that the value contained in ADMUOFF is consistent with the length of the TWA defined for the invoking transaction.

Through this mechanism, GDDM operates in a quasi-reentrant way. Although the GDDM nonreentrant interface module is **not** read-only, it does not prevent an invoking transaction from servicing more than one CICS/VS terminal at the same time.

Using the system programmer interface with dynamic load

If an application uses only the system programmer interface, all invocations of GDDM are through the entry point ADMASP. This entry point can be resolved by link-editing the application with the GDDM interface module ADMASPC, as described under "Link-editing under CICS/OS/VS" on page 7.

Or, the application can avoid these linkage-edit considerations by using CICS/VS facilities (EXEC CICS LOAD) to load dynamically a GDDM interface module ADMASPLC containing the ADMASP entry point as follows:

```
EXEC CICS LOAD PROGRAM(ADMASPLC) ENTRY(admasp-addr)
     SET(dummy-var)
```

CICS/VS pseudoconversational applications

A CICS pseudoconversational application is one which appears to the terminal user as a normal conversational transaction, but is, in fact, a series of separate transactions where the CONVERSE is implemented as SEND and RECEIVE. One transaction ends with a SEND, and the next starts with a RECEIVE.

In this way, system resources can be released for the duration of "operator think time" thus making more efficient use of CICS.

GDDM provides pseudoconversational support for procedural, mapped, or high-performance alphanumeric data and output-only graphics and image by means of a strictly defined protocol for GDDM application call sequences.

Essentially, while operating in pseudoconversational mode, GDDM storage and resources (except for device query data) are released at the termination of a particular transaction, and are reinitialized when the next transaction is reinvoked by CICS to process the next device input.

As no information is retained by GDDM across transactions (other than device query data), it is the responsibility of the application to ensure correct continuity of the application; see below for details of the call sequences to be used.

The following GDDM calls have a changed function when pseudoconversational mode is being used:

DSOPEN
The PSCNVCTL processing option indicates to GDDM whether pseudoconversational mode is in use, and whether this is the Start of it, or a Continuation.

- The processing option group code is 25
- The length is 2 full-words
- The values are 0, 1, and 2 corresponding to NO, START, and CONTINUE respectively
- The default is NO.

The nickname syntax for this processing option is:

```
(PSCNVCTL,{NO|START|CONTINUE})
```

ASREAD
When the application is in "Continue pseudoconversational" mode (PSCNVCTL,CONTINUE), the first ASREAD call issued by the application causes the output transmission to be suppressed, and only the input part of the ASREAD call functions.

Subsequent ASREAD calls work in the usual way, that is, they result in output plus a "wait" for input. In this way, transactions can drop into Conversational Mode if they need to; see the description of the CLEAR key handling and line-output errors below.

DSCLS
If pseudoconversational mode is in use, a DSCLS call always causes the device keyboard to be unlocked. Also, two options are provided that can be used by pseudoconversational applications to end the pseudoconversational mode, and are available to conversational applications to cause explicit keyboard Unlock.

The complete DSCLS options and their meanings are:

0	Erase the screen; if in pseudoconversational mode, unlock the keyboard, and save any changed device data.
1	Do not erase the screen; if in pseudoconversational mode, unlock the keyboard, and save any changed device data.
2	Erase the screen and unlock the keyboard; if in pseudoconversational mode, release the saved device data.
3	Do not erase the screen but unlock the keyboard; if in pseudoconversational mode, release the saved device data .

The following application scenario illustrates the call protocol for pseudoconversational mode:

- On the initial invocation of the transaction:
  - FSINIT
  - DSOPEN (Start pseudoconversational mode)
  - Create alphanumeric data for the first screen
  - Create any graphics output
  - FSFRCE
  - DSCLS (Option 1 – do not erase the screen)
  - FSTERM
  - EXEC CICS RETURN TRANSID(Tname) COMMAREA(Carea)

    The array "Carea" should contain any information required to continue the transaction processing, for example, Application Data Structures used for output of mapped data.
- On subsequent invocations of the transaction:
  - FSINIT.
  - DSOPEN (Continue pseudoconversational mode).
  - Create alphanumeric data for the "previous" screen using the identical set of calls used the last time, and also, if mapping is used, with the same Application Data Structures (as saved in "Carea").
  - **Do not** issue any graphics calls.
  - ASREAD.
  - Process input in the usual way.
  - Create alphanumeric data for the next screen.
  - Create any graphics output.
  - FSFRCE.
  - DSCLS (Option 1 – do not erase the screen).
  - FSTERM.
  - EXEC CICS RETURN TRANSID(Tname) COMMAREA(Carea) LENGTH(Clen).

    The array "Carea" should contain any information required to continue the transaction processing; in particular, it should contain the ADSs used for the output of any mapped data.
- Use DSCLS with Option 2 or 3 to terminate the pseudoconversation.

As stated above, the first ASREAD call in a transaction specifying "Continue pseudoconversational" mode, only performs the input function; all output is suppressed.

There are, however, two exceptions to this rule.

The first exception, when using mapped alphanumerics, is where the map group requests automatic handling of the CLEAR key.

In this case, the ASREAD call performs as usual; that is, it bypasses output and processes the input data (only a cursor address and the CLEAR aid), whereupon mapping signals a screen refresh.

The result of this is as if a second ASREAD call has occurred; that is, the screen is output again and the transaction waits for input.

Thus the ASREAD call effectively works in the usual way, and the transaction becomes a conversation for this invocation.

The other exception is where a GDDM line-output error message occurs before the ASREAD call.

In this case, the screen contents have been destroyed, and for GDDM to continue to process correctly, the screen has to be created again.

Thus once more, the ASREAD call works in the usual way; that is, output plus a "wait for input" and the transaction becomes "conversational" for this invocation.

Always-unlock-keyboard mode
Use of the always-unlock-keyboard processing option improves the performance of CICS pseudo-conversational applications by unlocking the keyboard at FSFRCE instead of DSCLS.

Data sets and file processing

When running under CICS/VS, GDDM Base and GDDM-PGF use three types of file processing:

- CICS/VS command-level (EXEC) File Control facilities, to read and write data on a VSAM key-sequenced data set.
- CICS/VS command-level (EXEC) Transient Data facilities, to write data for subsequent internal or external processing.
- CICS/VS command-level (EXEC) Temporary Storage facilities, to read and write data required for queued printer and external defaults support.

GDDM-IMD uses additional types of file processing; for details, see the *GDDM Interactive Map Definition* manual.

File control facilities

GDDM uses the File Control facilities to:

- Store and retrieve Image Symbol Sets (ISS) and Vector Symbol Sets (VSS), as required by calls to GSLSS, PSLSS, PSLSSC, SSREAD, and SSWRT, and through the Image Symbol Editor.
- Store and retrieve device-dependent pictures, as required by calls to FSSAVE, FSSHOR, and FSSHOW.
- Retrieve GDDM-IMD-generated mapgroups, as required by calls to MSPCRT, MSQADS, MSQGRP, MSQMAP, and MSREAD.
- Store and retrieve Graphics Data Format (ADMGDF) files, as required by calls to GSSAVE and GSLOAD.
- Store and retrieve image files, as required by calls to IMAPT and IMAGT.

GDDM maintains these symbol sets, pictures, generated mapgroups, and ADMGDF files as keyed records in VSAM key-sequenced data sets shared by transactions running in the CICS/VS subsystem. The VSAM data sets are referred to within GDDM using CICS/VS File Control statements, and the data sets specified in the DATASET option of these statements must be defined in the CICS/VS File Control Table (FCT). The VSAM data sets must be opened, either when CICS/VS is initialized, or dynamically, before GDDM requires access to them. The underlying OS/VS or DOS/VS data sets must have characteristics as shown in Table 1 on page 12. Procedures for creating and initializing suitable VSAM data sets are described in *GDDM Installation and System Management for MVS* or *GDDM Installation and System Management for VSE*.

The default VSAM data set names are as defined in Table 1 on page 12. These names can be changed, if required, after installation, as described in Chapter 1, "Customizing your program and its environment" on page 1.

The use of the VSAM data sets can be controlled by the ESLIB routine whose syntax is described in the *GDDM Base Programming Reference, Volume 1*. This routine establishes the set of VSAM data sets that are to be used to store or retrieve a given type of object. The VSAM data sets used are identified to this routine by a list of file names.

The VSAM data sets identified are searched in the order given in an attempt to find an object. An object is stored only by means of the first data set name of the list, even though it may have been retrieved from another one. If no data set name list is provided, only the default data set name is used for retrieving and storing GDDM objects.

GDDM uses CICS/VS Task Control ENQ/DEQ facilities to ensure the integrity of data as it is written or read on the VSAM data sets. Specifically, GDDM ensures that the particular records defining the content of a symbol set, picture, or generated mapgroup cannot be updated by one transaction while being read by another. If additional control of the use of the VSAM data sets is required (such as restricted write access), this should be implemented by security mechanisms external to GDDM, such as described in the *CICS/VS Facilities and Planning Guide*.

GDDM symbol sets, pictures, generated mapgroups, and ADMGDF files are stored on the VSAM data sets as 400-byte records, with an embedded key in the first 20 bytes, as follows:

Byte 0.....7	8....15	16....19	20....
Name	Type	Record sequence number	Data

Name is that specified in the GDDM call as "symbol-set-name", "picture-name", "group-name", or "name", subject to the character-substitution rules described in "Selecting symbol sets by device type" on page 67.

Type is an 8-byte character string identifying the type of the record, for example, "symbol set" or "picture", and is defined in Table 1 on page 12.

Record sequence number is a 4-byte binary full-word that sequences and uniquely identifies each record within a symbol set or picture.

This key format is such that, if required, all of the records defining a specific symbol set or picture can be deleted without calling GDDM. This can be done by using the CICS/VS File Control GENERIC DELETE function:

```
EXEC CICS DELETE DATASET (VSAM-data-set-name)
          RIDFLD (first-16-bytes-of-key)
          KEYLENGTH(16)
          GENERIC
```

The Interactive Chart Utility (part of GDDM-PGF) includes a directory function that supports list, delete, and copy operations on GDDM objects such as symbol sets, pictures, generated mapgroups, and ADMGDF files.

Transient data facilities

GDDM uses CICS/VS Transient Data facilities to:

- Write object modules resulting from requests from the Image Symbol Editor.
- Write output destined for a system printer device as the result of calls to DSOPEN and DSCLS.
- Write trace records resulting from the FSTRCE function.
- Write error log records resulting from invocation of the GDDM CICS/VS Default Error Exit.

Object modules are written consecutively to a single transient data destination. This must be defined in the CICS/VS Destination Control Table (DCT), typically in a manner that would route the object modules to a predefined extrapartition data set. Each object module generated contains a control section (CSECT) with the name as specified by the appropriate utility, and has a form suitable for link-editing with an application program for subsequent reference, typically using the GSDSS or PSDSS calls.

System printer device output is written to the transient data destination identified using the DSOPEN call. This must be defined in the CICS/VS Destination Control Table (DCT), typically in a manner that would route the output to a predefined extrapartition spool data set. If so routed, the definition should indicate the presence of ASA control characters in the data generated by GDDM.

GDDM uses CICS/VS Task Control ENQ/DEQ facilities to ensure that system printer output resulting from a single DSOPEN...DSCLS sequence remains contiguous, and is not interleaved with the output from another CICS/VS transaction. The application programmer should ensure that the use of these facilities in multiple transactions does not introduce excessive transaction delays or interlocks.

Trace records are written to a single transient data destination. This must be defined in the CICS/VS Destination Control Table (DCT), typically in a manner that would route the output to a predefined extrapartition spool data set. If so routed, the definition should indicate the presence of ASA control characters in the records generated by GDDM.

Trace records from different transactions may be interleaved. For this reason, each record contains the corresponding transaction name and terminal identifier. For a description of the use of the FSTRCE function, and of the format of the trace records, see the *GDDM Diagnosis and Problem Determination Guide.*

For information on the trace facilities obtainable with the new GDDM external default TRCESTR, see the *GDDM Diagnosis and Problem Determination Guide.*

The above Transient Data destination names are as defined in Table 1 on page 12. These names can be changed, if required, after installation (by specifying a value for the CICTRCE option, as described under "GDDM external defaults – CICS/VS" on page 128).

Error log records are written as they occur, to a single transient data destination, which must be defined in the CICS/VS Destination Control Table (DCT), in a manner to suit the installation's requirements. Typically, the destination would be defined as an extrapartition destination, which would route the error log records to an external data set for subsequent printing.

Error log records from different transactions may be interleaved. For this reason, each record contains the corresponding transaction name, number, and terminal identifier. The format of these error log records is described under "CICS/VS GDDM default error exit" on page 14.

The Transient Data destination name for error log records is ADML, and cannot be changed.

The programmer should ensure that the Transient Data destination names required are all defined in the appropriate CICS/VS tables. The underlying OS/VS or DOS/VS data sets must have characteristics as shown in Table 1 on page 12.

Temporary storage facilities

GDDM uses CICS/VS Temporary Storage facilities to write data to intermediate data sets used in the processing of calls to DSOPEN, DSCLS, FSOPEN, and FSCLS for queued printer output. The temporary data sets created are read by the GDDM CICS/VS Print Utility, and after output to the printer is completed, the data sets are purged.

By default, for queued printer output, GDDM selects temporary storage queue names beginning with the prefix "ADMT". This prefix can be changed, if required, by specifying a value for the CICTSPX option, as described under "GDDM external defaults – CICS/VS" on page 128.

GDDM also uses CICS/VS Temporary Storage facilities to read temporary External Defaults files. Such files are intended to be used for problem determination purposes only. For details, see the *GDDM Diagnosis and Problem Determination Guide.*

By default, for External Defaults files, GDDM assumes temporary storage queue names beginning with the prefix "ADMD". This prefix can be changed, if required, by specifying a value for the CICDFPX option, as described under "GDDM external defaults – CICS/VS" on page 128.

Also, GDDM uses temporary storage to hold Device Query data when running in pseudoconversational mode. The queue name is formed from a prefix "ADMQ", which can be changed, if required, by specifying a value for the CICTQRY option as described under "GDDM external defaults – CICS/VS" on page 128, and the terminal identifier.

Table 1. GDDM data-set characteristics for CICS/VS

Type of data	GDDM default name or record type	Data-set type	Data characteristics
Symbol sets	Data set name = ADMDF Record type = ADMSYMBL	Records in VSAM data set	RECORDSIZE (400 400) KEYS(20 0)
Pictures	Data set name = ADMF Record type = ADMSAVE	Records in VSAM data set	RECORDSIZE (400 400) KEYS(20 0)
Generated mapgroup	Data set name = ADMF Record type = ADMGGMAP	Records in VSAM data set	RECORDSIZE (400 400) KEYS(20 0)
GDF files	Data set name = ADMF Record type = ADMGDF	Records in VSAM data set	RECORDSIZE (400 400) KEYS(20 0)
Object modules	Queue name = ADMD	Transient data queue	Fixed-length records, length 80 bytes
System printer output	Queue name = ADMS	Transient data queue	Variable-length records, length 142 bytes or greater (see note 4)
Queued printer files	(assigned by GDDM)	Temporary storage data set	(assigned by GDDM)
Trace records	Queue name = ADMT	Transient data queue	Variable-length records, maximum length 137 bytes (including 4-byte RDW)
Error log records	Queue name = ADML (cannot be modified)	Transient data queue	Variable-length records, maximum length 120 bytes
External defaults files	Queue name = ADMDxxxx (xxxx is the CICS/VS terminal identifier)	Temporary storage data set	Variable-length records, maximum length 256 bytes
Pseudo-conversational saved device information	Queue name = ADMQxxxx (xxxx is the CICS/VS terminal identifier)	Temporary storage data set	Assigned by GDDM

Notes:

1. Record types for data stored in VSAM data sets cannot be changed.
2. For Transient Data DOS/VS disk output data sets, another 8 bytes, required by LIOCS for creation of the count field, should be added to the block size.
3. The definition of Transient Data queues for System Printer Output should indicate the use of ASA control characters,
   for OS/VS RECFORM = VARUNBA or VARBLKA
   for DOS/VSE CTLCHR = YES
4. The record length specified for System Printer Output queues should be enough to contain the 4-byte Record Descriptor Word (RDW), the ASA control character, any Translation Reference Character (TRC) for 3800 devices, and the maximum number of columns for the type of System Printer selected by the application. The value of 142 is enough for any of the System Printer device characteristic tokens distributed with GDDM.
5. The output for all 3800 devices should contain table reference characters (TRCs) and so, for OS/VS, the parameter DCB=OPTCD=J must be included in the output JCL. Under OS/VS or DOS/VS, additional DCB or SETPRT parameters, such as CHARS, FLASH, FORMS, and so on, may be required.
6. For more information, see the *OS/VS2 MVS JCL* manual or the *DOS/VSE System Control Statements* manual.

Display terminal conventions

In general, the CLEAR key and all PA and PF keys are available to be returned as terminal input by means of the GDDM ASREAD function. However, specific PA keys that were defined in the CICS/VS System Initialization Table for other purposes, such as printing, are not available for GDDM purposes.

Using GDDM with Basic Mapping Support

It is possible to write a CICS/VS transaction that uses both Basic Mapping Support (BMS) and GDDM functions to manage the screen. Three methods for doing this are described below. Note that GDDM uses CICS/VS Terminal Control facilities to manage the screen directly. For this reason, GDDM pictures displayed on the terminal cannot be paged using BMS paging mechanisms.

An application program that uses both CICS/VS Terminal Control and GDDM functions for input/output operations is subject to the same considerations. However, once GDDM is initialized, no transmissions should be sent by CICS/VS Terminal Control that would alter the state of the device, other than the screen buffer. In particular, no structured fields to alter the state of PS sets (other than those reserved by the GDDM PSRSV call) should be transmitted.

Using GDDM and Basic Mapping Support consecutively

When GDDM has formatted the screen and displayed data by means of calls to ASREAD, or FSFRCE, or both of these, the displayed panel can be replaced with one generated by BMS using a command such as:

```
EXEC CICS SEND MAP('map-name')...ERASE
```

The ERASE option should be specified, because BMS is not aware of the GDDM screen interactions that occurred since the last BMS interaction.

The BMS map can use any of the field description functions supported by CICS/VS, including references to PS sets loaded by GDDM calls. The application program can then read data entered by the terminal user using BMS.

When the BMS interactions are completed, GDDM can be called again to present the original or updated data. A call to FSREST(0) should be issued before calling FSFRCE or ASREAD, because GDDM would not be aware of the BMS screen interactions. GDDM interactions can then continue until the application program calls BMS again.

Using GDDM and BMS concurrently without coordination mode

It is possible to use GDDM and BMS to display data at the same time on the same screen. In this type of operation, it is recommended that GDDM be used only to **output** graphics data, and that BMS be used for all alphanumeric input/output processing. Specifically, the GDDM ASMODE function should **not** be used to set the character reply mode.

The GDDM picture should be presented first, using FSREST(0) if necessary to clear any preceding BMS data. The BMS map(s) should then be transmitted, omitting the ERASE option. The map(s) should be defined so that all screen areas used by GDDM for graphics are in protected fields with normal attributes (nonhighlighted, nonselectable, neutral color, normal intensity, and standard character set). The application program can then read data entered by the terminal user using BMS.

On completion of terminal data entry, the GDDM FSREST(0) call should again be used on resuming GDDM operations.

If the FSCOPY call is used to copy a panel containing both GDDM and BMS data, only the GDDM data is printed, because GDDM is unaware of the BMS data.

Using GDDM and BMS concurrently with coordination mode

Note: BMS is not supported with CICS pseudo-conversational mode.

The difficulty with the above method of using both BMS and GDDM is that whenever GDDM rewrites the screen it may choose to totally erase the screen and start afresh. This, of course, also removes any existing BMS output.

This problem is avoided if the device used for output is explicitly opened with the DSOPEN statement and the "coordination" mode of operation selected.

When GDDM generates the data streams for such a device it **never** totally erases the screen when an FSFRCE or ASREAD is issued. Instead it just rewrites the contents of the area covered by the graphics field. Any screen erasure required then becomes the responsibility of the application using either Terminal Control or BMS requests.

The following points should be noted:

- GDDM protects the graphics field by a column of attribute bytes to its left, or at the end of the preceding row if the graphics field is positioned in the first column.

  The BMS maps should not use the area used by these attribute bytes. If they do, the results are unpredictable.
- GDDM locks the keyboard when the device is opened, to interrogate the device properties. Therefore, any BMS request to release the keyboard should be issued after calling GDDM to open the device.
- GDDM writes only to the area of the screen covered by the graphics field. Further, no alphanumeric fields, even if they are within the graphics field, are written to the screen.
- ASREAD does not wait for input – it behaves as FSFRCE.
- Programmed symbol (PS) sets may still be loaded within coordination mode.

- The application program must erase the screen before issuing the first GDDM output request, to establish either the default or alternate screen size.
- After receipt of a CLEAR key the application should rewrite the BMS portions of the screen before issuing FSREST and FSFRCE calls to reestablish the GDDM picture.
- The action of the default error exit is to erase the screen and display a prompting message. This causes disruption of the BMS-managed screen layout. Therefore, the application should use the FSEXIT function to redefine the handling of errors.

CICS/VS GDDM default error exit

The function of the GDDM Default Error Exit is generally described in the *GDDM Base Programming Reference, Volume 1*. When GDDM is running under CICS/VS, the Default Error Exit operates as follows:

- The screen is cleared, and diagnostic messages describing the error are displayed.
- Another message, describing the other actions available to the terminal user, is displayed.
- If the terminal user presses the CLEAR key at this point, the screen is cleared and GDDM returns control to the point in the application program where the error exit was invoked. GDDM also retransmits the screen buffer contents on the next terminal input/output-related call.
- If the terminal user uses any key other than CLEAR, GDDM calls the CICS/VS Command Level ABEND facility with an ABCODE of "G000", indicating that the ABEND is in response to an error message displayed on the terminal.

In either of the above cases, GDDM tries to write one or more error log records to the CICS/VS Transient Data destination ADML, if it was specified in the CICS/VS Destination Control Table. The error log records contain the diagnostic messages displayed on the terminal, prefixed by transaction identification information, as follows:

Byte 0...3	4	5...8	9	10...13	14 15	16...
Trans-action ID	/	Task Number	/	Terminal ID		Diag-nostic Text

Note that in the special case of initialization errors a choice of action is not available to the terminal user after the diagnostic message is displayed. For these errors, GDDM unconditionally ABENDS, with an ABCODE of "G000", after displaying the corresponding diagnostic message on the terminal.

Requesting transaction-independent services

When running under CICS, GDDM usually uses transaction-dependent services to acquire storage and load programs. That is, GDDM uses CICS/VS services that ensure that storage and program resources are released should the task terminate normally or abnormally.

Application programs using SPINIT to initialize GDDM can request that transaction-independent services be used, by setting the CICTIF=YES option in an encoded UDSL in the SPINIT call; see "Format of the system programmer interface block" on page 104. This causes GDDM to use CICS/VS storage and program services in such a way that storage and program resources are not released at task or transaction termination.

Care must be taken when using this option, to ensure that resources are eventually released in all situations including abnormal termination of the task or transaction. The audit trail functions described in the following section can be used to monitor and control the status of the resources.

Using the resource audit trails

Care must be taken when requesting transaction-independent services as described above to ensure that resources are released in all situations including abnormal termination of the task or transaction.

Application programs requesting such services can also request resource audit trails, by specifying the CICAUD option in an encoded UDSL in the SPINIT call; see "Format of the system programmer interface block" on page 104. The application program can use this option to provide the addresses of 4-byte audit trail anchors for storage and program resources.

The storage audit trail is maintained as follows:

- All blocks of storage acquired but not yet released by GDDM are chained together by 4-byte pointers at offset +0 in each storage block.
- The storage audit trail anchor, addressed by the CICAUD option, is set by GDDM to locate this chain of storage blocks.
- The 4-byte pointer in the last storage block in the chain is set to the initial value of the storage audit trail anchor, as defined by the application program.
- If all storage blocks were released (as at termination), the storage audit trail anchor is reset by GDDM to its initial value.

Thus, if abnormal termination occurs, the storage audit trail anchor can be used to locate those blocks of storage that are not yet released by GDDM. To be effective, the audit trail anchor should be initialized to an identifiable value, such as 0.

The program audit trail is maintained as follows,

- At initialization, GDDM allocates a "program hold" table of 41 entries, each eight bytes in length. All but the last entry are initialized to blanks. The last entry is an "end-of-table" marker and is initialized to a value of X'FFFFFFFF'.
- The program audit trail anchor located by the field SPIBPRAP is set by GDDM to address this program hold table.
- Whenever GDDM loads a program, it replaces a blank entry in the program hold table with the program name.
- Whenever GDDM deletes a program, it resets the corresponding entry in the program hold table to blanks.

Thus, if abnormal termination occurs, the program hold table can be used to determine the names of those programs that are not yet deleted by GDDM.

Note that the program hold table itself is in a storage block in the storage audit chain. Therefore, any processing of this table should be performed before processing the storage audit chain.

GDDM application programs in VSE batch mode

GDDM application programs can be run in batch mode under VSE, provided the only devices that they open are page printers – in GDDM terms, family-4 devices. GDDM page printer output takes the form of a file containing either a primary or a secondary data stream.

A primary data stream is a complete document suitable for processing by a printer driver program – the Print Services Facility (PSF) for 38xx output or the Composed Document Print Facility (CDPF) for 4250 output, or equivalent programs. Conversely, a page segment must be imbedded into a document by a formatting program such as SCRIPT/VS, which in turn produces a complete document for processing by the printer driver program.

More information about printing on VSE systems is given in *GDDM Installation and System Management for VSE*.

In addition to user-written application programs, three new GDDM utilities can run in VSE batch mode:

- the Image Print Utility, see page 56,
- the VSE Print Job Utility, see page 48,
- and the Composite Document Print Utility, see page 57.

Instructions for running these are given in Chapter 7, "The GDDM print utilities" on page 47.

Link-editing

Before an application program can be run in VSE batch mode, it must be link-edited with two GDDM interface modules. One of these, ADMASLD, supports VSE batch mode. Here is some model job control language (JCL) for a link-edit job:

```
**************************************************
* This JCL assumes that DLBL, EXTENT, and LIBDEF*
* statements have already been used to define   *
* the GDDM relocatable libraries                 *
**************************************************
*
// JOB jobname
// OPTION CATAL
   PHASE phase-name,*
   INCLUDE phase-name
*
* In the following INCLUDE statement,
* leave   ADMASNB unchanged for GDDM Base using
*                 nonreentrant interface
* replace ADMASNB by ADMASRB for GDDM Base using
*                 reentrant interface
*   or by ADMASNO for GDDM Base + GDDM-PGF using
*                 nonreentrant interface
*   or by ADMASRO for GDDM Base + GDDM-PGF using
*                 reentrant interface
*   or by ADMASP  if using the system programmer
*                 interface
   INCLUDE ADMASNB
   INCLUDE ADMASLD
// EXEC LINKEDT
/*
/&
```

Large 4250 page segments

A formatting program such as SCRIPT/VS can imbed a page segment in two ways: it can either include the complete segment inline, which means physically putting it into its output file; or include the name of the segment, leaving the printer driver program to physically imbed it in the final output.

The CDPF program limits the size of inline page segments to 40K bytes. If you have larger page segments, they cannot be passed to CDPF inline. Instead, they must be stored in a VSAM ESDS file, from where CDPF will read them when required. However, GDDM stores any page segments that it creates in a phase library, not in a VSAM file. To overcome this problem, there is a GDDM utility called ADMUP2VD that copies page segments from the phase library to a VSAM ESDS file.

ADMUP2VD should not be used in the shared virtual area (SVA).

Here is some sample JCL to copy a page segment from a phase library to a VSAM ESDS file:

```
* $$ JOB JNM=CPYPHASE,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=D,DEST=(node,userid),JSEP=1
// JOB CPYPHASE
// DLBL gddm,'gddm.library.name'
// EXTENT ,volid
// DLBL libname,'phase.library.name'
// EXTENT ,volid
// LIBDEF *,SEARCH=(libname.sublib,gddm.sublib)
// DLBL IJSYSUC,'user.catalog.name',,VSAM
// DLBL fname,'vsam.file.name',,VSAM
// EXEC IDCAMS,SIZE=AUTO
 DELETE (vsam.file.name)    -
        CLUSTER
/*
// EXEC IDCAMS,SIZE=AUTO
 DEFINE CLUSTER                      -
          (NAME(vsam.file.name)      -
           NONINDEXED                -
           RECORDFORMAT(V)           -
           RECORDSIZE(4000 8202)     -
           TRACKS(5 5)               -
           VOL(volid))               -
           DATA                      -
           ( NAME(data.file.name) )
/*
IF $RC>4 THEN
GOTO $EOJ
// EXEC ADMUP2VD,SIZE=ADMUP2VD,PARM='fname'
//*
//&
* $$ EOJ
```

Only the name of the phase to be copied must be specified on the PARM='fname' parameter (up to eight characters long). The type PHASE must not be included.

Spill files

GDDM uses spill files when creating output for page printers, unless told otherwise in a processing option. This is true whether the processing is done by a user-written application or a GDDM utility. The spill files need to be defined. Some sample JCL for doing this is shown below.

```
* $$ JOB JNM=DEFSPILL,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=D,DEST=(node,userid),    *
         JSEP=1
* $$ LST CLASS=A,DISP=D,LST=1A0,               *
         DEST=(node,userid),JSEP=1
// JOB DEFSPILL
// DLBL IJSYSUC,'user.catalog.name',,VSAM
// DLBL ADM0001,'ADM00001.SPILL.FILE',,VSAM
// DLBL ADM0002,'ADM00002.SPILL.FILE',,VSAM
// EXEC IDCAMS,SIZE=AUTO
 DELETE (ADM00001.SPILL.FILE)        -
        CLUSTER
 DEFINE CLUSTER                      -
        (NAME(ADM00001.SPILL.FILE)   -
         NONINDEXED                  -
         REUSE                       -
         RECORDSIZE(1000 2000)       -
         RECORDS(10 10)              -
         VOL(PAC371))                -
         DATA                        -
        (NAME(ADM00001.SPILL.DATA))
 DELETE (ADM00002.SPILL.FILE)        -
        CLUSTER
 DEFINE CLUSTER                      -
        (NAME(ADM00002.SPILL.FILE)   -
         NONINDEXED                  -
         REUSE                       -
         RECORDSIZE(1000 2000)       -
         RECORDS(10 10)              -
         VOL(PAC371))                -
         DATA                        -
        (NAME(ADM00002.SPILL.DATA))
/*
/&
* $$ EOJ
```

You must decide how you want to use spill files. Either one spill file can be deleted and defined in each print job (as shown above) or several can be defined before a print job is run.

If you define several spill files before the print job is run, use the NOALLOC option in the define statement to save space. Spill files that have not been emptied correctly (as a result of a previous job ending uncleanly) should be erased periodically.

Sample JCL for GDDM under CICS/OS/VS using PL/I

```
//********************** CICS/OS/VS PL/I *********************************
//*
//* Sample JCL to translate, compile, and link-edit a GDDM/CICS/VS
//* sample program or user-written application.
//*
//* This JCL assumes the use of the CICS-supplied
//* cataloged procedure "DFHEITPL".
//*
//***********************************************************************
//*
//jobname    JOB   accounting info,..........
//           EXEC  PROC=DFHEITPL
//*
//* Translation step
//*
//TRN.SYSIN  DD    *
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
//*
//*
//* Compilation step
//*
//* Override SYSLIB to reference library containing GDDM sample
//* PL/I declarations, as shown.
//* Add SYSLIB DD override statements to reference any additional user
//* libraries required, for example libraries containing GDDM-IMD ADSs,
//* as shown.
//*
//PLI.SYSLIB DD
//           DD   DSN=GDDM.INST.GDDMSAM,DISP=SHR
//           DD   DSN=user.gddm.ads-lib,DISP=SHR
//*
//* Link-edit step
//*
//* Insert INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNC unchanged  if using the nonreentrant interface
//* replace ADMASNC by ADMASRC if using the reentrant interface
//*           or    by ADMASPC if using the system programmer interface
//*
//LKED.INCLIB DD  DSN=GDDM.INST.GDDMLOAD,DISP=SHR
//LKED.SYSIN DD *
 INCLUDE INCLIB(ADMASNC)
 NAME xxxxxxxx(R)        Sample Program or Application Name
//*
```

Sample JCL for GDDM under CICS/OS/VS using COBOL

```
//********************* CICS/OS/VS COBOL ********************************
//*
//* Sample JCL to translate, compile, and link-edit a GDDM/CICS
//* sample program or user-written application.
//*
//* This JCL assumes the use of the CICS-supplied
//* cataloged procedure "DFHEITCL".
//*
//**********************************************************************
//*
//jobname    JOB   accounting info,..........
//           EXEC  PROC=DFHEITCL,PARM.COB='as-required-by-CICS'
//*
//* Translation step
//*
//TRN.SYSIN  DD    *
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
//*
//*
//* Compilation step
//*
//* Add SYSLIB DD override statements to reference any additional user
//* libraries required, for example libraries containing GDDM-IMD ADSs,
//* as shown.
//*
//COB.SYSLIB DD
//           DD   DSN=user.gddm.ads-lib,DISP=SHR
//*
//* Link-edit step
//*
//* Insert INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNC unchanged  if using the nonreentrant interface
//* replace ADMASNC by ADMASRC if using the reentrant interface
//*           or    by ADMASPC if using the system programmer interface
//*
//LKED.INCLIB DD  DSN=GDDM.INST.GDDMLOAD,DISP=SHR
//LKED.SYSIN DD *
 INCLUDE INCLIB(ADMASNC)
 NAME xxxxxxxx(R)       Sample Program or Application Name
//*
```

Sample JCL for GDDM under CICS/OS/VS using Assembler

```
//********************* CICS/OS/VS ASSEMBLER ***************************
//*
//* Sample JCL to translate, compile, and link-edit a GDDM/CICS
//* sample program or user-written application.
//*
//* This JCL assumes the use of the CICS-supplied
//* cataloged procedure "DFHEITAL".
//*
//**********************************************************************
//*
//jobname    JOB   accounting info,..........
//           EXEC  PROC=DFHEITAL
//*
//* Translation step
//*
//TRN.SYSIN  DD    *
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
//*
//*
//* Compilation step
//*
//* Add SYSLIB DD override statements to reference any additional user
//* libraries required, for example libraries containing GDDM-IMD ADSs,
//* as shown.
//*
//ASM.SYSLIB DD
//           DD
//           DD   DSN=user.gddm.ads-lib,DISP=SHR
//*
//* Link-edit step
//*
//* Insert INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNC unchanged  if using the nonreentrant interface
//* replace ADMASNC by ADMASRC if using the reentrant interface
//*            or   by ADMASPC if using the system programmer interface
//*
//LKED.INCLIB DD  DSN=GDDM.INST.GDDMLOAD,DISP=SHR
//LKED.SYSIN DD *
 INCLUDE INCLIB(ADMASNC)
 NAME xxxxxxxx(R)       Sample Program or Application Name
//*
```

Sample JCL for GDDM under CICS/DOS/VS using PL/I

```
********************* CICS/DOS/VS PL/I ******************************
*
* Sample JCL to translate, compile, and link-edit a GDDM/CICS
* sample program or user-written application.
*
* This JCL assumes that DLBL, EXTENT, and LIBDEF statements have
* already been used to:
*   - Define the GDDM sample source statement libraries
*   - Define the GDDM relocatable libraries
*
* Add additional statements to define any additional user source
* statement libraries required (for example, libraries containing
* GDDM-IMD ADSs).
*
*********************************************************************
*
// JOB     jobname
// DLBL    IJSYSPH,'PL/I.TRANSLATION',yy/ddd
// EXTENT  SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC    DFHEPP1$
*PROCESS   INCLUDE;
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
/*
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'PL/I.TRANSLATION',yy/ddd
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION  CATAL
   PHASE   phase-name,*
   INCLUDE DFHPL1I
*
* In the following INCLUDE statement,
* leave   ADMASNB unchanged  for GDDM using nonreentrant interface
* replace ADMASNB by ADMASRB for GDDM using reentrant interface
*            or   by ADMASNO for GDDM + PGF using nonreentrant interface
*            or   by ADMASRO for GDDM + PGF using reentrant interface
*            or   by ADMASPC if using the system programmer interface
*
   INCLUDE ADMASNB
   INCLUDE ADMASLC
// EXEC    PLIOPT
// EXEC    LNKEDT
/&
// JOB     RESET
CLOSE      SYSIPT,SYSRDR
/&
```

Sample JCL for GDDM under CICS/DOS/VS using COBOL

```
********************* CICS/DOS/VS COBOL ******************************
*
* Sample JCL to translate, compile, and link-edit a GDDM/CICS
* sample program or user-written application.
*
* This JCL assumes that DLBL, EXTENT, and LIBDEF statements have
* already been used to:
*   - Define the GDDM sample source statement libraries
*   - Define the GDDM relocatable libraries
*
* Add additional statements to define any additional user source
* statement libraries required (for example, libraries containing
* GDDM-IMD ADSs).
*
**********************************************************************
*
// JOB     jobname
// DLBL    IJSYSPH,'COBOL.TRANSLATION',yy/ddd
// EXTENT  SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC    DFHECP1$
 CBL LIB
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
/*
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'COBOL.TRANSLATION',yy/ddd
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION  SYM,ERRS,NODECK,CATAL
   PHASE   phase-name,*
   INCLUDE DFHECI
*
* In the following INCLUDE statement,
* leave   ADMASNB unchanged  for GDDM using nonreentrant interface
* replace ADMASNB by ADMASRB for GDDM using reentrant interface
*           or   by ADMASNO for GDDM + PGF using nonreentrant interface
*           or   by ADMASRO for GDDM + PGF using reentrant interface
*           or   by ADMASPC if using the system programmer interface
*
   INCLUDE ADMASNB
   INCLUDE ADMASLC
// EXEC    FCOBOL
// EXEC    LNKEDT
/&
// JOB     RESET
CLOSE      SYSIPT,SYSRDR
/&
```

Sample JCL for GDDM under CICS/DOS/VS using Assembler

```
********************* CICS/DOS/VS ASSEMBLER **************************
*
* Sample JCL to translate, compile, and link-edit a GDDM/CICS
* sample program or user-written application.
*
* This JCL assumes that DLBL, EXTENT, and LIBDEF statements have
* already been used to:
*   - Define the GDDM sample source statement libraries
*   - Define the GDDM relocatable libraries
*
* Add additional statements to define any additional user source
* statement libraries required (for example, libraries containing
* GDDM-IMD ADSs).
*
**********************************************************************
*
// JOB     jobname
// DLBL    IJSYSPH,'ASM.TRANSLATION',yy/ddd
// EXTENT  SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC    DFHEAP1$
      . . . .
      . . . .
 Source deck here.
 Remember to define ADMUOFF if the program uses the nonreentrant
 interface.  (See "Using the Nonreentrant Interface of GDDM", on page 8.)
      . . . .
      . . . .
/*
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'ASM.TRANSLATION',yy/ddd
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION  SYM,ERRS,NODECK,CATAL
   PHASE   phase-name,*
   INCLUDE DFHEAI
*
* In the following INCLUDE statement,
* leave   ADMASNB unchanged  for GDDM using nonreentrant interface
* replace ADMASNB by ADMASRB for GDDM using reentrant interface
*          or   by ADMASNO for GDDM + PGF using nonreentrant interface
*          or   by ADMASRO for GDDM + PGF using reentrant interface
*          or   by ADMASPC if using the system programmer interface
*
   INCLUDE ADMASNB
   INCLUDE ADMASLC
// EXEC    ASSEMBLY
// EXEC    LNKEDT
/&
// JOB     RESET
CLOSE      SYSIPT,SYSRDR
/&
```

Chapter 3. Using GDDM under IMS/VS

This chapter describes the use of GDDM under the IMS/VS operating system. It covers the following topics:

- Restrictions on the use of GDDM under IMS/VS
- Application program structure
- Link-editing a GDDM application
- Using the system programmer interface with dynamic load
- PSBs for GDDM applications
- Data sets and file processing
- The IMS/VS default error exit
- GDDM and MFS
- GDDM DL/I interface
- IMS/VS considerations for GDDM utilities
- GDDM object import/export utility
- Sample JCL.

The use of the IMS version of the GDDM print utility is described in Chapter 7, "The GDDM print utilities" on page 47.

Application programs for IMS should carefully follow the instruction given under "Application program structure" on page 24. Careful note of the restrictions should also be taken. The IMS/VS samples in Appendix K, "Sample programs" on page 249 can be used as a model for application programs.

Two utilities are provided to assist in the use of GDDM under IMS/VS:

- The data-base utility used when installing GDDM and when the network is updated.
- The Import/Export utility that allows symbol sets, saved pictures, and other GDDM objects to be moved out of, and into, an IMS/GDDM system.

The description in this chapter assumes a working knowledge of IMS/VS.

Restrictions on the use of GDDM under IMS/VS

The main restrictions on the use of GDDM in an IMS/VS environment are:

- The IPDS printers are not supported.
- Picture interchange format (PIF) files are not supported.
- GDDM-IMD is not supported.
- GDDM-PCLK 1.1 is not supported.
- The 5080 Graphics System is not supported.
- GDDM only supports system network architecture (SNA) connection for 3179-G and 3192-G display stations, 3270-PC/G and 3270-PC/GX work stations, and 5550-family work stations.
- For 327x displays the amount of data that can be created by GDDM and successfully transmitted by IMS/VS depends on the line protocol and access method used to send this data to the terminal.

  For terminals defined as SLUTYPE2, or remote 3270 devices specified with data transparency, OPTIONS=XPAR, there are no restrictions.

  For all other 3270 displays the amount of data that may be created and sent by GDDM in one message is controlled by the OUTBUF parameter specified during system definition.

  For very complex pictures the length of the data streams generated by GDDM may exceed this maximum value. In such cases, the output message is rejected by IMS/VS and an IMS/VS error message is displayed at the terminal. If this occurs and the device token being used specifies COMPRES=NO, one way of reducing the length of the data stream is to use a different device token (one that has COMPRES=YES) that allows data-stream compression (assuming that the 3274 control unit is configured for PS compression). For more details, see Appendix G, "Device characteristics tokens" on page 203.
- For 3270-family terminals and printers output may only be sent to logical terminals that are defined in the GDDM System Definition data base. This contains information that describes the physical characteristics of the device.

  The information in the data base is located using the LTERM name of a message queue as a key rather than the physical terminal name, because only that piece of information is available to the application and thus GDDM. To prevent transmission errors the device to which the LTERM is assigned must have the characteristics identified in the data base. Reassignment of LTERMS must be reflected by changes to the data base.
- GDDM cannot be used to process input from the terminals. The use of message queues and the scheduling algorithms of an IMS/VS system are unsuited to the direct interaction allowed in other subsystems.

  Information on the interaction of GDDM and the message format service (MFS) and a description of how input from a display formatted by GDDM should be processed, is given on page 26.
- FSSAVE files generated under IMS/VS cannot be used under another subsystem, such as TSO, nor may such files created under other subsystems be sent to a device attached to IMS/VS using the FSSHOW functions.
- For the interactive utilities only, the use of PF key 12 allocated by IMS/VS to the COPY function should be avoided. If the keyboard has only 12 PF keys, the IMS/VS system definition for the terminal should specify NOCOPY.
- Plotters attached to 3179-G or 3192-G display stations or to 3270-PC/G or 3270-PC/GX work stations are not supported under IMS/VS.

- The WINDOW processing option and operator window functions are not supported under IMS/VS.
- ICU flat-file data import is not supported under IMS/VS.

Application program structure

The following list contains the steps that an IMS/VS transaction program might make when using GDDM.

1. Issue a GU call to the I/O program communication block (PCB) to acquire the first segment of the input message.
2. Issue FSINIT, or any of its aliases, to enable GDDM processing.
3. Optionally issue an FSEXIT call to nominate a user-provided error exit to replace the default exit provided with GDDM, or to raise the threshold of errors below which errors are not reported.
4. Issue one or more ESPCB calls to identify to GDDM the PCBs that it may use.
5. Issue one or more ESLIB calls to show which data bases are to be searched when retrieving and storing GDDM data.
6. If the I/O PCB has not been identified by an ESPCB call above, or if output is to go to a destination other than that of the I/O PCB, issue DSOPEN calls to define to GDDM the possible output destinations.

   If the PCB to be used by GDDM is modifiable, the destination of the PCB must be set using the CHNG call before the DSOPEN call is issued.

   This step is not needed if output is to go to the source of the input message and the I/O PCB has been identified to GDDM because this is the default destination and PCB used by GDDM.
7. Process the input message using GN calls to acquire subsequent message input. Generate output messages using the GDDM subroutines to describe any field-formatted or graphics output. Use the DSUSE statement to select the output destination if devices have been explicitly defined by DSOPEN.
8. Issue DSCLS statements for each device opened using DSOPEN.
9. Issue the FSTERM call to end GDDM processing.
10. Repeat from step 1 to process any more input messages.

This arrangement of an application program ensures that GDDM is inactive across a GU call that may reset certain information used by GDDM. Its drawback is the repeated initialization and termination of GDDM. An alternative structure that avoids this overhead is shown below. Care should be taken to ensure that all devices are closed across the GU call.

1. Issue FSINIT, or any of its aliases, to enable GDDM processing.
2. Optionally issue an FSEXIT call to nominate a user-provided error exit to replace the default exit provided with GDDM, or to raise the threshold of errors below which errors are not reported.
3. Issue one or more ESPCB calls to identify to GDDM the PCBs that it may use.
4. Issue one or more ESLIB calls to show which data bases are to be searched when retrieving and storing GDDM data.
5. Issue a GU call to the I/O PCB to acquire the first segment of the input message.
6. If the I/O PCB has not been identified by an ESPCB call above, or if output is to go to a destination other than that of the I/O PCB, issue DSOPEN calls to define to GDDM the possible output destinations.

   If the PCB to be used by GDDM is modifiable, the destination of the PCB must be set using the CHNG call before the DSOPEN call is issued.

   This step is not needed if output is to go to the source of the input message and the I/O PCB has been identified to GDDM because this is the default destination and PCB used by GDDM.
7. Process the input message using GN calls to acquire subsequent message input. Generate output messages using the GDDM subroutines to describe any field-formatted or graphics output. Use the DSUSE statement to select the output destination if devices have been explicitly defined by DSOPEN.
8. Issue DSCLS statements for each device opened using DSOPEN.

   If the default destination was used, GDDM automatically opens a device with an identifier of 0. This should be closed using a statement of the form

   ```
   CALL DSCLS(0,1)
   ```
9. Repeat from step 5 to process any more input messages.
10. Issue the FSTERM call to end GDDM processing when all input messages have been processed.

Link-editing a GDDM application program

Examples of the JCL that can be used to compile and link-edit application programs written in PL/I or COBOL are listed on pages 29 and 30.

Unless an application program uses dynamic load facilities to access GDDM through the system programmer interface (see below), a GDDM application program must be link-edited with the appropriate GDDM interface module as well as the DL/I interface module. The interface module used depends on the type of GDDM interface used and the language of the application program, or, to be precise, of the program specification block (PSB) for the transaction.

The module to be used may be explicitly controlled by linkage editor control statements, or one of the alternative versions of the initialization entry point can be used. The latter causes the correct GDDM interface modules to be loaded by the automatic library call capability of the linkage editor.

Table 2. GDDM data-set characteristics for IMS/VS

Type of Data	GDDM default filename	Data set type	DCB characteristics		
			Record format (RECFM)	Record length (LRECL)	Block size (BLKSIZE)
Symbol sets	ADMTRACE	Sequential data sets or SYSOUT classes	VA	≥125	LRECL
			VBA	≥125	≥LRECL + 4

There are four alternative initialization calls for GDDM in an IMS/VS environment. They allow for a choice of nonreentrant and reentrant interface and non-PL/I and PL/I PSBs. The names of the initialization calls are as follows:

Interface	Non-PL/I PSB	PL/I PSB
Nonreentrant	FSINNI	FSINNPI
Reentrant	FSINRI	FSINRPI

If direct control of the link-edit process is chosen, the initialization call should be coded using the FSINIT (or SPINIT) entry point, and the following modules explicitly included by the link-edit process:

Interface	Non-PL/I PSB	PL/I PSB
Nonreentrant	ADMASNI	ADMASNJ
Reentrant	ADMASRI	ADMASRJ
System Programmer	ADMASPI	ADMASPJ

Using the system programmer interface with dynamic load

If an application program uses only the system programmer interface (SPI), all invocations of GDDM are through the entry point ADMASP. This entry point can be resolved by link-editing the application program with one of the GDDM interface modules, ADMASPI or ADMASPJ, as described above.

However, the application program can avoid these linkage-edit considerations by using system facilities (the OS LOAD function) to dynamically load a GDDM interface module (ADMASPLI for non-PL/I PSBs or ADMASPLJ for PL/I PSBs). The main entry points for these modules are defined both with their load module names and with the name ADMASP.

Program specification blocks for GDDM applications

The PSB for a GDDM application must include the PCBs required by GDDM. These are:

- One TP PCB for each concurrently active device (for example, for which a DSOPEN call was issued).

  For family-1 and family-3 (3270-family and system printer) devices, the LTERM quoted in the PCB statement must be that of the terminal to which the output is to be sent. For family-2 devices, the NAME parameter should specify the transaction code assigned to the GDDM print utility.

  If the NAME or LTERM parameter is not supplied on the PCB statement, the PCB should be defined as modifiable and the application program should issue a CHNG call to set the destination before defining the PCB to GDDM.

- A DB PCB for the system definition data base if GDDM output is to be generated. A PROCOPT of G should be specified because no normal GDDM operation can alter information in this data base.

  A sample PCB statement for such a data base is:

  ```
  PCB    TYPE=DB,NAME=ADMSYSDF,PROCOPT=G,KEYLEN=8
  SENSEG NAME=ADMSDSGM,PARENT=0
  ```

  Ensure that the names used in the above sample were not altered during the initialization process. If they were, corresponding changes must be made in the IMSSDBD and IMSSEGS options in GDDM's external defaults, as described under "GDDM external defaults – IMS/VS" on page 131.

- A DB PCB for each object data base required.

  A sample PCB statement for such a data base is:

  ```
  PCB    TYPE=DB,NAME=ADMOBJ1,PROCOPT=G,KEYLEN=20
  SENSEG NAME=ADMOBROO,PARENT=0
  SENSEG NAME=ADMOBDEP,PARENT=ADMOBROO
  ```

  A PROCOPT of A should be specified if the program is to alter information in the data base using GDDM calls. Note the restriction that information is written only to the first of the data bases quoted in the ESLIB parameter list for any given type of object.

  It is possible to vary the DBD and segment names from those quoted above during IMS system generation. If they are changed, corresponding changes must be made in the OBJFILE and IMSSEGS options in GDDM's external defaults, as described under "GDDM external defaults – IMS/VS" on page 131.

  However, if only the data-base name is to be altered, the ESLIB statement can be used to notify GDDM of the data-base name rather than altering the external defaults. The name in the external defaults is only used to find the data base to search for objects if no ESLIB statement is coded.

An ESPCB call should be coded in the application for each PCB to be used by GDDM.

Data sets and file processing

When running under IMS/VS, GDDM uses two types of file processing:

- QSAM (Queued Sequential Access Method) is used to write data to sequential output destinations when certain trace functions are requested using the FSTRCE call. For more details, see the *GDDM Diagnosis and Problem Determination Guide*.
- DL/I is used to read and write information into the two types of DL/I data base used by GDDM.

In the first type, GDDM refers to the file using a ddname. The default value of this name is taken from the IMSTRCE option in GDDM's external defaults. (For details, see "GDDM external defaults – IMS/VS" on page 131). If output is to be created from this file, the dependent region JCL must be modified to include a DD statement for it. The data set type and DCB characteristics should be as shown in Table 2 on page 25.

The structure and requirements of the DL/I data bases used by GDDM are described in the *GDDM Installation and System Management for VSE* manual.

The Interactive Chart Utility (part of GDDM-PGF) includes a directory function that supports list, delete, and copy operations on GDDM DL/I objects such as symbol sets and pictures.

The IMS/VS default error exit

GDDM provides a default error exit, which is given control when GDDM detects an error in its processing. The user can control the severity level of an error that causes the exit to be taken and may also identify a user-written error exit, as described for FSEXIT in the *GDDM Base Programming Reference, Volume 1*.

The default error exit provided in the IMS/VS environment reports the error using a /BROADCAST command directed to the LTERM named in the I/O PCB. The transaction must, therefore, be authorized to issue this command. If the I/O PCB was not identified to GDDM by the ESPCB call, or the CMD call fails, the error message is issued using a "write to operator" (WTO) function. The route code and message descriptor for this WTO function are contained in GDDM's external defaults. The IMSWTOR and IMSWTOD options can be changed to suit the installation. For details of how to do this, see Chapter 1, "Customizing your program and its environment" on page 1.

GDDM and the Message Format Service

GDDM uses the Message Format Service (MFS) BYPASS function to send output to 3270 displays and to non-SCS printers. Output to SCS printers is sent using Basic Edit.

For displays, each message created by GDDM contains the information needed to format the screen. By default, it is sent using a Message Output Descriptor (MOD) with the name DFS.EDT (for a user application) or DFS.EDTN (for a GDDM or GDDM-PGF interactive utility). When a message using one of these MODs is detected by MFS, it does not format the information in the message but instead assumes that it contains a data stream that may be sent to the device without more processing.

Any input subsequently received from the device for a user application is not processed against a Message Input Descriptor (MID) but is instead passed to the Basic Edit process. This removes the device-dependent control information from the data stream and replaces it with blanks.

Using GDDM it is possible to create a message containing a picture and one or more input fields. When this has been displayed, the end user can enter the next transaction request from the terminal by typing into the input field and pressing the ENTER key.

The segment returned from the GU DL/I function call in the application program contains the contents of the fields modified by the end user in a single segment. There is no indication of the key (PF, ENTER, or PA) that caused the data to be sent to IMS/VS. The fields are of variable length, separated from each other by one or more blanks.

For more information on the detailed formatting of the input data stream, see the description of the Message Format Service in the IMS/VS reference manuals; see the Bibliography in Volume 1.

An installation can provide its own MOD to be used by GDDM for transmitting nonconversational messages from a user application to 3270-family devices. In this way, an installation can make special provision for processing subsequent input messages. To cause GDDM to use a MOD name other than DFS.EDT, the alternative MOD name must be specified in the IMSMODN option in GDDM's external defaults, as described under "GDDM external defaults – IMS/VS" on page 131.

GDDM DL/I interface

The GDDM routines use the same DL/I interface as a standard application program. To do so, GDDM needs to know which of the PCBs, passed to the application when it is scheduled, are to be used by GDDM. This information is passed to GDDM by the ESPCB subroutine call. The syntax of this function is described in detail in the *GDDM Base Programming Reference, Volume 1*.

Using this function, the application program can identify the I/O PCB, other TP PCBs, and DB PCBs. The use GDDM makes of each of these types of PCB is described in the next sections. The following general rules apply to the sharing of PCBs between an application and GDDM:

1. GDDM uses the TP PCBs to insert the data streams that it generates to the message queues. Such a PCB is considered to be in use between the times that the GDDM device services calls DSOPEN and DSCLS are issued. These calls are described in more detail in the *GDDM Base Programming Reference, Volume 1*. While a PCB is in use, the application program must not also insert data on the queue through the same PCB nor must it cause

the data on the PCB to be enqueued by issuing a GU to the I/O PCB or any other action that causes a checkpoint.

2. If an application program tries to send output when no primary device was explicitly defined, GDDM tries to open a device to use the I/O PCB.
3. If the application needs to insert another message to the message queue, using a PCB that was used by GDDM, the first segment of the message must be inserted using the DL/I PURG function to enqueue any message created by GDDM. GDDM itself inserts the first message segment, using this function to enqueue any application output already placed on the message queue before a device is opened.

Use of message queues

GDDM uses the I/O and TP PCBs to insert output to message queues for the primary and alternate devices. These devices can be 3270-family devices, queued printer devices, or system printer devices.

The PCB used by any device depends on the way in which the device was identified using the DSOPEN function and on the type of device. The method used by GDDM to select the PCB to be used is given below.

Each message is created by inserting one or more segments. The number of segments is dependent on the complexity of the output. For system printer devices, each output segment is a print record. For the other types of device, the message is segmented at arbitrary points in the generated output. In this latter case, the maximum size of the output segment is 84 bytes for a queued printer device, and is taken from the value of the IOBFSZ option in the current GDDM external defaults for a 3270-family device.

3270-family devices

The NAME parameter on DSOPEN supplies the name of the LTERM to which output is to be sent. GDDM selects the PCB to be used by checking first the I/O PCB and then each of the TP PCBs, in the order in which they were identified by ESPCB calls, for a destination of the given LTERM. It uses the first one of these PCBs that is not already in use for another device.

If the NAME parameter is omitted, or coded as "★", GDDM tries to use only the I/O PCB.

If no PCB with a matching name is found, or if all PCBs checked are already in use, the DSOPEN function fails.

The number of messages generated by GDDM for this family of device is dependent on the type of the target terminal. If it is a display, the output created from each FSFRCE or ASREAD call is sent as an individual message. If the terminal is a printer, all output created by the application program using the GDDM device is sent in a single message.

If the application is conversational and the I/O, or another PCB, is selected by GDDM for use with a display device, the application may only issue the FSFRCE or ASREAD call once because, in this situation, GDDM cannot issue the DL/I PURG request required to cause the message created by the first call to be enqueued.

Queued printer devices

These devices generate output that is sent to the GDDM-provided Print Utility for subsequent transmission to a real 3270-family terminal. The NAME parameter specified on DSOPEN identifies the LTERM name of the latter terminal and cannot be omitted. The output generated by GDDM directly from the application program is inserted to the first PCB in which the LTERM name is the transaction code of the GDDM print utility. The default value for this transaction name is ADMPRINT, but the installation may change this by altering the IMSPRNT option in the current GDDM external defaults, as described under "GDDM external defaults – IMS/VS" on page 131. If no such PCB can be found, or if all such PCBs are already being used by other GDDM devices, the DSOPEN function fails.

All the output created by GDDM between DSOPEN and DSCLS for a device of this type is sent as a single IMS/VS message.

System printer devices

The NAME parameter specified on DSOPEN should identify an LTERM to which print records, including carriage control characters, can be sent. If omitted, a default destination is assumed by GDDM. This is ADMLIST, but the installation may change the value by altering the IMSSYSP option in the current GDDM external defaults, as described under "GDDM external defaults – IMS/VS" on page 131.

The PCB to be used is again chosen by checking first the I/O PCB, and then all TP PCBs, in the order identified by the application, for an LTERM name matching that given or assumed on the DSOPEN call. If no match is found, or if all matching PCBs are already in use, the DSOPEN function fails.

All the output created by GDDM for any one device of this type forms a single IMS/VS message.

Use of data bases

GDDM uses two types of data base: one to contain the terminal characteristics information, and another to contain the "objects", such as symbol sets, saved pictures, generated mapgroups, and ADMGDF files. The DB PCBs that are to be used must be identified to GDDM by the ESPCB call before executing any routine that might require access to the data bases.

The use of the data bases containing objects is further controlled by the ESLIB routine whose syntax is described in the *GDDM Base Programming Reference, Volume 1*. This routine establishes the set of data bases that are to be used to store or retrieve a given type of object. The data bases to be used are identified to this routine as a list of DBD names. Before issuing this call the user must have issued ESPCB calls that referred to DB PCBs for all the data bases mentioned on the ESLIB call.

The data bases are searched in the order given in an attempt to find an object. An object is stored only in the first data base of the list, even though it may have been retrieved from another one.

The DBD name of the system definition data base is taken from the value in the IMSSDBD option in the current GDDM external defaults; see "GDDM external defaults – IMS/VS" on page 131. The external defaults also contain default DBD names for the data bases to be used for each of the object types.

IMS/VS considerations for GDDM utilities

Under IMS/VS, the GDDM and GDDM-PGF interactive utilities are run under the control of a single transaction that emulates the environment that they expect. The transaction is a "wait for input" conversational transaction. In these notes, the transaction code for the utility is assumed to be "ADM," but this may have been changed by the installation.

- The transaction can support only a predefined number of concurrent transactions. Any attempt to start a new session with a utility that would cause the limit to be exceeded is rejected with message ADM0772.

  The number of concurrent transactions allowed may be altered by modifying the value in the IMSUMAX option in the current GDDM external defaults. For details, see "GDDM external defaults – IMS/VS" on page 131.

- The transaction cannot continue conversations if, for any reason, it is rescheduled during the lifetime of a conversation. Such conversations are terminated with message ADM0774.

- A particular scheduling of the transaction usually ends when it has no record of any existing conversations. Because it is possible for a conversation to be terminated without the transaction's being aware of the fact (for example, because of particular error conditions), the transaction may not be completed even though the end user has terminated the conversation. In such a case, the end user should enter the request:

  ```
  ADM EXIT
  ```

  which causes the utility to note that all conversations against the LTERM, from which the request originates, were terminated.

- To force a return to the region controller by the transaction irrespective of the current state of any active conversations, the request:

  ```
  ADM SHUTDOWN
  ```

  can be entered from an authorized terminal. By default this authorized terminal has an LTERM name of MASTER.

The keywords EXIT and SHUTDOWN, and the LTERM name of the terminal authorized to issue the latter request, are as defined in the IMSEXIT, IMSSHUT, and IMSMAST options in the current GDDM external defaults. For details, see "GDDM external defaults – IMS/VS" on page 131.

- If, during a session with a utility, the current screen format is destroyed (for example, by a high priority or error message), it can be restored by entering two blank characters as the next input message.

- On some terminals, IMS/VS reserves Program Function key 12 for use as a print request key and does not pass this as a valid interrupt to the utility transaction. If the terminal has 24 rather than 12 PF keys, the use of PF key 12 can be avoided because PF 24 usually has the same function.

  If only 12 PF keys are available, the IMS/VS system definition for a terminal should specify NOCOPY if the GDDM utilities are to be accessed from that terminal.

GDDM object import/export utility

The GDDM object import/export utility is used to transfer GDDM objects (generated mapgroups from GDDM-IMD, ADMGDF objects, symbol sets, chart formats or data, or FSSAVE objects) between partitioned data set(s), and the data base in which they are kept for IMS/VS use, or to delete them from the data base.

Its purpose is to enable objects to be transferred between GDDM applications running on one IMS/VS system, and those running on either another IMS/VS system, or in a totally different environment (for example a TSO development system).

The operation and use of the utility are described in the *GDDM Installation and System Management for MVS* manual.

Sample JCL for GDDM under IMS/VS using PL/I

```
//********************* IMS/VS PL/I ************************************
//*
//* Sample JCL to compile, and link-edit a GDDM/IMS
//* sample program or user-written application.
//*
//* This JCL assumes the use of the IMS/VS-supplied
//* cataloged procedure "IMSPLI".
//*
//* The IMS/GDDM sample program or user-written application is
//* placed in IMSVS.PGMLIB.
//*
//* xxxxxxxx is the name under which the program load module is
//* generated.
//*
//**********************************************************************
//*
//jobname    JOB   accounting info,..........
//           EXEC  PROC=IMSPLI,MBR=xxxxxxxx,REGION.C=512K,
//             PARM.C='XREF,A,OBJ,NODECK,INC,OPT(TIME)'
//*
//* Compilation step
//*
//* Insert SYSLIB to reference library containing GDDM sample
//* PL/I declarations, as shown.
//*
//C.SYSLIB   DD   DSN=GDDM.INST.GDDMSAM,DISP=SHR
//C.SYSIN    DD   *
      . . . .
      . . . .
 Source deck here.
      . . . .
      . . . .
/*
//*
//* Link-edit step
//*
//* Insert INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNJ unchanged  if using the nonreentrant interface
//* replace ADMASNJ by ADMASRJ if using the reentrant interface
//*            or   by ADMASPJ if using the system programmer interface
//*
//L.INCLIB   DD   DSN=GDDM.INST.GDDMLOAD,DISP=SHR
//L.SYSIN    DD   *
 INCLUDE INCLIB(ADMASNJ)
/*
```

Sample JCL for GDDM under IMS/VS using COBOL

```
//********************* IMS/VS COBOL ***********************************
//*
//* Sample JCL to compile, and link-edit a GDDM/IMS
//* sample program or user-written application.
//*
//* This JCL assumes the use of the IMS/VS-supplied
//* cataloged procedure "IMSCOBOL".
//*
//* The IMS/GDDM sample program or user-written application
//* is placed in IMSVS.PGMLIB.
//*
//* xxxxxxxx is the name under which the program load module is
//* generated.
//*
//**********************************************************************
//*
//jobname    JOB   accounting info,..........
//*
//           EXEC  PROC=IMSCOBOL,MBR=xxxxxxxx
//*
//* Compilation step
//*
//C.SYSIN    DD    *
      . . . .
      . . . .
 Source deck here.
      . . . .
      . . . .
/*
//*
//* Link-edit step
//*
//* Insert INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNI unchanged  if using the nonreentrant interface
//* replace ADMASNI by ADMASRI if using the reentrant interface
//*            or   by ADMASPI if using the system programmer interface
//*
//L.INCLIB   DD   DSN=GDDM.INST.GDDMLOAD,DISP=SHR
//L.SYSIN    DD   *
 INCLUDE INCLIB(ADMASNI)
/*
```

Chapter 4. Using GDDM under MVS/XA

This chapter describes some special programming considerations for 31-bit mode GDDM applications, and provides general information on GDDM code and application programs (for CICS/OS/VS, IMS/VS, and TSO) that can run under the MVS/XA operating system. It also discusses object compatibility between System/370 and System 370/XA environments.

GDDM code above 16 megabytes

Under suitable subsystems and operating systems, the main body of GDDM code can reside above 16 megabytes. This is the default state.

Application code above 16 megabytes

Under suitable releases of TSO and CICS/OS/VS, GDDM applications can reside above 16 megabytes.

IMS/VS applications cannot be run above 16 megabytes because the DL/I stub (ASMTDLI) is link-edited with the application. GDDM presumes that the DL/I stub always runs in 24-bit mode.

AMODE(31) applications and application parameters above 16 megabytes

Under TSO, CICS/VS, and IMS/VS, applications can run in 31-bit mode and, if so, can pass to GDDM parameters that are located above 16 megabytes.

If GDDM is called in 31-bit mode, it assumes that any parameter addresses that are passed represent 31-bit addresses.

GDDM object compatibility between System/370 and System 370/XA

GDDM is object-compatible between System/370 and System 370/XA, although it may contain instructions that are unique to System 370/XA, which are run only if the operating system is MVS/XA.

MVS/XA terminology

For a full definition of MVS/XA terminology, refer to the associated MVS/XA documentation; see the Bibliography in Volume 1. The following section gives a short explanation of some of the relevant keywords.

AMODE(24), AMODE(31), AMODE(ANY)
This indicates that a module may be called in 24-bit addressing mode only, in 31-bit addressing mode only, or in either mode, respectively.

RMODE(24), RMODE(ANY)
This indicates that a module may be loaded only in 24-bit addressable storage (below 16 megabytes), or anywhere in storage, respectively.

TRUE
Interfaces categorized as TRUE allow a program running in 31-bit mode to use the interface and to pass 31-bit parameter addresses with values greater than 16 megabytes.

RESTRICTED
Interfaces categorized as RESTRICTED can only be called by programs running in 24-bit mode.

HOLLOW
Interfaces categorized as HOLLOW can be called by programs running in 24-bit mode or 31-bit mode, but the value of all address parameters must be less than 16 megabytes.

EITHER
Interfaces categorized as EITHER can be called by programs running in 24-bit mode or 31-bit mode, but restrictions exist with respect to parameters.

The GDDM Application Interface is TRUE under TSO and CICS/VS.

Subsystem-independent routines

All GDDM subsystem-independent routines (including subsystem-independent adapter routines) are compiled with:

```
AMODE(ANY) / RMODE(ANY)
```

Under MVS/XA, the Linkage Editor changes the attributes of these routines to:

```
AMODE(31) / RMODE(ANY)
```

Provided all routines within a load module have RMODE(ANY), the linkage editor assigns RMODE(ANY) to the load module, thus allowing it to be located above 16 megabytes. Note, however, that if any routine within a load module has RMODE(24), the linkage editor assigns RMODE(24) to the load module, which is therefore constrained to reside below 16 megabytes.

When linked and called under an MVS/XA system, AMODE(31) / RMODE(ANY) routines are called in 31-bit mode, run entirely in 31-bit mode, and can reside anywhere in storage.

When called under a 370 system, these routines run in 24-bit mode and use no 370/XA-unique facilities.

CICS/VS-dependent routines

All CICS/VS services used by GDDM are TRUE services. Therefore, all GDDM CICS/VS-dependent routines are compiled with:

```
AMODE(ANY) / RMODE(ANY)
```

Link-edit and execution considerations are as for GDDM subsystem-independent routines (above).

Because the GDDM Application Interface routines have RMODE(ANY), CICS/VS applications with RMODE(ANY) may be link-edited to GDDM, and may be located above 16 megabytes.

Under CICS/VS, nearly all of GDDM can reside above 16 megabytes. The only exceptions are the Call Format Descriptor Module and the APL Request Codes Module. These modules have RMODE(24) to ensure addressability from 24-bit mode applications. The whole of an application program can reside above 16 megabytes.

IMS/VS-dependent routines and TSO-dependent routines

The system services and interfaces in the IMS/VS and TSO environments are of all types: TRUE, HOLLOW, EITHER, and RESTRICTED. For simplicity, GDDM treats most services that are not TRUE as RESTRICTED (requiring invocation in 24-bit mode from below 16 megabytes). The one exception to this rule is R-format GETMAIN and FREEMAIN used in the Application Interface routines. This form of GETMAIN and FREEMAIN is EITHER, and can therefore be invoked above 16 megabytes to acquire or release storage located below 16 megabytes. Using this, and not treating the macro as RESTRICTED allows GDDM application programs under TSO to be run above 16 megabytes.

GDDM IMS/VS-dependent routines and TSO-dependent routines that call only TRUE services are compiled with AMODE(ANY) / RMODE(ANY) and are treated in the same way as subsystem-independent routines.

GDDM IMS/VS-dependent routines and TSO-dependent routines that call services that are not TRUE are compiled with AMODE(ANY) / RMODE(24), and are therefore located below 16 megabytes.

The non-TRUE services that these routines contain are, in general, treated as RESTRICTED (although there are exceptions). For these services;

- Before invoking the service, the routines enter 24-bit mode (if not already in 24-bit mode).
- On return from the service, the routines restore 31-bit mode (if entered in 31-bit mode).

Therefore, in most cases, the setting of 24-bit mode is highly localized.

Application programming considerations

Under all MVS/XA systems, a GDDM application program may have any valid AMODE attribute, and may call GDDM in any mode (24-bit or 31-bit) consistent with its location. In fact, it is possible (though not recommended) for an application program to call GDDM in both 24-bit and 31-bit modes in the same session.

Under MVS/XA, the "normal" AMODE for a GDDM instance is AMODE(31). The vast majority of GDDM processing is performed in its "normal" AMODE, with AMODE(24) being forced only for the duration of system service calls that are not known to be able to tolerate being called in AMODE(31).

The Application Interface routines (that is, those parts of the Application Interface that are link-edited with the application program) always run in the AMODE of the application. However, they mode-switch to the GDDM "normal" AMODE when control is passed to the dynamic part of GDDM.

If the application program runs in AMODE(24), GDDM clears the top byte of each parameter address word, in a copy of the parameter list that it usually generates, to prevent wrong addresses being formed in the main code running in 31-bit mode.

The SPINIT call

The SPINIT call is a form of initialization that allows parameters to be passed by a SPIB (SPI Initialization Block). The SPIB contains a number of address words that can be set by an application program.

If the SPINIT call is actually issued in 24-bit mode, GDDM clears the top byte (minus the top bit) of each address word that it processes.

The FSEXIT call

A user error exit, whose address is passed on an FSEXIT call, is assumed to be executable in 31-bit mode if either:

1. The application call is in 31-bit mode, or
2. The top bit of the address passed on the FSEXIT call is on. (For example, the address uses the MVS/XA convention that the top bit of the address identifies its AMODE.)

The first condition enables a high-level language program to pass the address of an exit that is link-edited with itself. (It is difficult (or not possible) to set the top bit of an address in, for example, FORTRAN.)

If a 24-bit application uses a 31-bit user error exit (by setting the top bit of the address), it is the user exit's responsibility to return control to the application in the correct AMODE (because GDDM issues the equivalent of an XCTL command to the exit).

User exits

A number of other user exits can be defined as described under "Specifying user exits" on page 104. That information describes MVS/XA considerations for such exits.

Chapter 5. Using GDDM under TSO

This chapter describes the use of GDDM under the TSO operating system. It covers these topics:

- Link-editing a GDDM application program
- Data sets and file processing
- Display terminal processing
- Using APL terminals
- Using GDDM under TSO or MVS batch
- Sample JCL.

An application program using GDDM has no particular restrictions or requirements. However, if a PL/I program uses the GDDM-supplied declarations it must have access to the library on which they are held. Also, it must be link-edited with one of the interface modules as described below.

Terminal users should be aware of the GDDM usage of PA1, PA2, and the CLEAR keys. Also, there is a possibility of unexpected terminal responses after a GDDM application program has ended abnormally. These matters are described under "Display terminal processing" on page 36.

Link-editing a GDDM application program

An example of the JCL that can be used to link-edit GDDM application programs is listed on page 40.

Unless the application program uses dynamic load facilities to access GDDM by means of the system programmer interface (see below), an application program using GDDM under TSO must be link-edited with an appropriate GDDM interface module. This interface module can be specifically included in the link-edit process. Or, if the application program uses one of the other FSINIT entry points described in the *GDDM Base Programming Reference, Volume 1*, the required GDDM interface module can be included by linkage editor automatic library call facilities.

This is a list of the GDDM interface modules for TSO:

Interface	Interface module	FSINIT alternative entry
Nonreentrant	ADMASNT	FSINN
Reentrant	ADMASRT	FSINR
System programmer	ADMASPT	–

Using the system programmer interface by means of dynamic load

If an application program uses only the System Programmer Interface, all invocations of GDDM are through the entry point ADMASP. This entry point can be resolved by link-editing the application with the GDDM interface module ADMASPT, as described above.

Or, the application can avoid these linkage-edit considerations by using system facilities (the OS LOAD function) to load dynamically a GDDM interface module ADMASPLT. The main entry point for this module is defined with both names: ADMASP and ADMASPLT.

Data sets and file processing

When running under TSO, GDDM-Base and GDDM-PGF use three types of file processing:

- BPAM (Basic Partitioned Access Method) is used to read and write members on partitioned data sets.
- QSAM (Queued Sequential Access Method) is used to read and write data to and from sequential destinations, such as sequential data sets or suitable SYSOUT classes.
- BDAM (Basic Direct Access Method) is used to write data to direct access data sets, such as the Master Print Queue data set used to control queued printer devices.

GDDM-IMD uses additional types of file processing. For details, see the *GDDM Interactive Map Definition* manual.

BPAM file processing

BPAM is used by GDDM to:

- Store and retrieve Image Symbol Sets (ISS) and Vector Symbol Sets (VSS) by calls to GSLSS, PSLSS, PSLSSC, SSREAD, and SSWRT, and also by using the Image Symbol Editor.
- Store and retrieve device-dependent pictures by calls to FSSAVE, and FSSHOW.
- Retrieve GDDM-IMD-generated mapgroups, as required by calls to MSPCRT, MSQADS, MSQGRP, MSQMAP, and MSREAD. Retrieve and store graphics data format (ADMGDF) files, as required by calls to GSLOAD and GSSAVE.
- Read 4250 printer typographical font and code page data, as required by calls to GSCPG and GSLSS.

GDDM maintains these symbol sets, pictures, generated mapgroups, and ADMGDF files as members of partitioned data sets. The member-names that GDDM uses are those specified in the corresponding GDDM calls as "symbol-set names", "picture-names", "group-names", and "names" subject to modifications of these names by any character-substitution rules that apply.

The use of partitioned data sets containing symbol sets, pictures, generated mapgroups, and ADMGDF files can be controlled by the ESLIB routine whose syntax is described in the *GDDM Base Programming Reference, Volume 1*. This routine establishes the set of partitioned data sets that are to be used to store or retrieve a given type of object. The partitioned data sets used are identified to this routine by a list of file names.

The partitioned data sets allocated to the specified file names are searched in the order given to try to find an object. An object is stored only using the first file name of the list, even though it may have been retrieved from another one. If no file name list is provided, only the default file name is used for retrieving and storing GDDM objects.

GDDM uses OS/VS ENQ/DEQ services to ensure the integrity of partitioned data sets as they are written to. That is, GDDM ensures that at any one time, no more than one instance of GDDM has a particular partitioned data set opened for output processing. Partitioned data sets are kept open for output only while servicing the corresponding GDDM call.

The Interactive Chart Utility (part of GDDM-PGF) includes a directory function that supports list, delete, and copy operations on GDDM objects such as symbol sets, pictures, generated mapgroups, and ADMGDF files.

QSAM file processing

QSAM is used by GDDM to:

- Read an External Defaults File as part of initialization processing; see "External defaults file" on page 2.
- Write object modules as the result of requests from the Image Symbol Editor.

  Within a single invocation of the Image Symbol Editor, object modules are written consecutively to the selected sequential output destination. Each object module generated in this manner contains a control section (CSECT) with the name as specified by the editor, and is in a form suitable for link-editing with an application program for subsequent reference (typically, by the GSDSS or PSDSS calls). The TSO LINK command can be used to call the OS Linkage Editor for this purpose.
- Write data to intermediate sequential data sets used in the processing of calls to DSOPEN, DSCLS, FSOPEN, and FSCLS for queued printer output. The temporary data sets created are read by the TSO Print Utility, and after output to the printer is completed, the data sets are purged.
- Write output destined for a System Printer device as the result of calls to DSOPEN and DSCLS.
- Write data to high-resolution image files as the result of calls to DSOPEN and DSCLS for family-4 devices.
- Write trace records resulting from the FSTRCE function in GDDM. For a full description of the use of the GDDM trace function, see the *GDDM Diagnosis and Problem Determination Guide.*

BDAM file processing

BDAM processing is used by GDDM to read and write data to the Master Print Queue data set, used by GDDM to control requests for queued printer output made by calls to DSOPEN, FSOPEN, DSCLS, and FSCLS. GDDM uses OS ENQ/DEQ services to ensure the integrity of the Master Print Queue, because it is written to by multiple TSO users and by the GDDM TSO Print Utility. GDDM ensures that at any one time, no more than one instance of GDDM has the Master Print Queue available for input/output processing.

File-name usage

GDDM uses **file names** to refer to all the partitioned data sets and sequential destinations, with the exception of:

- The Master Print Queue and intermediate sequential data sets that are used in the processing of queued printer output.
- (Optionally, in the absence of appropriate file names): High-resolution image files used in the processing of family-4 devices.

The file names used are as defined in Table 3 on page 35. They can be changed, if required, after installation, by specifying new values in GDDM's external defaults, as described in Chapter 1, "Customizing your program and its environment" on page 1.

The user should ensure that the required file names are allocated to suitable data sets or destinations before GDDM is called. The data sets or destinations should have Data Control Block (DCB) characteristics as shown in Table 3 on page 35. The DCB characteristics for the data sets that contain GDDM-IMD's generated application data structures (file name ADMGNADS) and export files (file name ADMIFMT) are given in the *GDDM Interactive Map Definition* manual.

If necessary, GDDM supplies default DCB characteristics when output data sets are first opened.

Required file names can be allocated to the selected data sets or destinations using the TSO ALLOCATE command. Or, the file names can be allocated by DD statements in the user's TSO logon procedure, or by dynamic allocation routines in the application program.

GDDM uses OS/VS dynamic allocation services to refer to the Master Print Queue and associated intermediate sequential data sets. The data-set names used include a qualifier that is defined in the current GDDM external defaults. This can be changed, if required, after installation, as described in Chapter 1, "Customizing your program and its environment" on page 1. Or, the file name ADMPRNTQ can be used to identify a Master Print Queue data set other than that defined by the current GDDM external defaults.

The intermediate sequential data sets are allocated with a space allocation that is defined in the TSOS99S option in the current GDDM external defaults. The default allocation is equivalent to SPACE=(13030,(57,57)). If required, this can be changed after installation, as described under "GDDM external defaults – TSO" on page 134.

Dynamic allocation services will also be used if a print request has been specified to go directly to JES – by means of the PRINTDST processing option; see "TSO background print utility" on page 49.

GDDM also uses OS/VS dynamic allocation services to refer to high-resolution image files (for family-4 processing), unless suitable file names were previously allocated.

Table 3. GDDM data-set characteristics for TSO

Type of Data	GDDM default file name	Data set type	DCB characteristics		
			Record format (RECFM)	Record length (LRECL)	Block size (BLKSIZE)
Symbol sets	ADMSYMBL	Partitioned	F	400	400
			FB	400	400★n
Pictures	ADMSAVE	Partitioned	F	400	400
			FB	400	400★n
Generated mapgroups	ADMSAVE	Partitioned	F	400	400
			FB	400	400★n
GDF files	ADMGDF	Partitioned	F	400	400
			FB	400	400★n
4250 fonts (Note 3)	FONT4250	Partitioned	V	2052 (includes RDW)	≥ LRECL + 4
			VB		
4250 code pages (Note 3)	FONT4250	Partitioned	V	2052 (includes RDW)	≥ LRECL + 4
			VB		
Object modules	ADMDECK	Sequential data sets or SYSOUT classes	F	80	80
			FB	80	80★n
System Printer Output	ADMDECK	Sequential data sets or SYSOUT classes	VA	≥142 (Notes 1 and 2)	LRECL + 4
			VBA		≥ LRECL + 4
Family-4 Output	ADMCOLn or ADMIMAGE (optional)	Sequential data sets	V	2004 (for 4250) 8202 (for 38xx)	LRECL + 4
			VBM	2004 (for 4250) 8202 (for 38xx) (excludes RDW)	≥ LRECL + 4
Master print queue	ADMPRNTQ (optional)	BDAM data set	(Data set attributes provided when data set)		
Queued printer files	(Assigned by GDDM)	Sequential data sets	FBM	80	3200
Trace records	ADMTRACE	Sequential data sets or SYSOUT classes	VA	≥ 125	LRECL + 4
			VBA	≥ 125 (includes RDW)	≥ LRECL + 4
External default files	ADMDEFS	Sequential data sets	F	≥ 256	LRECL
			FB		LRECL★n
			V		LRECL + 4
			VB		LRECL + 4
Image files	ADMIMG	Partitioned	F	400	400
			FB	400	400★n
Image projection files	ADMPROJ	Partitioned	F	400	400
			FB	400	400★n

Notes:

1. The logical record length specified for files allocated for System Printer Output should be sufficient to contain the 4-byte Record Descriptor Word (RDW), the ASA control character, any Translation Reference Character (TRC) for 3800 devices, and the maximum number of columns for the type of System Printer selected by the application. The value 142 is adequate for any of the System Printer device characteristic tokens distributed with GDDM.
2. The output for all 3800 devices should contain table reference characters (TRCs). Consequently, the parameter DCB=OPTCD=J must be included in the output JCL. Additional parameters such as CHARS, FLASH, or FORMS may be required. For more information, see the *OS/VS2 MVS JCL* manual.
3. 4250 printer fonts and code pages are referenced by GDDM and are supplied as part of the 4250 typographical fonts licensed programs (program numbers 5771-AAA through 5771-AAW, and 5771-ACx, where x varies).

In TSO foreground operation, GDDM allows the unit specification for dynamically allocated data sets to be defaulted from the TSO user attribute data set (UADS).

In TSO Batch or MVS Batch, GDDM uses a unit specification taken from the TSOS99U option in the current GDDM external defaults. The default specification is "SYSDA". If required, this can be changed after installation, as described under "GDDM external defaults – TSO" on page 134.

Display terminal processing

By default, the PA1, PA2, and CLEAR keys are processed separately from other terminal input. The effects of these keys are:

CLEAR	clears the screen (no other action)
PA1	raises an TSO attention interrupt
PA2	raise a GDDM "reshow" condition.

The TSO CLEAR/PA1 protocol option of the DSOPEN function can be used to suppress this separate processing of the PA1 and CLEAR keys. The TSO Reshow protocol option of the DSOPEN function can be used to specify that a key other than PA2 should act as a "reshow" key. The use of these DSOPEN options is described in the *GDDM Base Programming Reference, Volume 1*.

The processing of these key functions is described in more detail below. Note that, because of this special processing, these key functions cannot be returned as terminal input by the ASREAD, FSSHOR, or MSREAD call, unless the key processing was modified by use of the DSOPEN protocol options.

Using the CLEAR key in full-screen mode

By default, terminal input using the CLEAR key is prevented by full-screen-mode protocols from being returned to GDDM and the application program. If the terminal user presses the CLEAR key, the screen is cleared, but no other operations occur. Specifically, GDDM may still wait to read input from the terminal, as a result of a call to ASREAD, FSSHOR, or MSREAD. Subsequently, terminal input by the user may conflict in format with that expected by GDDM; in this case, on return to the application program, an ASREAD or MSREAD operation issues this error message:

```
ADM0270 E SCREEN FORMAT ERROR
```

If this error message is issued, GDDM ensures that the screen buffer contents are subsequently restored.

The TSO PA1/CLEAR protocol option of the DSOPEN function can be used to suppress this special processing of the CLEAR key.

Entering attention interrupts in full-screen mode

By default, PA1 may be used, while GDDM is operating the terminal in full-screen mode, to cause an TSO attention interrupt. Unless the application program has established a special attention-processing function by means of the TSO STAX macro, using PA1 suspends the operation of both the application program and GDDM, and causes control to be passed to the terminal user, with the terminal in READY mode.

At this point, normal TSO protocols allow the terminal user to take the following alternative actions concerning the application program and GDDM:

- Abandon, by entering a new command to be executed
- Resume at the point of interruption, by using the ENTER key.

In the latter case, if GDDM had been interrupted while waiting for terminal input (as the result of a call to ASREAD, FSSHOR, or MSREAD), the ASREAD, FSSHOR, or MSREAD operation is completed without reading any input. On return to the application program, this error message is displayed:

```
ADM0405 E ATTENTION INTERRUPT
```

GDDM ensures that the screen buffer contents are subsequently restored.

If the application program has established a special attention-processing function by means of the TSO STAX macro, using PA1 clears the screen and displays an attention indicator, but does not force a paging condition or otherwise indicate to GDDM that the screen buffer contents were cleared. In these circumstances, the application program should subsequently issue an FSREST(1) call to cause the display buffer contents to be restored.

The TSO PA1/CLEAR protocol option of the DSOPEN function can be used to suppress this special processing of the PA1 key.

Reshow key processing in full-screen mode

Under TSO, GDDM operates an IBM 3270 series display in what is known as "full-screen mode". In this mode, if the terminal is to receive a non-full-screen message, such as an error message, or a message from another TSO user, the display screen is cleared, the alarm is sounded (if applicable), and the message is displayed.

If several such messages occur consecutively, the screen is cleared once, the alarm is sounded, and the messages are displayed in sequence. When the next GDDM full-screen transmission is received, a paging condition (indicated by three asterisks, ★★★, at the current line) is forced.

Pressing the ENTER key at this point queues a request to GDDM to completely retransmit the display buffer contents to the terminal (this is equivalent to the call FSREST(1)). Note that GDDM receives this reshow request only if it is (or when it is next) testing for input as a result of a call to ASREAD, FSSHOR, FSSHOW, GSREAD, MSREAD, or FSFRCE. TSO protocols are such that more partial GDDM transmissions may occur before GDDM starts retransmission of the contents of the buffers.

Using the reshow key (by default, PA2) during normal full-screen processing simulates the above conditions and causes GDDM to retransmit the contents of the buffers.

The TSO Reshow protocol option of the DSOPEN function can be used to define a key other than PA2 to act as the reshow key.

Device errors in full-screen mode

Under TSO in full-screen mode, non-full-screen output to the terminal can cause some full-screen transmissions to be "discarded" or wrongly interpreted. In some circumstances, this can cause device errors (displayed in the Operator Information Area of the terminal as "X PROGnnn").

After non-full-screen output has been received at the terminal, it is possible for more partial GDDM transmissions to occur before GDDM is able to begin retransmission of the screen contents; see "Reshow key processing in full-screen mode" on page 36.

In some circumstances, such partial GDDM transmissions may no longer be valid, and may cause device errors; for example:

- A partial transmission may contain a reference to a PS set. The PS set may not have been initialized because:
  - The particular PS set has not been used since the device was powered on, and
  - The GDDM transmission initializing the PS set was discarded by TSO in favor of a non-full-screen message.
- A partial transmission may assume the existence of a specific partition state on a 3290. The partition state may not exist because the GDDM transmission creating the partition state was followed by non-full-screen output that cleared the screen and thus destroyed the partition state.

If such device errors occur ("X PROGnnn" displayed in the terminal Operator Information Area), the terminal user should press the ENTER key to acknowledge the transmission. More partial transmissions (and more device errors) may occur until GDDM receives the reshow request, at which time GDDM automatically reconstructs the entire screen contents.

Line-by-line input in full-screen mode

In full-screen mode, TSO does not update line counts for any non-full-screen input entered at the terminal. This may result in such input being obliterated by subsequent non-full-screen output to the terminal.

Usually, this does not concern an application program using GDDM, because the program expects to use GDDM to read input from the terminal in full-screen mode. Also, GDDM sets full-screen mode off when invoked for termination by means of the FSTERM call.

However, if an application program ends without a call to FSTERM (as the result of an ABEND or other error), it is possible for the terminal user subsequently to be prompted to enter line-by-line input with full-screen mode still enabled for that terminal. In this situation, the terminal user may be able to prevent obliteration of the line-by-line input by using PA1. This raises an TSO attention interrupt, and also turns off full-screen mode.

NOEDIT mode under TSO

Under TSO, GDDM uses NOEDIT mode to operate a "queriable" IBM 3270 series terminal (that is, a terminal that supports the Read Partition (Query) Structured Field).

Usually, this would not concern an application program using GDDM, because GDDM maintains this mode only when reading from a terminal. However, if GDDM or the application program is abnormally terminated, it is possible for the terminal user subsequently to be prompted to enter line-by-line input with the NOEDIT mode still enabled for that terminal.

In this situation, the user may find that line-by-line input cannot be correctly interpreted, and may receive one of these messages:

```
IKJ56601I COMMAND SYSTEM RESTARTING DUE TO CRITICAL
          ERROR
IKJ56600I UNRECOVERABLE COMMAND SYSTEM ERROR
```

To recover from this situation, and to prevent the TSO logon session from being terminated, the terminal user must press PA1; this causes an TSO attention interrupt and turns off the NOEDIT mode.

Using APL terminals

Under TSO, device information provided by the subsystem does not distinguish between an IBM 3277 Model 2 display terminal and an IBM 3278 or 3279 Model 2 display terminal, unless the latter is defined to be "queriable"; that is, is defined to support the Read Partition (Query) Structured Field by the 3274 Controller Configuration Support C and the Extended Character Set Adapter (feature number 3610).

By default, GDDM resolves this ambiguity by assuming that the device is an IBM 3277 Model 2. If the device is actually a nonqueriable IBM 3278 or 3279 Model 2 with an APL Feature, and if the APL character set is to be referred to by an application, the GDDM default assumption must be overridden to ensure correct operation of the device. The GDDM default can be overridden in any of these ways:

1. The application can specify an explicit device token (for example, ADMK782A) on a DSOPEN call to initialize the device; see the *GDDM Base Programming Reference, Volume 1*.
2. The TSOAPLF option in GDDM's current external defaults can be modified to cause GDDM to assume by default that a nonqueriable Model 2 display terminal is an IBM 3278 or 3279. This option can be specified:
   - In an External Defaults Module
   - In an External Defaults File that was allocated to ddname ADMDEFS, or
   - In a SPINIT, ESSUDS, or ESEUDS call in an application program.

   See Chapter 1, "Customizing your program and its environment" on page 1.

Also, under TSO, device information provided by the subsystem does not indicate whether a 3277 Model 2 display or a nonqueriable 3278 or 3279 display actually has the appropriate APL feature.

By default, GDDM assumes that such a device has the APL feature, and it selects an appropriate set of translation tables. (For more details, see the description of ASTYPE in the *GDDM Base Programming Reference, Volume 1* and the *GDDM Installation and System Management for MVS* manual.) If the device does not have the APL feature, the use of character code points that correspond to APL characters may result in incorrect output at the device.

The GDDM default can be overridden in either of the following ways. The application program can:

- Specify an explicit device token (for example, ADMK7720) in a DSOPEN call to initialize the device (see the *GDDM Base Programming Reference, Volume 1*) or by means of nickname facilities (see "Using nicknames to define device characteristics" on page 3).
- Use the ASTYPE call to specify the appropriate set of translation tables, as follows:

Device type	Translation type number
3277	3277
3277-APL	32771
3278, 3279	3279
3278-APL, 3279-APL	32791

  For a full description of the operation of alphanumeric translation tables, see the *GDDM Installation and System Management for MVS* manual.

Using GDDM under TSO batch

TSO Extensions (TSO/E) is a licensed program (program number 5665-285) that provides a TSO Batch environment in which TSO commands and command procedures can be run in the background. GDDM can be used in this environment, in normal MVS Batch, subject to the following considerations.

- TSO Batch applications must be link-edited using the information under "Link-editing a GDDM application program" on page 33.
- GDDM processes any External Defaults File allocated by means of a DD statement; the default ddname is ADMDEFS.
- The GDDM default error exit reports errors using WTP (Write-To-Programmer). These messages usually appear on the JOB LOG output.
- GDDM dynamically allocates queued printer files or high-resolution image files for family-4 devices using a unit specification that is defined in the TSOS99U option in the current GDDM external defaults. The default unit specification is SYSDA. If required, this can be changed, as described under "GDDM's default values, listed by subsystem" on page 127.
- The GDDM-supplied interactive utilities necessarily use the default primary device (the "TSO terminal"), unless called for noninteractive processing. Therefore, these utilities cannot be run interactively in TSO batch.
- The default primary device (the simulated TSO terminal) is not suitable for GDDM full-screen operations. GDDM diagnoses any attempt to use this device.

  Therefore, an application must include an explicit DSOPEN to identify a nondefault primary device (for example, a dummy device or non-family-1 device).
- The GDDM default error exit reports errors using WTP (Write-To-Programmer). User PROFILE options can be used to cause the messages to appear as part of the session output file (SYSTSPRT). The TSO command to request that WTP messages appear on the session output file is:

  ```
  PROFILE WTPMSG
  ```

  and this should be included in the session input file (SYSTSIN) before GDDM is used.
- Unless the application is running as part of a RACF job with USERID, no default data-set-name prefix or userid is defined. A default data-set-name prefix may be required by GDDM for dynamic allocation of queued printer files or high-resolution image files (for family-4 devices). The TSO command to establish a default data-set-name prefix is:

  ```
  PROFILE PREFIX(dsname-prefix)
  ```

  and this should be included in the session input file (SYSTSIN) before GDDM is used.
- GDDM uses the userid only for annotation purposes (in print files and trace files). In the absence of a userid, GDDM uses the JOB name.

Using GDDM under MVS batch

These items are specific to processing under MVS Batch:

- MVS Batch applications must be link-edited using the information under "Link-editing a GDDM application program" on page 33.
- GDDM processes any External Defaults File allocated by means of a DD statement; the default ddname is ADMDEFS.
- The GDDM default error exit reports errors using WTP (Write-To-Programmer). These messages usually appear on the JOB LOG output.
- GDDM dynamically allocates queued printer files or high-resolution image files for family-4 devices using a unit specification that is defined in the TSOS99U option in the current GDDM external defaults. The default unit specification is SYSDA. If required, this can be changed, as described under "GDDM's default values, listed by subsystem" on page 127.
- The GDDM-supplied interactive utilities necessarily use the default primary device (the "TSO terminal"), unless called for noninteractive processing. Therefore, these utilities cannot be run interactively MVS Batch.

- The default primary device (the simulated TSO terminal) is not available for GDDM full-screen operations. GDDM diagnoses any attempt to use this device.

  Therefore, an application should include an explicit DSOPEN to identify a nondefault primary device (for example, a dummy device or non-family-1 device).

- The default data-set-name prefixes or userids that are given under TSO are not applied. GDDM does not apply such a prefix for dynamic allocation of queued printer files or high-resolution image files for family-4 devices. Queued printer files are allocated with names of the form:

  ```
  ADMPRINT.REQUEST.#nnnnn
  ```

  where the string ADMPRINT is as provided in GDDM's defaults. The name ADMPRINT can be changed by specifying a new value in the TSOPRNT option in GDDM's external defaults. For full details, see "GDDM external defaults — TSO" on page 134.

- GDDM uses the JOB name for annotation purposes in print and trace files.

Sample JCL for GDDM under TSO

```
//************************ TSO ****************************************
//*
//* Sample JCL to link-edit a GDDM/TSO
//* sample program or user-written application.
//*
//* xxxxxxxx is the name under which the program load module is
//* generated.
//*
//*********************************************************************
//*
//jobname    JOB  accounting info,..........
//*
//* Link-edit step
//*
//* Include INCLIB to reference library containing GDDM interface
//* modules, as shown.
//*
//* In the specified INCLUDE statement,
//* leave   ADMASNT unchanged  if using the nonreentrant interface
//* replace ADMASNT by ADMASRT if using the reentrant interface
//*            or   by ADMASPT if using the system programmer interface
//*
//LKED       EXEC PGM=IEWL,PARM='XREF,LIST',REGION=768K
//SYSPRINT   DD   SYSOUT=A
//SYSLIB     DD   DSN=as-required-by-application,DISP=SHR
//INCLIB     DD   DSN=GDDM.OSPID.GDDMLOAD,DISP=SHR
//SYSLMOD    DD   DSN=user-load-module-dataset,DISP=SHR
//SYSUT1     DD   UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSLIN     DD   *
      . . . .
      . . . .
 Program object deck here.
      . . . .
      . . . .
 INCLUDE INCLIB(ADMASNT)
 NAME    xxxxxxxx(R)
/*
```

Chapter 6. Using GDDM under VM/CMS

This chapter describes the use of GDDM under the VM/CMS operating system. It contains the following topics:

- Compiling a GDDM PL/I application program
- Loading a GDDM application program
- Running a GDDM application program or utility
- Data sets and file processing
- Display terminal conventions
- Using APL terminals
- Batch processing
- Running programs under VM/XA.

How to use the GDDM print utility is described in Chapter 7, "The GDDM print utilities" on page 47.

Note: GDDM cannot be run in the VM CMS/DOS environment. Therefore, it cannot be successfully invoked under VM/CMS by application programs compiled using DOS compilers such as the PL/I DOS Optimizing Compiler.

When writing an application program, you must access MACLIBs to compile your programs, if you are to include the GDDM standard declarations. You must also access TXTLIBs to load your program, and possibly to run your program, as described on page 41.

If you are a terminal user you must know the PA key usage and other terminal conventions.

You must also be aware of the file usage of GDDM to help you manage the storage of your virtual machine.

You should also be aware that under VM/CMS the print utility may have to be invoked separately after invoking printing functions from the ICU or from an application program. The print utility is described under Chapter 7, "The GDDM print utilities" on page 47.

Compiling a GDDM PL/I application program

If you use the GDDM-supplied declarations in your program, you must access the library that contains them before compiling, by issuing a command of the form:

```
GLOBAL MACLIB ADMLIB
```

Loading a GDDM application program

Before loading a VM/CMS application, the CMS GLOBAL command must be executed to identify the appropriate GDDM TXTLIB to be searched for GDDM function references.

The GDDM TXTLIB to be specified in the CMS GLOBAL command depends on the type of GDDM interface being used, as follows:

Interface	**GDDM TXTLIB**
Nonreentrant	ADMNLIB
Reentrant	ADMRLIB
System programmer	ADMPLIB

The command takes the form:

```
GLOBAL TXTLIB ADMxLIB
```

where ADMxLIB is one of the TXTLIBs above.

The application can then be loaded, typically with a command of the form:

```
LOAD appl-name
```

Running a GDDM application program or utility

All the required run-time GDDM facilities may have been made available in a VM/CMS Discontiguous Shared Segment (DCSS) as described in *GDDM Installation and System Management for VM* manual. If not, before running a GDDM application program or utility, the CMS GLOBAL command must be executed to identify appropriate GDDM TXTLIBs to be searched for routines required dynamically during execution.

If GDDM/VM or GDDM/VMXA ("GDDM Base") only has been installed, the installation procedure will have placed the required routines in ADMGLIB TXTLIB.

If GDDM-PGF has also been installed, the installation procedure will have placed additional GDDM-PGF routines in ADMPLIB TXTLIB.

If the GDDM National Language (GDDM NL) special feature has also been installed, the installation procedure will have placed additional GDDM NL routines in ADMPLLIB TXTLIB (this contains language-dependent routines for languages other than American English).

Therefore, the CMS GLOBAL command to be executed is:

GDDM Base only:

```
GLOBAL TXTLIB ADMGLIB
```

GDDM Base and GDDM-PGF:

```
GLOBAL TXTLIB ADMPLIB ADMGLIB
```

GDDM Base and GDDM Base NL:

```
GLOBAL TXTLIB ADMHLIB ADMGLIB
```

GDDM Base, GDDM-PGF NL, and both NL:

```
GLOBAL TXTLIB ADMHLIB ADMPLIB ADMQLIB ADMGLIB
```

If any other GDDM product besides GDDM Base has been installed, issue ADMGLIB as the last parameter in the GLOBAL command parameters list. Failure to do so may cause GDDM abend code 1064.

Having issued the GLOBAL command, if required, the application can then be started, typically with a command of the form:

```
START appl-entry-point
```

If the application requires no special parameters on the START command, the steps described above of loading and starting an application can be combined. For example:

```
GLOBAL TXTLIB ADM.... ... ...
LOAD appl-name (START
```

Note: It is mandatory that ADMGLIB is specified ***after*** ADMPLIB on the GLOBAL command.

Considerations for running multiple instances of GDDM

An application using the reentrant or system programmer interface to GDDM may invoke more than one instance of GDDM concurrently. Such an application should ensure that the first instance of GDDM to be initialized (using FSINIT or SPINIT) is also the last to be terminated (using FSTERM). This prevents any GDDM Shared Segment (DCSS) being unloaded prematurely.

Data sets and file processing

When running under VM/CMS, GDDM/Base and GDDM-PGF use two types of file processing:

- "Native" CMS file processing to read and write conventional CMS disk files direct.
- "Native" CMS spool file processing to write output to the punch device, 00D, and the printer device, 00E.

GDDM-IMD uses additional types of file processing. For details, see the *GDDM Interactive Map Definition* manual.

Native CMS file processing

Native CMS file processing is used by GDDM:

- To store and retrieve Image Symbol Sets (ISS) and Vector Symbol Sets (VSS), as a result of calls to GSLSS, PSLSS, PSLSSC, SSREAD, and SSWRT, and through the Image Symbol Editor.
- To store and retrieve device-dependent pictures, as a result of calls to FSSHOW, FSSHOR, and FSSAVE.
- To retrieve GDDM-IMD-generated mapgroups, as required by calls to MSPCRT, MSQADS, MSQGRP, MSQMAP, and MSREAD.
- To retrieve and store Graphics Data Format (ADMGDF) files, as required by calls to GSLOAD and GSSAVE.
- To write text files, as a result of requests through the Image Symbol Editor.
- To write queued print files, as a result of calls to DSOPEN, DSCLS, FSOPEN, and FSCLS, subsequently to be processed by the GDDM VM/CMS Print Utility.
- To write system printer disk files, as the result of calls to DSOPEN and DSCLS.
- To write data to high-resolution image files as the result of calls to DSOPEN and DSCLS for family-4 devices.
- To read 4250 printer typographical font and code page data, as required by calls to GSCPG and GSLSS.
- To write trace output resulting from execution of GDDM with the trace facility enabled. For a description of the enablement and use of GDDM trace facilities, see the *GDDM Diagnosis and Problem Determination Guide*.
- To read an External Defaults File as part of initialization processing; see page 2.

All the above types of data are stored and retrieved using CMS file identifiers where, by default:

filename is determined according to the type of data, as follows:

- For symbol-sets, pictures, generated mapgroups, ADMGDF files, print files, high-resolution image files (for family-4 devices), and 4250 printer fonts and code pages, the filenames used are those specified in the corresponding GDDM calls as symbol-set names, picture names, group names, ADMGDF file names, print-destination names, device names, and code-page names, subject to modification of these names by character-substitution rules.
- For text-files, the filenames used are those specified through the symbol editor. Each text file generated contains a correspondingly-named control section (CSECT), and is in a form suitable for link-editing with an application program for subsequent reference, typically by the GSDSS or PSDSS call.
- For trace output, the filename used is as defined in Table 4 on page 43 or as modified by the user in the CMSTRCE option in the current GDDM external defaults; see "GDDM external defaults – VM/CMS" on page 137.
- For External Defaults File input, the filename used is as defined in Table 4 on page 43 or as modified by the user in the CMSDFTS option in the current GDDM external defaults; see "GDDM external defaults – VM/CMS" on page 137.

filetype is determined by the GDDM default name (see Table 4 on page 43) or as modified by the user in the current GDDM external defaults (see "GDDM external defaults – VM/CMS" on page 137).

filemode is:

"A1" for output, causing data to be stored on the A-disk (which should be accessed as read/write for such operations).

"★" for input, causing accessed data to be searched in the standard order.

Table 4. GDDM data-set characteristics for VM/CMS

Type of data	GDDM default filetype	Record format (RECFM)	Record length (LRECL)
Symbol sets	ADMSYMBL	F	400
Pictures	ADMSAVE	F	400
Generated mapgroups	ADMGGMAP	F	400
GDF files	ADMGDF	F	400
Text files	ADMDECK	F	80
System printer output	ADMLIST (but directed to virtual printer by default)	V	according to device characteristics
Family-4 output	ADMCOLn or ADMIMAGE	V	≤ 2000 (for 4250) ≤ 8202 (for 38xx)
4250 printer fonts (see Note)	FONT4250	V	≤ 2048
4250 printer code pages (see Note)	FONT4250	V	≤ 2048
Queued printer files	ADMPRINT	F	80
Trace records	ADMTRACE (default filename is ADM00001)	V	≤ 121
External	ADMDEFS	F	≤ 256
Files	filename is PROFILE)	V	≤ 256
Image files	ADMIMG	F	400
Image Projection Files	ADMPROJ	F	400

Note: 4250 printer fonts and code pages are referenced by GDDM and are supplied as part of the 4250 typographical fonts licensed programs (program numbers 5771-AAA through 5771-AAW, and 5771-ACx, where x varies).

The DSOPEN call allows the filenames, filetypes, and filemodes of queued printer, system printer, and high-resolution image (family-4) disk file devices to be explicitly specified by means of the name-list parameter.

The Interactive Chart Utility (part of GDDM-PGF) includes a directory function that supports list, delete, and copy operations on GDDM objects such as symbol sets, pictures, generated mapgroups, and ADMGDF files.

Native CMS spool file processing

Native CMS spool file processing is used by GDDM:

- To write output to the virtual punch, as the result of calls to DSOPEN and DSCLS.
- To write output to the virtual printer, as the result of calls to DSOPEN and DSCLS.
- To write trace output resulting from the execution of GDDM with the trace facility enabled. For a description of enabling and using GDDM trace facilities, see the *GDDM Diagnosis and Problem Determination Guide*.

GDDM writes 3270 device (family-1) output either directly to a 3270-type terminal or to the virtual punch, according to the name specified in the DSOPEN call. 3270 device output written to a virtual punch is in the form of 80-byte records in the following format:

Record 1 Virtual CCW (8 bytes) including SIO count. The CCW opcode is one of the following:

X'01'	Write
X'05'	Erase/Write
X'0D'	Erase/Write Alternate
X'11'	Write Structured Field.

Record 2 Data stream – as many 80-byte records as are necessary to contain "SIO count" bytes of data.

Record n Virtual CCW (8 bytes) including SIO count.

Record n+1 Data stream – as many 80-byte records as are necessary to contain "SIO count" bytes of data.

CP SPOOL and CP TAG commands should be used to direct the virtual punch output to a destination that is capable of processing data in the above format (such as RSCS Networking Version 2). The CPSPOOL and CPTAG processing options in DSOPEN can be used to issue such commands automatically.

GDDM writes System Printer output either to a disk file or to the virtual printer, according to the name specified by the DSOPEN call. Data written to a System Printer device contains ASA control characters and, for 3800 devices, Translation Reference Characters (TRCs). The CP SPOOL and CP TAG commands should be used to specify additional special parameters such as CHARS, FLASH, or FCB that may be required for 3800 devices.

GDDM writes trace output either to a disk file or to the virtual printer, according to the filename defined in the current GDDM external defaults (or modified in the CMSTRCE option; see "GDDM external defaults – VM/CMS" on page 137). If the filename is defined as all blanks, GDDM directs the trace output to the virtual printer.

Display terminal conventions

The following comments apply only when the display terminal being used is the CMS user virtual console.

Under VM/CMS, by default, the PA1 and PA2 keys are processed separately from other terminal input. The effect of using these keys is as follows:

PA1 Pressing this key causes CP mode to be entered and a CP READ status to be displayed. In this environment, any CP commands may be issued. To return from the CP environment, issue the CP command BEGIN.

PA2 Pressing this key causes the CMS SUBSET environment to be entered and a RUNNING status to be displayed. In the CMS SUBSET environment, any CMS commands that run in the transient area may be issued. For example:

```
ACCESS    LISTFILE    RENAME
CP        PRINT       RETURN
DISK      PUNCH       SET
ERASE     QUERY       STATE
EXEC      READCARD    TYPE
```

To return from the CMS SUBSET environment, issue the CMS SUBSET command RETURN.

On return from the CP or CMS SUBSET environment, GDDM retransmits the screen buffer contents, and then waits for more input.

As a result of the above special processing, PA1 and PA2 cannot, by default, be returned as terminal input by the ASREAD, FSSHOR, or MSREAD call. However, the CMS PA1/PA2 protocol option of the DSOPEN function can be used to suppress this special processing selectively. The use of this option to the DSOPEN function is described in the *GDDM Base Programming Reference, Volume 1*.

Asynchronous interrupts on VM/CMS

The following comments apply only when the display terminal being used is the CMS user virtual console.

Using the ENTER key

Unless the application program has established any special attention-processing functions, the ENTER key (and no other attention key) may be used while GDDM is operating to cause an asynchronous CMS attention interrupt. This suspends the operation of both the application program and GDDM, and causes control to be passed to the terminal user, with the terminal in line-by-line VM READ mode.

In this mode, normal CMS protocols usually allow the terminal user to take one or more of the following actions:

- Resume at the point of interruption, by pressing the ENTER key.
- Enter an "immediate" CMS command (for example, HO, HT, HX, RO, RT, or SO).
- Enter other commands – such commands are stacked for execution at the next entry into normal CMS or CMS SUBSET mode.

After any of the above actions (except HX), GDDM ensures that the screen buffer contents are restored.

Using other attention keys

Application programs can request extended processing of asynchronous interrupts by specifying the CMS attention handling option (processing option group 1001) of the DSOPEN call.

Requesting "extended attention handling" indicates that an application program attention feedback block may have been located by means of the DSOPEN CMS attention option.

If this is done, an attention key may be used while GDDM is operating to cause an asynchronous CMS attention interrupt (unless a line-by-line message has already placed the terminal into line-by-line mode, in which case, only ENTER causes an attention interrupt). An exception is the PA1 key, which causes CP mode to be entered, unless the PA1 special processing was suppressed as described above.

Also, if the attention feedback block is of nonzero length, GDDM stores up to two words of information in this block (according to the length specified), indicating the nature of the interrupt. The information stored is as follows:

- Attype – attention type (full-word integer)
- Attval – attention type value (full-word integer).

where these are as defined for the ASREAD call (see the *GDDM Base Programming Reference, Volume 1*).

An application program may intercept such attention interrupts by establishing a special attention-processing exit using the VM/CMS simulation of the TSO STAX macro. A STAX exit of this form should be established before the device representing the virtual console is initialized (that is, before SPINIT/DSOPEN), and should not be cleared until after the device has been terminated (that is, after FSTERM/DSCLS). A STAX exit may examine the contents of the attention feedback block to determine the cause of the interrupt. GDDM must not be invoked from a STAX exit if GDDM was already running at the time of the interrupt.

GDDM disables all STAX exits and attention-processing functions before initiating the CMS SUBSET environment, and restores them on return.

VM-initiated asynchronous interrupts

VM/CMS may generate "virtual" asynchronous interrupts before the display of a priority message.

If such an interrupt occurs while the terminal user is entering data in response to an ASREAD, FSSHOR, or MSREAD call, GDDM allows the priority message to be displayed immediately, but saves and restores any data entered by the terminal user. An interrupt occurring at this time may also cause any application program attention-processing exit to be entered, with an attention feedback block indicating an interrupt of type 6 ("Undefined").

VM-initiated asynchronous interrupts are not otherwise apparent to the GDDM terminal user or application program.

Interactions with non-GDDM device interrupt handling

An application program that uses GDDM to communicate with the CMS virtual console and uses the CMS HNDINT macro as part of its own interrupt handling for devices not controlled by GDDM must be written in such a way as to avoid recursion of the CMS HNDINT macro.

If the virtual console operator causes an asynchronous attention interrupt, GDDM's STAX exit gains control. This exit attempts to read from the terminal to determine the nature of the interrupt. During this processing, GDDM issues a CMS HNDINT WAIT macro.

If the application program already has a CMS HNDINT WAIT macro active at the time, interference between the macros occurs, and the application program's HNDINT WAIT macro is likely to complete immediately, with random results.

To prevent this type of interaction, the application program should suppress GDDM's STAX exit (and the attention-processing functions that go with it) over the duration of its own HNDINT WAIT macro. The application program can do this by clearing (and saving) the value in the TAXEADDR field in the CMS Nucleus Constant Area (NUCON) before invoking HNDINT WAIT and by restoring the value in TAXEADDR after the HNDINT WAIT macro has completed.

Using APL terminals

This section describes how GDDM interacts with nonqueriable displays and printers that have the APL feature.

Using nonqueriable displays with the APL feature

Under VM/CMS, device information provided by the subsystem does not indicate whether a nonqueriable 3278 or 3279 display has the appropriate APL feature. (A "queriable" terminal is one that supports the Read Partition (Query) structured field.)

If the CP TERM APL ON command was issued, GDDM assumes by default that such a device has the APL feature, and selects an appropriate set of translation tables. (For more details, see the description of ASTYPE in the *GDDM Base Programming Reference, Volume 1* and the *GDDM Installation and System Management for VM* manual.) If the device does not have the APL feature, the use of character code points corresponding to APL characters may result in wrong output at the device.

If the CP TERM APL OFF command was issued, GDDM assumes that such a device does not have the APL feature.

The GDDM default can be overridden in either of the following ways. The application program can:

- Specify an explicit device token (for example, ADMK7720) in a DSOPEN call to initialize the device (see the *GDDM Base Programming Reference, Volume 1*) or by means of nickname facilities (see "Using nicknames to define device characteristics" on page 3).
- Use the ASTYPE call to specify the appropriate set of translation tables, as follows:

Device type	Translation type number
3278, 3279	3279
3278-APL, 3279-APL	32791

  For a full description of alphanumeric translation tables, see the *GDDM Installation and System Management for VM* manual.

Using nonqueriable printers with the APL feature

Under VM/CMS, device information provided by the subsystem does not distinguish between IBM 3270 printers, unless they are "queriable" (that is, unless they support the Read Partition (Query) Structured Field).

By default, GDDM assumes that any APL feature on a nonqueriable printer is the APL/Text Feature, rather than the Data Analysis – APL Feature. If a printer (such as an IBM 3284 or 3286) has the Data Analysis – APL Feature, and if the APL character set is to be referenced, the GDDM default assumption must be overridden to ensure correct operation of the device.

The CMSAPLF option in GDDM's external defaults can be modified (by specifying the value DATAANAL) to cause GDDM to assume by default that an APL feature installed on a nonqueriable IBM 3270 printer terminal is the Data Analysis – APL Feature. This option can be specified:

- In an External Defaults Module, or
- In an External Defaults File.

See Chapter 1, "Customizing your program and its environment" on page 1.

Batch processing

A disconnected Virtual Machine, such as a machine using the CMS batch facility, can simulate batch processing. In such an application, you **cannot** communicate with the default primary device because there is no such device. The application must use DSOPEN to indicate the device that is to be used; for example:

- A dummy device
- A queued printer
- A high-resolution image file
- A dialed-in display station
- An attached printer.

In batch processing, an application might:

- Create queued printer output for subsequent printing by the GDDM print utility. The queued printer output would, perhaps, be created by using the chart utility noninteractively.
- Create a high-resolution image file for a family-4 device.
- Create FSSAVE files for subsequent interactive use with FSSHOW. The files would be created by using a dummy device.

GDDM application programs under VM/XA

The Base product, GDDM/VMXA, enables GDDM and application programs to exploit VM/XA SP, in particular 31-bit addresses and virtual machines bigger than 16 megabytes. Generally, programming for GDDM/VMXA is no different from programming for GDDM/VM, although there are a few special considerations.

Migration: To run under VM/XA SP, modules must be generated with GDDM/VMXA. To run under VM/SP, they must be generated with GDDM/VM. Programs transferred from one system to the other must therefore be re-generated.

User exits: Programmers should take care when specifying the addresses of user exits to GDDM. GDDM uses the convention that the top bit of such addresses identifies its addressing mode (AMODE). Also, if GDDM is initialized with the SPINIT call, and this call was issued in 24-bit mode, GDDM clears bits 1 through 7 of each address word that it processes.

Interception of PA1: Programs that request (with the GDDM CMSINTRP processing option) that PA1 key interrupts be passed to them will cause the CP TERMINAL BRKKEY value to be set to NONE, regardless of its original setting. This action is consistent with that of CMS when its full-screen mode is entered.

Dialed devices: If GDDM is used to drive a dialed display device, then when that device is closed it will also be dropped from the virtual machine. This is due to a feature of the CMS Console Services support that causes a dialed device to be dropped when the last console path to it is closed.

Chapter 7. The GDDM print utilities

There are several utility programs provided as a part of GDDM: the queued printer support facility, the image print utility, and the composite document print utility. They are described in that order in this chapter.

The main GDDM print utility is a queued printer support facility that consists of two parts operating asynchronously:

- The GDDM printing subroutines

  These are invoked by call statements in the application program. When a queued printer is closed by a call to FSCLS or DSCLS, a request is queued to the output print utility, and all output to be printed is copied to a print file.

- The output print utility

  This is supplied in a version appropriate for the subsystem in use. Invocation and operating instructions for each version are given below. Messages issued by the utility are listed in the *GDDM Messages* manual. The output print utility can write a print file both to a 3270-family printer and (except under IMS/VS) to a plotter that is attached to a 3179-G or 3192-G color display station, or a 3270-PC/G or 3270-PC/GX work station.

  Note: In general, print files produced under one GDDM release cannot be printed using the print utility of another release.

On printers (but **not** on plotters), a header page is printed at the start of each file (unless it was explicitly suppressed when the FSOPEN or DSOPEN call was issued). The header page identifies the origin of the print file and the date and time that it was created. The origin of the file depends upon the subsystem as follows:

CICS/VS	the transaction identifier
IMS/VS	the userid, if available, or the logical terminal name (or asterisks, ★, if neither is available)
TSO	the userid
VM/CMS	the userid.

The formats of the date and time are defined in the current GDDM defaults. For details of how to change these values, see "Changing GDDM's default values" on page 127.

For plotters, the ORIGINID processing option in DSOPEN (see the *GDDM Base Programming Reference, Volume 1*) can be used to superimpose an alphanumeric field containing similar information on each plotted page.

Processing for a printer device

By default, GDDM performs a page eject at the end, but not at the start, of a print file. This action is controlled by setting the appropriate value in the current GDDM defaults. For details, see "Changing GDDM's default values" on page 127.

If any errors are detected during the printing process, an error page is printed that summarizes a maximum of 19 errors. Each error message is prefixed by the number of the page that was being generated when the error was detected, and, if possible, by the function that GDDM was running at the time the error was detected. (The count starts with the header page, if there was one.)

If multiple copies are being printed, an error page is printed after each copy during the printing of which errors were detected. If only a single page is being printed for each copy, the processing is optimized so that some errors are only detected during the printing of the first copy. In other circumstances, errors may be repeated for each copy.

Serious errors where the messages cannot be printed (for example, errors occurring at initialization of the print utility), are written to a system-dependent destination as follows:

CICS/VS	the error log
IMS/VS	broadcast to the Master Terminal Operator
TSO	the system operator
VM/CMS	the terminal operator.

Processing for a plotter device

To cause the GDDM print utility to write a print file to a plotter, it is usually necessary to specify the DSOPEN processing option, STAGE2ID, when the print file is created. For full details, see Appendix B, "Processing option groups and name-lists" on page 149.

To ensure that control information for the plotter is honored by GDDM, nickname statements are required in the defaults file. For example, to change the pen velocity:

```
ADMMNICK NAME=QPLOT,TOFAM=2,
         TONAME=LUNAME,DEVTOK=L7372,
         PROCOPT=((STAGE2ID,PLOTTER))
ADMMNICK NAME=PLOTTER,FAM=1,
         TONAME=(*,ADMPLOT),
         PROCOPT=((PLTPENV,10))
```

In the above example, the user would direct the output to "QPLOT".

The GDDM print utility begins to plot on a device as soon as it receives a print file for that device. At the same time, it sends a status message to the associated 3179-G or 3192-G color display station, 3270-PC/G or 3270-PC/GX work station, or device supported by GDDM-PCLK.

The GDDM print utility pauses at the end of each page that is plotted to give the operator an opportunity to reload the plotter device. GDDM sends another status message at that time and prompts the operator to press any attention key to cause plotting to continue, or to complete the processing of the print file.

At any time, the plotting of the current page may be canceled by pressing the CLEAR key on the associated work station.

If any errors are detected during the plotting process, the error messages are added to the status messages on the associated 3179-G or 3192-G color display station, 3270-PC/G or 3270-PC/GX work station, or device supported by GDDM-PCLK. These messages are accumulated; each error message is prefixed by the number of the page that was being plotted when the error was detected and, if possible, by the function that GDDM was running at the time the error was detected.

If multiple copies are plotted, the process is repeated for each copy.

Serious errors, where the messages cannot be displayed (for example, errors that occur when the print utility is initialized), are written to the error log (under CICS/VS), the system operator (under TSO), or the terminal operator (under VM/CMS).

Note: Any nickname processing for the associated 3179-G color display station, or 3270-PC/G or 3270-PC/GX work station, is suppressed during the time that plotting takes place.

CICS/VS print utility

The CICS/VS version is ADMOPUC, and runs as a transaction that automatically processes print requests.

Invocation

No explicit invocation is required. Print requests automatically schedule the transaction, using the Interval Control facilities of CICS/VS. Note that CICS/VS Interval Control uses CICS/VS Temporary Storage facilities. If this is defined as recoverable, the transaction is not initiated until a synchronization point is reached. Printing may therefore be delayed until the display task terminates or until a user synchronization point is reached. The GDDM-PGF Interactive Chart Utility (ICU) does not contain any explicit synchronization points.

Printer and plotter operating instructions

The printer or plotter must be prepared for use according to the operating instructions for the model, and the paper must be aligned to the top of the page.

SCS mode PA switches

For a printer operating in SCS mode, the PA1 and PA2 switches can be used. These switches allow limited communication with the utility, and are used with the Hold Print/Enable Print switch, as described in the Component Description and Operator's Guide for the appropriate printer. The effect of each switch is as follows:

PA1 Sending a PA1 switch code to the utility causes it to restart printing of the current request at the page after the header page. For a multiple-copy request, printing is resumed at the start of the copy being processed at the time of the interrupt.

PA2 Sending a PA2 switch code to the utility causes it to restart printing of the current page.

Messages

Most messages are issued as part of the output from the utility. If it is not possible to send the error messages to the terminal, they are sent to the GDDM error log by Transient Data Facilities as described in Chapter 2, "Using GDDM under CICS/VS" on page 7.

The VSE print job utility

The VSE Print Job Utility creates print files for 38xx and 4250 printers, from ICU chart format and data (ADMCFORM and ADMCDATA) files, GDDM graphics data format (GDF) files, and GDDM images (ADMIMG files). Further details, including its end-user interface, are given in the *GDDM Release Guide*.

The user interface program ADMUPRTC merges values entered into a menu with skeletal JCL supplied by a system programmer, to form a job-stream that it submits to VSE for batch processing. The skeletal JCL is defined when GDDM/VSE is installed: instructions are given in *GDDM Installation and System Management for VSE* manual.

The VSE Print Job Utility creates a primary data stream, unless a page segment (secondary data stream) is specified in a GDDM processing option when the utility is invoked. The print program stores any page segments it creates in a sublibrary. You will need:

```
LIBDEF *,CATALOG=....
```

in the job stream that invokes the print program to create a page segment, otherwise you will get a GDDM abend with the code SVC6E. If ADMUPRTC is used, the LIBDEF statement must be added to the skeletal JCL.

Running the VSE print program without using ADMUPRTC

You do not have to use the ADMUPRTC utility to generate and submit the batch jobs that invoke the print program ADMUCDSD. You can create and submit the job yourself. Some sample JCL for creating a primary data stream for a 38xx printer is shown below.

```
1)  * $$ JOB JNM=jobname,CLASS=x,DISP=y
2)  * $$ LST CLASS=x,DISP=y,DEST=(node,userid), *
             JSEP=1
3)  * $$ LST CLASS=x,DISP=y,DEST=(,psfid),      *
             LST=cuu,JSEP=1
4)  // JOB jobname
5)  // DLBL libname,'name.of.a.library'
5)  // EXTENT ,volid
6)  // LIBDEF *,SEARCH=(l1.s1,l2.s2,...,li.si)
7)  // DLBL IJSYSUC,'user.catalog.name',,VSAM
8)  // DLBL ADMF,'gddm.objects.file.name',,VSAM
9)  // ASSGN SYSyyy,cuu
10) // EXEC ADMUCDSD,SIZE=ADMUCDSD,             *
            PARM='filename filename 99 4 token  *
            (procopts) (SYSyyy)'
    /*
    /&
    * $$ EOJ
```

1. This is the POWER JOB statement for the GDDM batch job.
2. This is the POWER LST statement for the SYSLST output.
3. This is the POWER LST statement for the primary data stream to go to the 3800 printer at address cuu.
4. Job name card.
5. DLBL and EXTENT statements for every library named in the LIBDEF statement. EXTENT only, if the library is not VSAM controlled.
6. Search chain for all the libraries you want to read from, that is the library containing GDDM.
7. DLBL for the user catalog (contains ADMF).
8. DLBL for GDDM objects file.
9. This assignment statement links the programmer logical unit SYSyyy to the device cuu, the spooled 3800 printer.
10. EXEC card for ADMUCDSD.

To write secondary data stream to the VSE phase library, a few changes are necessary in the above job stream. The differences are in the way that PSF is invoked.

A `LIBDEF *,CATALOG=lib.sulib` statement must be included for the library to which you write the page segment. The name of phase written to replace SYSyyy on the EXEC card, and the processing options, must be altered to specify a secondary data stream (replace 0 with 1 in the processing options). Some example JCL for printing a primary data stream on the 4250 is shown below.

```
1)  * $$ JOB JNM=jobname,CLASS=x,DISP=y
2)  * $$ LST CLASS=x,DISP=y,DEST=(node,userid),    *
          JSEP=1
3)  // JOB jobname
4)  // DLBL lib1,'cdpf.library.name'
4)  // EXTENT ,volid1
4)  // DLBL lib2,'font.library.name'
4)  // EXTENT ,volid2
5)  // LIBDEF *,SEARCH=(lib1.sublib1,lib2.sublib2)
6)  // DLBL usercat,'user.catalog.name',,VSAM
7)  // DLBL INPUT,'print.file.name',,VSAM,         *
          CAT=usercat,DISP=(OLD,DELETE)
8)  // EXEC BFUCDPF,SIZE=AUTO,                      *
          PARM='PRINT (BRACKET vtam_printer_name)'
    /*
    /&
    * $$ EOJ
```

1. This is the POWER JOB statement for the GDDM batch job.
2. This is the POWER LST statement for the SYSLST output.
3. Job name card.
4. DLBL and EXTENT statements to define the libraries containing CDPF and the FONTLIB.
5. Search chain for the libraries containing CDPF and the fonts.
6. DLBL statement for the user catalog.
7. DLBL statement for the print file.
8. EXEC card for CDPF.

Note that the DELETE option on the DLBL statement for the print file means that the input file is deleted after it is printed. Use the option KEEP if you want to keep the print file.

IMS/VS print utility

The IMS/VS version is ADMOPUI, and runs as either a message-processing program or batch message program (BMP).

Note: The IMS/VS print utility does not support plotters.

Invocation

No explicit invocation is required if the transaction is defined as a message-processing program. Standard JCL is required to initiate the transaction as a BMP.

Messages

Most error messages are issued as part of the output from the utility. If it is not possible to send the error message to the LTERM for which the output is destined, the utility issues a /BROADCAST MASTER command containing the error message.

TSO background print utility

Under TSO, there are two methods of printing GDDM files available to the user:

- By means of a queue of requests (the ADMPRINT queue), which is serviced by the ADMOPUT print utility
- By means of the JES/328X Print Facility Version 2 Release 2 Modification 0, and the ADMOPUJ print utility.

Note: Throughout both volumes of the *GDDM Base Programming Reference*, references to JES/328X indicate the **JES/328X Print Facility Version 2 Release 2 Modification 0**, unless stated otherwise.

The user has complete control over which of these methods to use, and can use both.

Using the ADMPRINT queue means using a master print queue, which consists of pointers to print datasets. These print datasets are created by GDDM when the user requests a print.

In turn, the GDDM ADMOPUT print utility runs as a Batch Job servicing this queue, and performing the print.

With GDDM/MVS Version 2, another method is available that has no need for the ADMPRINT queue.

This method entails the installation of the JES/328X Program Offering – Version 2 Release 2 – which interfaces directly to JES2 or JES3.

With this method, when the user requests a print, the print dataset created is written directly to the JES Spool.

JES passes these requests to the JES/328X print utility, which runs as a Batch Job, and JES/328X passes each GDDM Print request to a GDDM program, ADMOPUJ, that performs the actual printing.

While these print datasets are within the JES Spool, they can be manipulated like any other JES Spool file, thus giving the installation greater control over the print requests.

The PRINTDST processing option is used to determine the destination of the print request and thus which method of printing is to be used.

The three possible destinations are:

- The existing ADMPRINT queue destination

  This sends the print request to the ADMPRINT queue for processing by ADMOPUT (as described below).

  This can be done by using the PRINTDST processing option in a nickname statement containing:

  ```
  PROCOPT=((PRINTDST,*,*))
  ```

  which is the default.

- Directly to JES

  This sends the print request directly to the JES Spool for processing by JES/328X and ADMOPUJ (as described below).

  This can be done by using the PRINTDST processing option in a nickname statement containing:

  ```
  PROCOPT=((PRINTDST,class,destname))
  ```

  where `class` is the system-defined class for punch output, and `destname` is the JES Remote Work Station destination name.

- To an intermediate user dataset for subsequent processing

  This sends the print request to a dataset for later processing, either by means of Batch JCL, or the JES/328X DSPRINT command.

  This can be done by using the PRINTDST processing option in a nickname statement containing:

  ```
  PROCOPT=((PRINTDST,*,ddname))
  ```

  where `ddname` is a previously allocated ddname.

The ADMPRINT utility is described in detail in the next section, and JES/328X is described on page 53.

The ADMPRINT print utility

The ADMPRINT print utility has the name ADMOPUT. It manages one or more printers, and uses the VTAM interface to communicate with them.

Print requests for each device are processed in the order in which they are received, and the queue of print requests is maintained on a direct-access device to avoid the loss of output should a system failure occur.

A printer can be left unattended, with only an occasional check on the paper supply. Output from each print request is usually preceded by a descriptive header (printed on a separate page) for identification. Thus, one printer can be shared efficiently by a number of terminal operators, and interactive application programs need not wait until printing is completed.

Printing alphanumeric files

The GDDM Sequential File Print Program (ADMOPRT) allows files to be printed that contain alphanumeric data. ADMOPRT converts a sequential file into a graphics print file. The GDDM print utility then prints it.

The syntax of the command that calls the GDDM Sequential File Print Program is:

```
CALL 'data-set-name(ADMOPRT)' 'file-name ON
     printer-name [ (NOCC ]'
```

where:

data-set-name
is the name of the data set into which ADMOPRT was installed.

file-name
is either a ddname allocated to the data set to be printed or (if such a ddname is not present) the name itself of the data set to be printed.

If **file-name** represents a data set name, it must be entered using normal TSO naming conventions; in this case, ADMOPRT does **not** support a data set name that represents a member of a partitioned data set.

| **Note:** From Version 2 Release 2 of GDDM, ADMOPRT uses the values given by the PRINTCTL processing option for the number of characters per line and the number of lines per page. Previously it used values of 132 and 66 respectively. To obtain the same results as under previous releases, use a nickname statement that specifies `PROCOPT=((PRINTCTL,1,1,66,0,0,0,132))`.

ON
is a required keyword that must be specified before the name of the printer.

printer-name
is the name of the queued printer on which the file is to be printed.

NOCC
indicates that any existing carriage-control characters are to be ignored. If NOCC is not specified, the presence or absence of carriage-control characters is determined by ADMOPRT according to the record format of the input file.

Carriage-control characters are processed as described for FSLOGC in the *GDDM Base Programming Reference, Volume 1*.

Deleting a print request

A print request can be purged, if it is not in the process of printing, by deleting the OS data set representing the request. If this happens, the utility prints a diagnostic noting that the request was deleted, and it then proceeds with any other requests. All TSO request data sets are cataloged with the user-nominated data set name prefix (by default, the userid) as a first qualifier, and the request sequence number as a last qualifier. The TSO LISTC command can, therefore, be used to identify the names and order of pending print requests for a specific userid.

When a request has started to print, it may not be possible to delete the corresponding OS data set, because this would require exclusive (DISP=OLD) access. In this case, the output can be canceled at the printer, as described under "Canceling printer output" below.

Some types of errors prevent a request from proceeding to the point where it can be canceled in this manner, but they can still let the Print Utility retry the request at regular intervals. In this case, exclusive access to the request data set (for the purpose of deletion) can be acquired by first stopping the utility as described under "Invocation." It should then be possible to delete the request data set, before restarting the utility.

Printer and plotter operating instructions

The printer or plotter must be prepared for use according to the appropriate operating instructions for the model, and the paper must be aligned to the top of the page.

Canceling printer output

Printer output can be canceled after it has started, by switching the printer off and on at least three times during the printing of a single page. After each power-on, the utility tries to reprint the interrupted page. The printer must be powered off during the reprinting of this page to maintain the cancelation sequence.

For a printer operating in SCS mode, the CANCEL PRINT switch can be used, as described below.

Canceling plotter output

Plotter output can be canceled by pressing the CLEAR key on the associated 3179-G or 3192-G color display station, 3270-PC/G or 3270-PC/GX work station, or device supported by GDDM-PCLK.

SCS mode PA and CANCEL PRINT switches

For a printer operating in SCS mode, the PA1/PA2 and CANCEL PRINT switches can be used. These switches allow limited communication with the utility, and are used with the Hold Print/Enable Print switch, as described in the Component Description and Operator's Guide for the appropriate printer. The effect of each switch is as follows:

PA1	Sending a PA1 switch code to the utility causes it to restart printing of the current request at the page after the header page. For a multiple-copy request, printing is resumed at the start of the copy being processed at the time of the interrupt.
PA2	Sending a PA2 switch code to the utility causes it to restart printing of the current page.
CANCEL PRINT	Sending a CANCEL PRINT switch code to the utility causes it to cancel printing of the current request.

Invocation

Before the utility is run, the print queue data set must be initialized to the format described in the *GDDM Installation and System Management for MVS* manual.

The devices to be used can be activated before the utility is started. Under VTAM, this is achieved by commands:

```
VARY NET,ACT,ID=printername
```

If a device is activated **after** the utility is started, the utility may have to be notified that the device is available. This can be done explicitly by using the LOGON operand when the device is activated, as follows:

```
VARY NET,ACT,ID=printername,LOGON=applname
```

where "applname" is the Print Utility's VTAM application name (usually ADMPRINT). This command can be entered to notify the print utility, even if the device is already active.

Or, the VTAM network can be defined such that the print utility is automatically notified when a device is activated. This is done by nominating ADMPRINT (or applname) as the controlling application by the LOGAPPL parameter on network terminal definition macros.

The utility is started by submitting the job-control statements listed below, modified as necessary (for example, to respecify the library on which the utility is kept).

The utility can be started either as a submitted batch job or as a started task. The example below gives the alternative statements for doing this.

The job control statements are usually held in SYS1.PROCLIB. They take the form:

FOR STARTING AS A BATCH JOB

```
//applname  EXEC PGM=ADMOPUT,DYNAMNBR=n,REGION=mK,
//               PARM='NAME=xxxx,AUTO,MAXPRTRS=nnn'
//STEPLIB    DD  DSN=
//ADMSYMBL   DD  DSN=
//ADMGGMAP   DD  DSN=             [ optional ]
//ADMPRNTQ   DD  DSN=             [ optional ]
//ADMDEFS    DD  DSN=             [ optional ]
//SYSABEND   DD  SYSOUT=A
```

FOR STARTING AS A STARTED TASK

```
//        PROC PGM=ADMOPUT,DYNAMNBR=n,REGION=mK,
//             PARM='NAME=xxxx,AUTO,MAXPRTRS=nnn'
//STEPLIB    DD  DSN=
//ADMSYMBL   DD  DSN=
//ADMGGMAP   DD  DSN=             [ optional ]
//ADMPRNTQ   DD  DSN=             [ optional ]
//ADMDEFS    DD  DSN=             [ optional ]
//SYSABEND   DD  SYSOUT=A
```

The VTAM application name ("applname") used by the print utility must be defined by the APPL macro in the VTAM network definition. The name is usually ADMPRINT but this can be changed if required. "applname" is derived as follows:

- For a batch job that does **not** use a cataloged procedure, "applname" is taken from the job step name.
- For a batch job that uses a cataloged procedure, "applname" is taken from the procedure step name.
- For a started task with **no** task identifier specified in the START command, "applname" is taken from the member name in the procedure library of the started-task JCL.
- For a started task with a task identifier specified in the START command, "applname" is taken from the task identifier.

The DYNAMNBR parameter should specify at least one more than the number of printers to be operated at the same time. Note that this does not impose a limit on the number of printers that can be run at the same time; that is done by the MAXPRTRS parameter. The region size should be calculated as described in the *GDDM Installation and System Management for MVS* manual.

If you have a large number of printers defined to the print utility, you do not have to specify a region size sufficient for that number. Instead, you can limit the number of printers that ADMOPUT services at the same time using the MAXPRTRS parameter, described below. ADMOPUT processes work for printers that would otherwise be beyond the MAXPRTRS limit after work for other printers has completed.

After the PARM parameter, any combination of these optional parameters can be specified (in any order):

NAME=name
indicates that the name specified is to be used in any messages to the operator to identify which instance of ADMOPUT issued the message. If the NAME parameter is not specified, the VTAM application name is used.

AUTO
indicates that no outstanding reply to the operator is generated. ADMOPUT is terminated automatically when there are no more active print subtasks running.

MAXPRTRS=nnn
indicates the maximum number of printers that the print utility tries to operate at any one time.

The STEPLIB DD statement specifies the data set on which the GDDM load library resides.

The ADMSYMBL DD statement specifies the data set on which symbol sets reside.

If at the time the print request was made, named symbol sets had been identified for use through one or more of the GDDM statements GSDSS, GSLSS, PSDSS, PSLSS, or PSLSSC (see the *GDDM Base Programming Reference, Volume 1*) the TSO user requesting the printer must ensure that the symbol sets required reside in the data set identified by the ADMSYMBL DD statement.

The ADMGGMAP DD statement specifies the data set on which IMD-generated mapgroups reside. This statement is required if the user's application programs issue FSCOPY against mapped pages.

The ADMPRNTQ DD statement is optional, and, if supplied, is taken to identify the Master Print Queue data set. If this DD statement is omitted, GDDM dynamically allocates to the data set:

```
'xxxxxxxx.REQUEST.QUEUE'
```

where xxxxxxxx (usually ADMPRINT) is defined in the current GDDM external defaults (for the details, see "Changing GDDM's default values" on page 127). This may be modified by the installation, if desired.

The ADMDEFS DD statement is optional; if supplied, it identifies an External Defaults File. This is described under "External defaults file" on page 2. For the print utility, you are recommended to use a **data set** as an External Defaults File, and not inline data (that is, not //ADMDEFS DD ★). In some operating environments, the implementation of inline data is such that it may be read and acted upon only by the first of the many subtasks that ADMOPUT uses.

If AUTO is not specified, this message is received at the system console:

```
ADM2000 I ADMOPUT(instance-name). TO TERMINATE,
              REPLY 'STOP', 'STOPQ', OR 'STOPS'
```

The system operator can cause ADMOPUT to terminate printing by replying STOP, STOPQ, or STOPS; the effect of each of these replies is:

STOP ADMOPUT terminates when all requests in process have been completed.

STOPQ ADMOPUT terminates immediately. (The current requests are restarted when ADMOPUT is next initialized.)

STOPS If this is entered, GDDM issues the message **ADM2019**, which gives the operator the choice of either

- Entering STOPQ, causing ADMOPUT to stop immediately, or
- Ignoring this message, in which case ADMOPUT continues to run until all requests in process have been completed.

Thus, the utility terminates when VTAM is halted, when a request is made to terminate by a reply to message ADM2000, or if AUTO is specified and there are no more active print subtasks running.

Messages

Messages issued by the TSO version of the print utility are numbered from ADM2000. These messages are described in the *GDDM Messages* manual.

JES/328X

The JES/328X Print Facility Program Offering Version 2 Release 2 Modification Level 0 extends the support of Remote Job Entry (RJE) devices provided by MVS JES2 and JES3 to include the wide range of 3270 printers, or printers that are compatible with them, and the family of IPDS (Intelligent Printer Data Stream) printers. JES stands for Job Entry Subsystem.

Output can be routed to these printers by :

- JCL
- The TSO ALLOCATE command
- The JES/328X SYSOUT command
- The JES/328X DSPRINT command.

Because all output can be spooled by JES, the JES operator commands can be used to provide such functions as rerouting, queue reordering, and output cancelation.

JES/328X does not require any changes to JES code. It operates as a VTAM secondary application when communicating with JES, and appears to JES as a Remote Work Station. Having received the data from JES, JES/328X then operates as a primary VTAM application when communicating with the printer.

When processing GDDM print requests (or files destined for IPDS printers) JES/328X invokes the GDDM print utility ADMOPUJ, to process the print request.

The installation and operation of JES/328X is fully described in the *JES/328X Print Facility Program Description and Operators Manual.*

An overview of this process is contained in the *GDDM Installation and System Management for MVS* manual.

Usage

As mentioned above, GDDM (by means of the PRINTDST processing option) now provides the ability to send a print request either into the JES Spool for processing by JES/328X, or to a file. Using the second method, the user can process the print request in a number of ways:

- Transfer the file to VM for processing there.
- Send the file to JES for JES/328X processing, by means of either the JES/328X DSPRINT command, or JCL.
- Send the file directly to the printer, by means of the JES/328X DSPRINT command.

While the request is in the JES Spool, it can be manipulated by the usual JES operator commands. For example, the request can be rerouted to another destination, it can be canceled, or its place in the queue can be reordered. Also, JES/328X provides ISPF panels to enable the user to issue some JES2, JES3, or JES/328X commands.

Full details are given in the *JES/328X Print Facility Program Description and Operators Manual.*

Examples

The following examples illustrate the various methods of printing available.

They assume that:

- JES/328X has been installed.
- The JES/328X DSPRINT command has superseded the TSO DSPRINT command.
- Remote destinations RMT1 and RMT2 have been defined to JES with the printers serving Class P and the punches serving Class G.
- Remote destinations RMT1 and RMT2 have been defined to VTAM.
- Remote destinations RMT1 and RMT2 have been defined to JES/328X, as follows:
  - RMT1 is a 3287 device with an LUNAME of L870, and GDDM is called to process all requests for CLASS G (PASSTHRU=JSXGDDM and CLASS=G specified)
  - RMT2 is an IPDS device with an LUNAME of L890 (data destined for IPDS devices automatically causes GDDM to be invoked provided PASSTHRU=INTERNAL is specified).
- The TSO user's ADMDEFS file has these entries:

```
NICKNAME NAME=R187,FAM=2,
         PROCOPT=((PRINTDST,G,RMT1))
NICKNAME NAME=R2IP,FAM=2,DEVTOK=X4224SE,
         PROCOPT=((PRINTDST,G,RMT2))
NICKNAME NAME=DA87,FAM=2,
         PROCOPT=((PRINTDST,*,DD87))
NICKNAME NAME=DAIP,FAM=2,DEVTOK=X4224SE,
         PROCOPT=((PRINTDST,*,DDIP))
```

- Invocations of the Interactive Chart Utility use the above ADMDEFS file, and have DDNAMES DD87 and DDIP allocated to datasets USER.GDDMPRT.DATA and USER.IPDSPRT.DATA respectively.
- The user has a dataset USER.PRINT.DATA containing alphanumeric data (specifically the dataset does **not** have carriage-control set).
- The JES/328X Print Facility Job is running.

The order of events is:

1. The Chart Utility is invoked and a print is created for R187:

   This sends the print request (formatted for a 3287) directly into the JES Spool, where it is passed to JES/328X. JES/328X, in turn, passes the request to GDDM for printing.
2. The Chart Utility is invoked and a print is created for R2IP:

   The same as above, but the print request is formatted for an IPDS printer.
3. The Chart Utility is invoked and a print is created for DA87:

   This outputs the print request (formatted for a 3287) to the dataset USER.GDDMPRT.DATA.

4. The Chart Utility is invoked and a print is created for DAIP:

   This outputs the print request (formatted for an IPDS printer) to the dataset USER.IPDSPRT.DATA.

In the first two examples, the print requests are sent automatically through the system to GDDM for printing. However, in the last two examples, the user now has a choice of methods available to process the requests. The user can:

- Transfer the files to VM for processing.

  Because the format of the print request is subsystem-independent, the print request can be processed on either VM or TSO.

- Send the files through JES for printing using JES/328X and GDDM.

  The following commands will do this:

  ```
  DSPRINT 'USER.GDDMPRT.DATA' RMT1 CLASS(G) NONUM
  DSPRINT 'USER.IPDSPRT.DATA' RMT2 CLASS(G) NONUM
  ```

  These commands result in message DSP012, indicating that the print requests have been sent to JES for subsequent processing.

- Print the files directly using JES/328X and GDDM, thereby bypassing the JES Spool.

  This facility is particularly useful where data of a confidential nature is being printed and the user does not want the print request to be sent by way of the "public" spool facility. Typically, the user is not far from the printer and can thus ensure that the print is collected as soon as it is finished.

  The following commands will do this:

  ```
  DSPRINT 'USER.GDDMPRT.DATA' L870 GDDM
  DSPRINT 'USER.IPDSPRT.DATA' L890 GDDM
  ```

  These commands result in a foreground print operation, the end of which is signaled by message DSP040, indicating that the print has been completed.

  Because this is a foreground process, the user's terminal is "locked out" for the duration of the print request. Also, any symbol sets required for printing have to be allocated by the user before the JES/328X DSPRINT command.

Printing alphanumeric files

As with printing GDDM files, the user has two methods of printing files containing alphanumeric data:

1. By sending the files through JES for printing using JES/328X (and GDDM if the printer is an IPDS device).

   The following commands will do this:

   ```
   DSPRINT 'USER.PRINT.DATA' RMT1 CLASS(P) NONUM
   DSPRINT 'USER.PRINT.DATA' RMT2 CLASS(P) NONUM
   ```

   These commands result in message DSP012, indicating that the print requests have been sent to JES for subsequent processing.

2. By printing the files directly using JES/328X and GDDM.

   The following commands will do this:

   ```
   DSPRINT 'USER.PRINT.DATA' L870 GDDM NONUM
   DSPRINT 'USER.PRINT.DATA' L890 GDDM NONUM
   ```

   These commands result in a foreground print operation, the end of which is signaled by message DSP040 indicating that the print has been completed.

Common errors

ADM0244 E INVALID PRINT RECORD SEQUENCE
This message appears on the print request if GDDM has been called to process invalid data. A common way for this to occur is to send alphanumeric data through JES to JES/328X using the class defined for GDDM files.

```
DSPRINT 'USER.PRINT.DATA' RMT1 CLASS(G) NONUM
```

The above command sends the alphanumeric data through JES to JES/328X as CLASS G output. JES/328X insists that GDDM data has carriage-control set on, and ignores all records without carriage-control. If this should happen this message is sent to the operator console:

```
JSX209 - NON-GRAPHICS RECORDS IGNORED
```

and an empty file is sent to GDDM. GDDM then sends message ADM0244 to the printer; for more information, refer to the *GDDM Messages* manual.

Interfaces

GDDM-to-JES

GDDM uses SVC99 Dynamic Allocation services to output a print request directly into the JES Spool with the CLASS and DEST parameters set from the PRINTDST Class and Destname values respectively.

This is the one case where GDDM creates an Output Print request **without** carriage-control; this is so the data can be correctly passed through to GDDM.

However, when sending the Output Print request to a dataset, GDDM sets the carriage-control indicator on in the dataset — even if it is a pre-allocated dataset.

DSPRINT caters for this automatically, but if the user wants to route the data through JES by some other means, for example by an IEBGENER process, the RECFM will need to be overridden:

```
//*
//*  ** SEND GDDM DATA THROUGH JES TO JES/328X
//*
//GENER1    EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT2    DD   SYSOUT=G,DEST=RMT1,
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//SYSUT1    DD   DSN=USER.GDDMPRT.DATA,DISP=SHR
//SYSIN     DD   DUMMY
```

JES/328X-to-GDDM

The GDDM–JES/328X utility ADMOPUJ is called from the JES/328X special exit JSXGDDM, to either process a print request or, at initialization and termination time, to OPEN and CLOSE the VTAM ACB.

JSXGDDM passes this parameter list, the address of which is in Register 1:

JXEACBN	address of an 8-byte area containing the VTAM ACB name
JXEDEST	address of an 8-byte area containing the VTAM LUname of the printer destination
JXEDSN	address of a 44-byte area containing the ddname of the GDDM input print file
JXECOMM	address of a 4096 common area
JXESHUT	address of the full-word JES Shutdown ECB
JXEMSG	address of 160-byte Return Message area.

The JXEDSN ddname has two special values, namely OPEN and CLOSE; these indicate initialization and termination requests respectively.

The Common area is used as a work area by GDDM.

The JES shutdown ECB enables GDDM to detect a shutdown request.

The Message area enables GDDM to inform JES/328X of any errors detected in printing.

VM/CMS print utility

Under VM/CMS, the print utility is named ADMOPUV; it controls a printer or plotter attached to the invoking virtual machine or it can send the printer data stream as a punch file to another destination for processing (such as RSCS Networking Version 2). The printer may have been explicitly attached by the system operator or authorized user using the CP ATTACH command, or may be automatically attached at logon by its inclusion in the directory of the invoking virtual machine.

A print request by a GDDM subroutine is created as a file on the user's A-disk. The file is subsequently printed by running ADMOPUV, specifying the file and the virtual address of the printer or plotter as parameters.

If the DSOPEN processing option (INVKOPUV,YES) was specified, function equivalent to that performed by ADMOPUV is called automatically, after which the print file is erased. Otherwise, if the installation has provided an ADMQPOST EXEC, GDDM calls this after creating the print file. An ADMQPOST EXEC can be used, typically, to send the print file to a separate, possibly automatic, virtual machine for processing. For details of this facility, see the *GDDM Installation and System Management for VM* manual.

Invocation

The utility is started by this command:

```
ADMOPUV filename [filetype [filemode]] [ON cuu]
        [([CC | NOCC] / [DEV dev-token])]
```

where optional parameters are indicated by [...]. The meanings of the parameters are:

filename
is the name of the file to be printed. It must be specified.

filetype
is the type of the file to be printed. If this parameter is not specified, ADMPRINT is used as the default (unless a different filetype was specified in the current GDDM external defaults).

filemode
is the mode of the file to be printed. If this is not specified, "*" is the default.

cuu
can be used to specify the printer device name or address. If **cuu** is omitted, the device is identified by the STAGE2ID processing option group (18 in DSOPEN) specified when the print file was created. If this processing option group was not specified, the default is 061.

CC
(the default) interprets the first character of each record as a carriage-control character.

NOCC
interprets the first character as part of the data. Carriage-control characters are processed as described for FSLOGC; see the *GDDM Base Programming Reference, Volume 1*.

DEV
indicates that a device token is to be specified.

dev-token
can be used to override the device characteristics that GDDM usually infers. It identifies a device characteristics token, as defined in Appendix G, "Device characteristics tokens" on page 203.

Printing alphanumeric files

As well as printing files that were created under GDDM, ADMOPUV lets you print files that contain alphanumeric data.

Printing GDDM files through RSCS

If the Remote Spooling Communication Subsystem (RSCS) Networking Version 2 (under VM/SP Release 4) is available at your location, output can be directed to a printer connected to RSCS.

The way to do this is to specify the special device name "PUNCH" in the ADMOPUV command, as a result of which GDDM writes 3270 device output to the virtual punch. If the virtual punch is spooled to RSCS, and is suitably tagged, RSCS prints the device output on the required printer.

A **nickname** of the following form can be used to encapsulate the necessary device, spool, and tag information:

```
        ADMMNICK FAM=1,NAME=prt-name,
                 TONAME=PUNCH,DEVTOK=L87,
                 PROCOPT=((CPSPOOL,TO,RSCS),
                 (CPTAG,node,prt-name,50,PRT=GRAF))
```

and enables the ADMOPUV command to be entered simply as:

```
ADMOPUV filename ON prt-name
```

Notes:

1. For a full discussion on nicknames, see "Using nicknames to define device characteristics" on page 3.
2. For a full explanation of the CPTAG string, see *VM RSCS Networking Version 2: Operation and Use* manual.
3. Specify a device token corresponding to the printer to be used.
4. Printing on IPDS printers requires RSCS Version 2 Release 2.

Automatically initiating the VM/CMS print utility

The DSOPEN processing option, INVKOPUV, can be used to cause function equivalent to that performed by ADMOPUV to be called automatically, whenever a print file is created. For full details, see the *GDDM Base Programming Reference, Volume 1.*

A **nickname** can be used to get the same effect, but without the need for changing the application program. For example,

```
        ADMMNICK FAM=2,NAME=prt-name,
              PROCOPT=((INVKOPUV,YES))
```

This nickname could effectively be used together with a nickname set up to direct printer output to RSCS, as described above.

An ICU user could use these nicknames to cause the ICU print panel to initiate asynchronous printing on a printer attached to RSCS.

Printer and plotter operating instructions

The printer or plotter must be prepared for use according to the appropriate operating instructions for the model, and the paper must be aligned to the top of the page. This should be done before the ADMOPUV command is issued.

Messages

Messages generated by the VM/CMS version of the printer utility are numbered from ADM2101, and are described in the *GDDM Messages* manual.

Nonqueriable printers with the APL feature

By default, under VM/CMS, the GDDM Print Utility assumes that any APL feature installed on an IBM 3270 printer is the APL/Text Feature. This default must be overridden if printer output containing APL characters is to be directed to a printer, such as an IBM 3284 or 3286, with the Data Analysis – APL Feature.

The CMSAPLF option in GDDM's external defaults can be modified (by specifying the value DATAANAL) to cause GDDM to assume by default that an APL feature installed on a nonqueriable IBM 3270 printer is the Data Analysis – APL Feature. This option can be specified:

- In an External Defaults Module, or
- In an External Defaults File.

See Chapter 1, "Customizing your program and its environment" on page 1.

Image Print Utility

The Image Print Utility creates page printer files from GDDM image (ADMIMG) files. It can be run under CMS, MVS (including TSO), and VSE (in batch mode). The entry name and parameter format depend on the environment:

CMS
: ADMUIMPV
`('name')('scale')('prtname')('procopts')`
`('token')('field')`

MVS
: ADMUIMPT
`'name(scale)prtname(procopts)token(field)'`

VSE
: ADMUIMPD
`'name(scale)prtname(procopts)token(field)'`

The meanings of the six parameter groups are shown below. All parameters except groups 1 (`name`) and 3 (`prtname`) are optional. The utility will assign default values to empty groups.

name
Name of image file to be printed:

Under CMS:	`filename filetype filemode`
Under TSO:	`ddname`
Under VSE:	`dlbl`

scale
Scaling control:

0	Field to fit image: dimensions in `field` ignored Forced if `fieldh` or `fieldv` = 0.
1	Original image size: image may be clipped or spaced (default).
2	Scale to fit field: image may be distorted.
3	Re-scale with same aspect ratio: image may contain blank space.

prtname
Name to be assigned to print file:

Under CMS:	`filename filetype filemode`
Under TSO:	`ddname`
Under VSE:	`dlbl`

procopts
: Processing options (procopts) to be used in DSOPEN for printer.

token
: Printer device token. Default is `IMG240`.

field
: Image field size, in this format:

Under CMS:	`fieldh fieldv fields`
Under TSO:	`fieldh,fieldv,fields`
Under VSE:	`fieldh,fieldv,fields`

where:

`fieldh` =	Horizontal dimension (default 0)
`fieldv` =	Vertical dimension (default 0)
`fields` =	Units for above: `0` = tenths of inches (default); `1` = millimeters.

Sample code invoking these modules is supplied with the GDDM Base programs, as follows:

- ADMUIMP, a REXX-language CMS procedure that invokes ADMUIMPV. It will produce a file that can be printed on a 3800 Model 3 printer as a page segment to fit an area 6 inches wide and 4 inches deep. The procedure can be amended to suit your installation and users. Normally, you should choose the device token and type (document or page segment), and select the most appropriate output file type (by assigning the required value to `outft`).
- ADMUJT10, sample MVS JCL that invokes ADMUIMPT.
- ADMUJD10, sample VSE JCL that invokes ADMUIMPD.

Composite Document Print Utility

The utility is invoked by a call to CDPU. This function is part of the GDDM Base programming interface, and the CDPU call must be part of a GDDM application program. You can either write such a program yourself, or use a ready-made program called ADM4CDUx (where x is subsystem-dependent) supplied with GDDM. This is described in "ADM4CDUx" on page 58.

Two sets of PL/I DECLARE statements for GDDM functions are provided. These will be useful if you write a CDPU-calling program in PL/I. They are:

ADMUPINK – calls starting CD...for nonreentrant use.
ADMUPIRK – calls starting CD...for reentrant use.

For more details about GDDM-supplied PL/I DECLARE statements, refer to the *GDDM Base Programming Reference, Volume 1*.

See Appendix L, "Format of a Composite Document Presentation Data Stream" on page 255 and "AFPDS structured fields supported by the CDPU" on page 63 for more information about the format of CDPU input files.

Example program

The program shown creates an AFPDS file from a CDPDS file. By omitting the DSOPEN and DSUSE calls, the program can be used to view a document on CICS, MVS/TSO, or CMS.

```
SAMPLE: PROCEDURE OPTIONS(MAIN);

    /* DECLARE GDDM ENTRY POINTS */
 %INCLUDE ADMUPIND;  /* NAMES BEGINNING D... */
 %INCLUDE ADMUPINF;  /* NAMES BEGINNING F... */
 %INCLUDE ADMUPINK;  /* NAMES BEGINNING CD.. */

    /* OTHER DECLARES  */
 DCL DEVID  FIXED BIN(31) INIT(11);
 DCL FAMILY FIXED BIN(31) INIT(4);
 DCL DEVTOK CHAR(8) INIT('A4');
 DCL IN(1)  CHAR(8) INIT('CDPIN');
 DCL OUT(1) CHAR(8) INIT('CDPOUT');
 DCL NONE(1) FIXED BIN(31);   /* DUMMY ARRAY */

    /* INITIALIZE GDDM  */
 CALL FSINIT;

    /* OPEN THE DEVICE  */
 CALL DSOPEN( DEVID, FAMILY, DEVTOK, 0, NONE,
    1, OUT);
 CALL DSUSE( 1, DEVID);

    /* PRINT THE DOCUMENT */
 CALL CDPU( 1, IN, 0, NONE);

    /* TERMINATE GDDM */
 CALL FSTERM;
 END SAMPLE;
```

Application control

If an application calls the CDPU with the view control parameter set to a nonzero value, the application can control how the document is browsed. The CDPU creates a GDDM page containing the specified document page, but does no input or output. The application must issue its own ASREAD (or other input/output call) and interpret the returned values. Additionally, the application can:

- Define a graphics field for the document page to be shown in. The default is a field covering the whole screen.
- Display instructions to the user.
- Test for requests for document pages beyond the document end.

Specifying the device

The output from the CDPU goes to the primary device specified by the DSOPEN and DSUSE calls. The DSOPEN specification can be modified without changing the application by using GDDM nickname statements.

The device can be a directly connected (in GDDM terms, family-1) graphics or IPDS printer, a queued (family-2) graphics or IPDS printer, a page (family-4) printer, or the user's terminal (a family-1 device).

When the CDPU call is executed, the CDPU checks the type of primary device that the application program has opened, and generates the appropriate data stream.

The default primary device is the terminal, which is why the CDPU displays the document at the terminal if the DSOPEN and DSUSE calls are omitted.

If the device is family-2 or -4 printer, an intermediate print file is created. Its name is taken from the **name-list** parameter of the DSOPEN call. If a file with the same name already exists, it is deleted without warning.

Running the CDPU application program

The application program that calls the CDPU, whether it is ADM4CDUx or a user-written program, can be executed under CMS, TSO, or CICS, or in MVS or VSE batch mode. You will need to supply any necessary commands or JCL to invoke it.

A sample CMS REXX-language procedure, ADMUBCDV, and TSO command list (CLIST), ADMUBCDT, are supplied with GDDM/VM, GDDM/VMXA, and GDDM/MVS as appropriate. These call ADM4CDUx to browse a file at the terminal.

Other CMS example procedures are shown on pages 59 and 60. The first sends the document to a 4224 printer, and the second sends it to a 38xx AFPDS printer. The first is similar to ADMUBCDV, except that the `devtok` and `namelist` variables are set to printer values.

In all cases – the ADMUBCDx samples in addition to the procedures shown here – the document must be in the form of a CDPDS or AFPDS file, the filename and, optionally, filetype and filemode, of which, are passed as parameters to the procedure.

ADM4CDUx

The entry name of the ADM4CDUx program varies depending on the environment:

CICS	ADM4CDUC
VSE Batch	ADM4CDUD
MVS Batch	ADM4CDUT
TSO	ADM4CDUT
VM/CMS	ADM4CDUV
CMS/XA	ADM4CDUX.

ADM4CDUx can be invoked by the user to process and print the CDPDS or AFPDS. Under CICS, MVS/TSO, and CMS, it can also be used to view CDPDS or AFPDS files. The parameter list, described below, is read as up to six groups, each group being separated by a blank or comma. All groups are optional, with default values. Groups are positional, and may be empty.

In the CICS environment, the standard transaction identifier is ADM4. This name may have been changed at your installation. The parameter list is specified in the **from** option of the CICS START.

The parameter groups are:

1. Composite document presentation data stream identifier

   The requirements and defaults for this parameter are the same as for the **cd-name** parameter of the CDPU call (see *GDDM Base Programming Reference, Volume 1*).

2. Printing options corresponding to those described for the CDPU call:

   a. Number of uncollated copies of document

   b. Duplex control

   c. View control.

   The default values are the same as for the CDPU call.

The remaining groups correspond to parameters on a call to DSOPEN:

3. Device name list

   There is no default value – the parameter must be specified explicitly – except when the program is being run under CMS and family-4 is specified (or defaulted) in parameter 6. In that case, the default name is the same as the input file name.

4. Device processing options, as a sequence of decimal numbers

   The default is none.

5. Device token

   The default is S4224QE (SNA-attached 4224) under CICS, or A4 (38xx AFPDS printer) under other systems.

6. Device family

   The default is family-1 under CICS, or family-4 under the other subsystems.

A simple parameter list to print a CDPDS file called DOC, on a 38xx page printer defined by the device token IMG240, using 20 swathes, would be:

```
(DOC) ( ) ( ) (7 20) (IMG240)
```

```
/* Name : CD42SAMP - sample exec  (4224 printer)                    */
/* This is a sample user exec that takes a Composite Document       */
/* Presentation Data Stream (CDPDS) file and prints the document on */
/* a 4224 printer.                                                  */

 Arg fn ft fm .
/* Check invocation parameters                                      */
 If fn = '?' | fn = ''            /* If parameters are incorrect    */
    then signal prompt            /* prompt user .....              */
/* Substitute default value for filetype & filemode if non-specified */
 Parse Value ft "LISTCDP"  With ft . /* I/P file type               */
 Parse Value fm "*"        With fm . /* I/P file mode               */
/* Set default parameters for ADM4CDUV                              */
 copies   = "1"                   /* number of copies               */
 duplex   = "1"                   /* 1 = simplex                    */
                                  /* 2 = normal duplex              */
                                  /* 3 = tumble duplex              */
 procopts = ""                    /* Processing options             */
 devtok   = "X4224QE"             /* Device token                   */
 family   = "1"                   /* GDDM Family                    */
 namelist = "061"                 /* Namelist entry (print address) */

/* Check that the specified CDPDS file exists                       */
 address command 'STATE' fn ft fm  /* Look for specified file       */
 If rc ¬= 0 then                  /* If not, issue error message    */
    do;                           /* and exit with CMS return code  */
      say                         /*                                */
      say fn ft fm 'NOT FOUND'    /*                                */
      say                         /*                                */
      exit rc                     /*                                */
    end;                          /*                                */

/* Start the GDDM Composite Document Print Utility                  */
 address command 'ADM4CDUV ' fn ft fm '(' copies duplex ')',
         '(' namelist ')(' procopts ')(' devtok ')(' family ')'
 exit rc

/* Provide a description of the invocation parameters for this exec */
prompt : parse source . . execname .
say 'This exec reads a Composite Document Presentation Data Stream (CDPDS)'
say 'file and prints the composite document on a 4224 printer.          '
say '                                                                   '
say '   Format :-                                                       '
say '                                                                   '
say '          'execname' filename filetype filemode                    '
say '                                                                   '
say '          where filename is the input filename                     '
say '                filetype is the input filetype                     '
say '                filemode is the input filemode                     '
say '                                                                   '
exit 0
```

Figure 1. REXX procedure for printing composite document on 4224 under CMS

```
/* Name : CD38SAMP - sample exec  (38xx AFPDS printer)            */
/* This is a sample user exec that takes a Composite Document     */
/* Presentation Data Stream (CDPDS) file and creates a LIST38PP   */
/* file for printing by a 38xx AFPDS printer.                     */

 Arg fn ft fm .

/* Check invocation parameters                                    */
 If fn = '?' | fn = ''            /* If parameters are incorrect  */
    then signal prompt            /* prompt user .....            */
/* Substitute default value for filetype & filemode if non-specified */
 Parse Value ft "LISTCDP"  With ft . /* I/P file type             */
 Parse Value fm "*"        With fm . /* I/P file mode             */
/* Set default parameters for ADM4CDUV                            */
 copies   = "1"                   /* number of copies             */
 duplex   = "1"                   /* 1 = simplex                  */
                                  /* 2 = normal duplex            */
                                  /* 3 = tumble duplex            */
 procopts = "9 1 7 20"            /* GDDM processing options      */
                                  /*  9 = 1 Formatted output      */
                                  /*  7 = 20 Swathes              */
                                  /* 32 = 0 no inline resources   */
 devtok   = "IMG240"              /* Device token                 */
 family   = "4"                   /* GDDM Family                  */
 postproc = "PRT3812"             /* Post processing              */
                                  /* PRT3812 - print on 3812      */
                                  /* PSF     - print on 3800-3    */
 outfn    = fn                    /* O/P file name                */
 outft    = "LIST38PP"            /* O/P file type                */
 namelist = outfn outft           /* output file name             */

/* Check that the specified CDPDS file exists                     */
 address command 'STATE' fn ft fm /* Look for specified file      */
 If rc ¬= 0 then                  /* If not, issue error message  */
    do;                           /* and exit with CMS return code */
      say                         /*                              */
      say fn ft fm 'NOT FOUND'    /*                              */
      say                         /*                              */
      exit rc                     /*                              */
    end;                          /*                              */

/* Erase the output file if it already exists                     */
 address command 'STATE' outfn outft 'A'
 If rc = 0 then
   address command 'ERASE' outfn outft 'A'
```

Figure 2 (Part 1 of 2). REXX procedure for printing composite document on 38xx AFPDS printer under CMS

```
/* Start the GDDM Composite Document Print Utility              */
 address command 'ADM4CDUV ' fn ft fm '(' copies duplex ')',
           '(' namelist ')(' procopts ')(' devtok ')(' family ')'

/* Post processing                                              */
 Select
                                   /*Invoke PRT3812 to print the file*/
   When postproc = 'PRT3812' then
   'PRT3812 ' namelist ' ( COPIES ' copies

                                   /* Invoke PSF to print the file   */
   When postproc = 'PSF'     then
     Do
       'SPOOL PRINTER CLASS B FORM PAGEQUAR NOHOLD'
       'PSF'  namelist ' ( COPY ' copies

/* If procopt 32 = 1 use the following line instead             */
/*     'PSF'  namelist ' ( COPY ' copies 'FORMDEF (F1ADM001))'      */
     End

   Otherwise;
 End

 exit rc

/* Provide a description of the invocation parameters for this exec  */
prompt : parse source . . execname .
say 'This exec reads a Composite Document Presentation Data Stream (CDPDS)'
say 'file and creates a LIST38PP file for printing on a 38xx printer.     '
say '                                                                     '
say '   Format :-                                                         '
say '                                                                     '
say '          'execname' filename filetype filemode                      '
say '                                                                     '
say '          where filename is the input filename                       '
say '                filetype is the input filetype                       '
say '                filemode is the input filemode                       '
say '                                                                     '

exit 0
```

Figure 2 (Part 2 of 2). REXX procedure for printing composite document on 38xx AFPDS printer under CMS

Printers for composite documents

The IBM printers that support composite documents are:

3800 model 3 and model 8
3812
3812 Model 2 with 3270 Attachment Feature
3820
4224.

GDDM uses two data streams to support them. The Advanced Function Presentation Data Stream (AFPDS) supports all except the 3812 Model 2 with a 3270 Attachment Feature and the 4224; these are supported through the Intelligent Printer Data Stream (IPDS).

IPDS enables you to use text, image, and graphics to produce composite documents that can be printed in color on the 4224 printer. However, rotation of composite document pages, rotation of lines of text, and rotated fonts are not supported.

Use of 4224 memory

GDDM Version 2.2 stores text and graphics in the memory of the 4224 printer before the page is printed. Because the text is stored first, there is less memory available to store graphic drawing orders. So, a graphics object that printed correctly on the 4224 printer using GDDM Version 2.1 may not print correctly as part of a composite document using GDDM Version 2.2.

Fonts, page sizes, and characters

GDDM does not check the suitability of the code pages or the fonts for the target printer. The application program that generates the CDPDS or AFPDS must ensure that the correct ones are selected. The application program must also ensure that the text is formatted to fit within the page.

GDDM does not issue messages for undefined characters. Checks for such characters are handled by the printer for IPDS output and by the print server for AFPDS.

Color masters from CDPDS documents

Only one color master is allowed from a CDPDS document. In other words, the only valid value for the MASTERS parameter of the ADMMCOLT macro is 1. The IBM-supplied color table ADM00006 uses this value, and is suitable for translating colors in CDPDS documents into a gray scale.

Inline resources for AFPDS printers

To print a CDPDS document correctly on an AFPDS printer, you may need to use a new processing option when the printer is opened by GDDM. This is because the CDPDS document may have information at the front to control how the rest should be printed. The information is called **inline resources**. Inline resources contain information such as page offsets, overlay names, simplex/duplex control, and paper source. Processing option 32 indicates whether or not the CDPU is to transfer inline resources from the CDPDS input to the AFPDS output.

Not all AFPDS printer drivers support inline resources but if the version installed in your environment does support them, they can be generated using processing option 32.

A suitable nickname is:

```
NICKNAME FAM=4,PROCOPT=((INRESRCE,YES))
```

The CDPU supports inline resources in CDPDS input and AFPDS output only. Inline resources contained in AFPDS input files are ignored.

GDDM error reporting

It is possible that users of programs that create and print composite documents will see GDDM messages. These messages are prefaced by the letters ADM. GDDM messages are explained in the *GDDM Messages* manual.

Generally, errors that do not cause the printer to stop are collected into an error report. The error report is printed on a separate page at the end of the document. Message ADM2779 is displayed on the screen if there is such an error report.

Errors in data streams

The following message can be generated with a reason code of 4 if the printer cannot process the color separation required, or a reason code of 5 if the printer cannot process image and graphics on the same page:

```
ADM3179 W IMAGE CANNOT BE SHOWN, REASON CODE n
```

When you are using IPDS, if the printer detects an error, printing stops and GDDM issues an error message.

Errors in user environment or invocation of CDPU

Errors can be generated by invalid data. Messages returned by application programming interface calls are also reported.

Errors in the programming interfaces

One overall message, ADM2779, is returned to the caller of the SPI or API if an error report has been produced.

The GDDM font emulation and conversion tables

These two tables contain information about fonts and code pages that the CDPU uses to emulate CDPDS and AFPDS documents on screens (the font emulation table) and to print them on IPDS printers (the AFPDS to IPDS font and code page conversion table). One table of each type is supplied with GDDM. They are suitable for most applications. You do not need to change them unless you have special requirements.

Each entry in the font emulation specifies:

- Name of CDPDS or AFPDS font, coded font, or code page to be emulated
- Name of GDDM symbol set to be used for the emulation
- Symbol set character width
- Symbol set character height
- Symbol set character shear
- Symbol set color
- Symbol set code page
- Code page identifier for coded font or code page.

The AFPDS to IPDS conversion table allows the code page, or the font, or both to be converted. Each entry specifies:

- What is to be converted: name of AFPDS font, code page, or coded font (font/code page combination)
- Type of IPDS printer to which the entry applies
- Identifier of IPDS font or code page, or both, to be used
- IPDS font width: normal or wide
- IPDS font weight: normal or bold
- IPDS font descriptor: normal or italic, or double-struck, or both.

Each table is in a module. The one for the font emulation table is called ADM4FONT, and that for the AFPDS to IPDS conversion table is called ADMDKFNT. They are link-edited with GDDM. A copy of each module is supplied with GDDM for installations that want to add new fonts, or change the fonts or code pages used by GDDM. Entries in ADM4FONT are generated using the ADMMFONT macro, and in ADMDKFNT using the ADMMKFNT macro. Instructions are given in the *GDDM Installation and System Management* manuals.

The comments in the IBM-supplied ADMDKFNT table contain some guidance on how to obtain a best approximation to AFPDS output on an IPDS printer. In principle, it is necessary to use only AFPDS fonts, code pages, and coded fonts that can be converted into exact IPDS equivalents. Where this is not possible, some characters may sometimes print incorrectly.

Users who installed GDDM Version 2 Release 1 Modification 1 and modified the font emulation table should inspect the ADM4FONT macros shipped with Version 2 Release 2. They contain definitions for the emulation of 4250 fonts, which these users may want to add to their own versions of the table.

AFPDS structured fields supported by the CDPU

The Composite Document Print Utility (CDPU) permits printing and viewing of a document, a page segment or an overlay in Advanced Function Presentation Data Stream (AFPDS) format. The AFPDS file can be formatted for a 38xx or 4250 device (in, for example, LIST38PP, LIST4250, LISTAPA, PSEG38PP, PSEG4250, OVLY38PP, or OVLY4250 format).

The AFPDS cannot contain multiple documents or page segments. The structured field length must not exceed 8202 bytes. Input formatted for the 4250 can only be viewed. A document may contain page segments inline. Secondary input is not supported. Page segments that contain text cannot be printed unless they are imbedded in a document.

The formats of individual structured fields, such as "begin-document", are defined in the *PSF Data Stream Reference for MVS and VSE* and *CDPF Data Stream Interface Typographic Fonts Interface*.

Summary of AFPDS structured fields supported by the CDPU

Hex code	Meaning
D3A67B	Image input descriptor (IID)
D3A69B	Composed text descriptor (CTD)
D3A6AF	Page descriptor (PGD)
D3A77B	Image output control (IOC)
D3A79B	Composed text control (CTC)
D3A85F	Begin page segment (BPS)
D3A87B	Begin image block (BIM)
D3A89B	Begin composed text block (BCT)
D3A8A8	Begin document (BDT)
D3A8AF	Begin page (BPG)
D3A8C9	Begin active environment group (BAG)
D3A8DF	Begin medium overlay (BMO)
D3A95F	End page segment (EPS)
D3A97B	End image block (EIM)
D3A99B	End composed text block (ECT)
D3A9A8	End document (EDT)
D3A9AF	End page (EPG)
D3A9C9	End active environment group (EAG)
D3A9DF	End medium overlay (EMO)
D3AC7B	Image cell position (ICP)
D3B18A	Map coded font (MCF)
D3EE7B	Image raster data (IRD)
D3EE9B	Composed text data (CTX)

The include page segment structured field is not supported. If found, it will be treated as an error.

Structured field introducer extensions (bit 0, flag byte 5 of SFI) are not supported. If found, they will be treated as errors. Padding bytes (bit 4, flag byte 5 of SFI) are not supported.

Chapter 8. Symbol sets

This chapter describes the ways in which GDDM processes its various symbol set operations for the different device types.

The chapter also contains descriptions of the GDDM sample Image and Vector symbol sets that are supplied with GDDM. The sample symbol sets can be used by application programs instead of the defaults provided with GDDM.

How GDDM handles symbol sets

GDDM provides facilities for loading and using symbol sets other than the default characters, markers, and shading patterns. These may be image symbol sets (ISS), or vector symbol sets (VSS). Two methods of loading symbol sets are available:

- Loading image symbol sets directly into programmed symbol (PS) stores in the device
- Loading image symbol sets or vector symbol sets into GDDM storage.

PS stores are used for alphanumerics and mode-1 graphics text, GDDM storage for mode-2 and mode-3 graphics text.

For these operations, the symbol sets can be loaded from auxiliary storage, or passed as data from the application program.

In addition to being loaded by these operations, symbol sets can be passed as data between the application program and auxiliary storage.

| Symbol sets can be tagged with country-extended code page (CECP) identifiers. CECP sets are automatically converted when they are used. So a set tagged with code page identifier 00037 (for the United States) is converted to represent code page 00297 when it is loaded into a French device.

Where possible, GDDM loads symbol sets into device storage. For example, for 3270-PC/G and 3270-PC/GX work stations, both image symbol sets and vector symbol sets can be loaded into the device, whereas for | 3179-G and 3192-G, only image sets can be loaded.

Note: No symbol sets can be loaded into the 5550-family work stations.

Loading programmed symbol stores

Display devices and printers equipped with the programmed symbol (PS) feature contain PS stores that can be loaded with symbol definitions. These PS stores are used for:

- Storing additional or special symbol sets
- Storing symbols or cell definitions used in constructing a picture.

Symbol sets that are to be loaded into PS stores must have the same matrix dimensions as the device character cell. These are:

PS store numbers

The PS store number may optionally be specified as a parameter of the loading call (PSLSS or PSDSS). If specified, it must exist on the device in use at the time, and it must be a triple-plane store if a multicolor symbol set is to be loaded. Call statements are available to determine the number and types of PS stores in the device.

The specified store number controls the function key that can be used by the terminal operator to select the symbol set for data entry. The correspondence between store numbers and keys is:

Table 5. PS store number and PS key relationship

PS store number	PS key and indicator
2	A
3	B
4	C
5	D
6	E
7	F

A store number should always be specified if data entry using the symbol set is expected. The number need not be specified if data entry is not allowed by the application program.

When no store number is specified, the symbol set is loaded into an appropriate PS store, if one is available. Monochrome symbol sets may be loaded into either single-plane or triple-plane stores (for example: numbers 2, 3, 4, 5, 6, and 7 on a 3279 display), but multicolor symbol sets require triple-plane stores (for example: numbers 4, 5, and 7 on a 3279 display).

Symbol-set identification

Displays and printers identify loaded symbol sets by a one-byte symbol-set identifier. Usually, symbol sets are held on auxiliary storage. When a set is loaded into the PS store, a symbol-set identifier specified as a parameter in the loading call is associated with the data. It is then used to identify the symbol set during execution of the application program.

Reference to the symbol-set identifier takes one of two forms:

- A single character

  This form must have a character code greater than X'40', and it is used when identifying the symbol set associated with individual characters, as in the ASCSS call. If this function or the related query function ASQSS is used, it is likely that the symbol-set identifier chosen will be an alphanumeric character.

Table 6. Device cell-size dimensions			
Device	**Models**	**Character cell**	
		Width	**Height**
3179-G and 3192-G color display	All	9	12 (see note below)
3268 printer	2C	10	8
PCLK adapter with CGA card	All	8	8
PCLK adapter with EGA card (64 K)	All	8	8
PCLK adapter with EGA card (128+ K)	24-row screen	8	14
PCLK adapter with EGA card (128+ K)	32-row screen	8	11
PCLK adapter with MCGA card	24-row screen	8	19
PCLK adapter with MCGA card	32-row screen	8	14
PCLK adapter with VGA card	24-row screen	8	19
PCLK adapter with VGA	32-row screen	8	14
8514/A + 8503, 8512 or 8513	24-row screen	8	19
8514/A + 8503, 8512 or 8513	32-row screen	8	14
8514/A + 8514	24-row screen	12	30
8514/A + 8514	32-row screen	12	23
8514/A + 8514	43-row screen	12	17
8514/A + 8514	27-row by 132 col screen	7	24
3270-PC display	All	9	14
3270-PC/G work station	All	9	10 or 16 (selectable)
3270-PC/GX work station	All	12	20
3278 display	2, 3 4	9 9	16 12
3279 display	2B, 3B	9	12
3287 printer	All	10	8
3290 information panel display	All	9	16
4224 printer	1E2, 1C2	20	18
8775 display	1, 11 1, 12	9 9	16 12 or 16 (selectable)

Note: The alphanumeric cell-size on a 3179-G or 3192-G can be either 9 by 12, or 9 by 16. The actual cell-size is governed by the depth of the GDDM page and subsystem-related factors. Usually, any page with 24 rows or less causes a cell-size of 9 by 16, with other page sizes receiving a cell-size of 9 by 12. The substitution character for a 3179-G or 3192-G is independent of the page size, and always corresponds to the 9 by 12 cell.

- A full-word integer

  This form is used when specifying the symbol-set identifier to be associated with given data, an alphanumeric field, or a graphics character string.

The correspondence between the integer and the character specifications is:

- Characters "0" and "1" correspond to the integers 0 and 1. These refer to "read-only" character sets.
- Other characters correspond to their character codes. For example, "A" corresponds to 193.

Integer symbol-set identifiers in the range 224 through 239 are reserved for graphics use and cannot be assigned to loaded symbol sets.

Using preloaded PS sets

When GDDM is initialized, the current state of the PS stores is determined by a device query, which returns the identifier of any loaded sets. These preloaded sets are noted by the GDDM PS management routines, which maintain knowledge of the contents of the PS stores.

GDDM's PSLSSC call **conditionally** loads a symbol set into a PS store only if the PS store does not already contain a symbol set with the specified identifier. Conditional loading can be used to optimize PS loading, but it must be used with care, because incorrect results occur if different symbol sets have the same identifier. For example, an application program may load a symbol set with a given identifier, and another program running subsequently on the same device may attempt a conditional load of a different set having the same identifier. This situation can be avoided if a convention is adopted that assigns unique identifiers to specific symbol sets.

Selecting symbol sets by device type

If an application program is designed to be used with different devices, it may be necessary to control symbol set loading on the basis of cell size. This can be done by using a GDDM symbol-set naming convention. The symbol-set name is specified as a parameter of the loading call. If the last character of the name is the period character "•", GDDM replaces it by another character, depending on the current device.

In this way, a symbol set that matches the device in use can be retrieved from auxiliary storage and loaded. As a particular application, if a display containing PS is to be printed, this function allows the selection of a symbol set specific to the printer when printing begins.

For the details of which symbol sets are loaded for a particular device cell size, see Table 7 on page 69.

Using PS with graphics

| This section does not apply to 3179-G or 3192-G color display stations, 3270-PC/G and 3270-PC/GX work stations, 4224 printers, 5550-family work stations, the 5080 graphics system, and devices supported by GDDM-PCLK, because PS is not used to construct the graphics for these devices.

When GDDM is constructing a picture, the assumption is made that all PS stores in the device are available for use except those that have either been loaded with symbol sets, or explicitly reserved by the application program. Because the number of PS stores is limited, if an application program uses both additional PS character sets and graphics construction, special attention to PS allocations may be required. This is especially true for printers, because only one PS store can hold a multicolor symbol set.

In general, PS stores should be loaded with any additional symbol sets before graphics picture construction is started, because the PS stores are also used for picture display. An attempt to load a symbol set when graphics are displayed is usually rejected by GDDM. Only when all graphics items are deleted from all pages do the PS stores become released for loading symbol sets.

If the programmer anticipates the need to load a PS store while graphics data is present, the PSRSV call is available to **reserve** a PS store. This must be done before any graphics calls are issued. The specified PS store is not used for graphics data, and is explicitly referred to in the call statement to load the symbol set. When the symbol set is no longer needed, the symbol set can be released from the reserved PS store, and another symbol set can be loaded, or, the PS store itself can be released.

In a windowing environment, the PS stores are allocated in the following order:

1. For symbol sets in the active window
2. For graphics in the active window
3. For graphics for window borders (all windows)
4. Any remaining PS slots are allocated for symbol sets and graphics in non-active windows.

Loading graphics symbol sets

Symbol sets that are not suitable for loading into PS stores can be loaded into GDDM storage. (For 3270-PC/G and 3270-PC/GX work stations, these symbol sets can also be loaded into the device; for the | 3179-G and 3192-G, image symbol sets can be loaded into the device.)

Four types of symbol sets can be loaded in this way:

- Image symbol sets used as graphics text
- Image symbol sets or vector symbol sets used as marker symbols
- Image symbol sets used for shading graphics areas
- Vector symbol sets used for graphics text.

Unlike when loading into PS stores, there is no restriction on symbol size when loading image symbol sets into GDDM storage. Any size that can be created with the Image Symbol Editor can be used. However, when shading patterns are used, the symbol is truncated or padded to the cell size and repeated **at cell intervals**. Therefore, in most circumstances shading patterns should be the same size as the cell.

| Devices other than work stations, 3179-Gs, 3192-Gs, 4224s, and GDDM-PCLK devices

For these devices, in graphics there is occasionally a choice between loading a symbol set into a PS store for use in mode 1, and loading it into GDDM storage and using mode 2. Mode 2 is required if the character set does not match the device, or if exact positioning is required. If neither of these conditions exists, it should be remembered that the PS load transmits all characters in the symbol set to the device once only. Using the characters in mode 2 requires the transmission of only those characters actually used, but more than one cell definition may be transmitted for each.

For details of how to set the mode, see the *GDDM Base Programming Reference, Volume 1.*

Also, for guidance information on mode-1 and mode-2 usage for graphics, see the *GDDM Application Programming Guide, Volume 1.*

3270-PC/Gs, 3270-PC/GXs, 3179-Gs, and 3192-Gs

| **Note:** This section also applies to 3179-G and 3192-G color display stations, except that they cannot be loaded with vector symbol sets.

Image and vector character sets can be stored in the work stations themselves. Also, these displays support a maximum of two monoplane PS stores; the precise number depends on how the display has been configured. Programmed symbol sets are not used to construct graphics because the displays have their own graphics capability. Up to 8 character sets can exist in the display at any one time. For reasons of performance, the device-provided default character sets should be used whenever possible. Only PS sets can be used for alphanumeric characters.

| **Note:** The 3179-G or 3192-G display stations, and 3270-PC/G or 3270-PC/GX work stations have a different pixel aspect ratio and default graphics character-box size from displays such as the 3279. Thus, character mode 1 graphics character strings and character mode 2 text and images appear differently on the two types of device.

To prevent storage problems in the display, any symbol sets that have been loaded (by using GSDSS or GSLSS calls) should be released when they are no longer needed. The storage occupied by these symbol sets is common to that used for storing segments, so loading unnecessary symbol sets can cause segment storage to be exhausted (thereby causing GDDM to enter unretained mode with a subsequent effect on performance).

Note also that unless the work station has enough symbol-set storage to hold the current user-defined pattern sets, the default shading patterns are used (GDDM issues a warning message when this happens).

For details of how to set the mode, see the *GDDM Base Programming Reference, Volume 1*.

Also, for guidance information on mode-1 and mode-2 usage for graphics, see the *GDDM Application Programming Guide, Volume 1*.

PS overflow caused by picture complexity

| PS overflow cannot occur on 3179-G and 3192-G display stations, 3270-PC/G and 3270-PC/GX work stations, 5550-family work stations, the 5080 Graphics System,
| or on devices supported by GDDM-PCLK; therefore, ignore this section for these devices.

When a picture is extremely complex, it may require more PS stores than GDDM and the device can handle. This is known as PS overflow. When PS overflow occurs, message ADM0273 is issued to inform the user that the picture cannot be accurately completed.

In a windowing environment, this message is only issued if the overflow occurs in the active window.

The 4224 printer performs its own vector-to-raster conversion for graphics data. The graphics data stream that is sent to these printers contains GDF orders. The amount of storage available in these printers may not be enough to hold all of the graphics data that defines the picture. When this occurs, message ADM3282 is issued to inform the user that the picture cannot be accurately completed.

The FSCHEK function can be used to discover if PS overflow will occur when a picture is displayed. If PS overflow would occur, the error can be intercepted and action taken to simplify the picture or delete segments until it can be shown.

Using symbol sets in printing

When a call is issued to copy screen data to the printer, the names of symbol sets in use, both on the screen and in GDDM storage, are noted. These names include the final character "•" if it was originally specified, not the character that was substituted for it.

When the print operation begins, an attempt is made to reload the symbol sets. The appropriate substitution character replaces the "•", so that a printer symbol set is retrieved, if one exists on auxiliary storage. If not, the default symbol set is used and an error message is issued.

Note that if the symbol set was loaded into the display by a conditional PS load, a conditional load is also performed before printing. Therefore, the convention associating symbol sets with unique identifiers must apply for both displays and printers.

Because there may be more PS stores available on a display than on a printer, if an application program explicitly uses PS stores, a picture that can be displayed may not print. Also, because only one triple-plane store is available in the 3287 Printer (Models 1C and 2C), if the application reserves this store for a non-graphics symbol set when the print request is processed, multicolor graphics printing is not performed correctly.

Using DBCS symbol sets

For Kanji/Hangeul applications that have double-byte character string (DBCS) symbol sets installed, this type of symbol set can be used directly (by the application program loading the required symbol set and using the definition in the normal manner) or indirectly (by the application program indicating that it requires to use DBCS symbol sets). In the second case, if the GSCS call specifies character set 8 (DBCS) or if mixed (single-byte and double-byte) character strings are enabled (by specifying MIXSOSI=YES in GDDM's external defaults), GDDM recognizes DBCS characters and uses the first byte of the character to identify the symbol set to be loaded and the second byte to retrieve the symbol definition.

GDDM's external defaults define whether mixed strings are enabled and indicate the maximum number of DBCS symbol sets of each type that are to be loaded concurrently. When this maximum number is reached, the least recently used symbol set is unloaded to allow the currently required symbol set to be loaded. For details on how to change the settings of these GDDM external defaults, see the information on the MIXSOSI and DBCSLIM processing options in Chapter 1, "Customizing your program and its environment" on page 1.

For graphics, DBCS symbol sets are available for mode 2 and mode 3 only.

Naming conventions for sample image symbol sets

Except for shading patterns with a final character of N or R (and ADMDHIPK), the final character of the name of each image symbol set conforms to the convention for generic retrieval by GDDM, showing the cell size of the symbol set.

The shading patterns with N and R as the final character differ in that the patterns are defined on an 8 by 12 cell size. This allows complete shading, as defined in the GSLSS call; see the *GDDM Base Programming Reference, Volume 1*.

The following table shows the character that GDDM uses to replace the "•" substitution character that is used in a GSLSS call (for a graphics symbol-set name), in a PSLSS or PSLSSC call (for an alphanumerics symbol-set name), or in an SSREAD or SSQF call (for either alphanumerics or graphics).

Table 7. Cell sizes for sample image symbol sets

Substituted final character	Cell size in display points (width by depth)
A	9 by 16
C	9 by 12 (monochrome)
D	9 by 12 (multicolor)
E	9 by 10 (alphanumerics only)
G	10 by 8 (monochrome)
H	10 by 8 (multicolor)
J	Family-4 high-resolution symbol sets (400 pixels per inch or greater)
K	20 by 18 (alphanumerics only)
L	Family-4 medium-resolution (less than 400 pixels per inch)
M	See Note 4
N	8 by 16 (graphics only)
Q	24 by 30 (monochrome)
R	12 by 20
	12 by 24
U	Plotter symbol sets

Notes:

1. If the device has a cell size that is not one listed above, GDDM selects the character that corresponds to the smallest containing cell size. For example, for a device cell size of 9 by 14, GDDM selects an image symbol set with a cell size of 9 by 16 (character A).
2. If the device cell size does not fit into any of the cell sizes given in the table, GDDM selects an image symbol set with a cell size of 9 by 16 (character A).
3. For a family-3 printer, the character A is always used as the final character.
4. GDDM provides a sample image symbol set with M as the last character. However, M is not one of the characters in the substitution rules.

Sample image symbol sets

Table 8. Sample image symbol sets

Set name	Contents
ADMCOLSD ADMCOLSN ADMCOLSR	Sample shading patterns, which create the appearance of 64 color shades.
ADMDHIIA ADMDHIIC ADMDHIIE ADMDHIIG ADMDHIIK ADMDHIIN ADMDHIIQ ADMDHIIR	The standard CECP set of characters.
ADMDHIMA ADMDHIMC ADMDHIMG ADMDHIMK ADMDHIMN ADMDHIMQ ADMDHIMR	Ten standard markers, which correspond to the defaults provided with GDDM. See the description of the GSMS call in the *GDDM Base Programming Reference, Volume 1.*
ADMDHIPA ADMDHIPC ADMDHIPG ADMDHIPJ ADMDHIPM ADMDHIPN ADMDHIPR	Seventeen standard patterns, which correspond to the defaults provided with GDDM. See the description of the GSPAT call in the *GDDM Base Programming Reference, Volume 1*. ADMDHIPJ is for use on an IBM 4250 high-resolution printer, and ADMDHIPM is for use on IBM 3800-3 and 3800-8 medium-resolution printers.
ADMDHIPK	Eight patterns, which can be used when producing color masters on high-resolution and medium-resolution printers. For more information, see "The ADMMCOLT macro" on page 79.
ADMDHIPL	Sample shading patterns, which can be used for converting colors into shades of gray on high and medium-resolution printers.
ADMIPATA ADMIPATC ADMIPATG ADMIPATN ADMIPATR	Seventeen standard patterns, which correspond to the defaults provided with GDDM. Used with the Image symbol Editor INFILL function. See the description of the GSPAT call in the *GDDM Base Programming Reference, Volume 1.*
ADMITALA ADMITALC ADMITALG ADMITALK ADMITALN	Sample italic CECP characters.
ADMPATTA ADMPATTC ADMPATTG ADMPATTN ADMPATTR	Sixty-four sample geometric shading patterns. See the description of the GSPAT call in the *GDDM Base Programming Reference, Volume 1.*
ADMIKxx	Sample double-byte character set image characters, where "xx" is in the range X'41' through X'68'.
ADMDISKA ADMDISKC ADMDISKG	Contain image symbols for use with Katakana displays and printers.
ADMDISKN ADMDISKP ADMDISKR	Contain 8x16 and 12x24 image symbols for use with 5550-family work stations.

Note: The symbol sets are only provided as samples. GDDM does not ensure that all styles of characters and patterns are provided for all possible suffix characters.

Sample vector symbol sets

GDDM's sample vector symbol sets are as shown below:

Table 9. Sample vector symbol sets

Set name	Contents
ADMDHIMJ	Contains the GDDM vector marker symbols for use by the Interactive Chart Utility.
ADMDHIMV	Contains ten standard vector markers that correspond to the defaults provided with GDDM.
ADMDHIVJ	Contains the default vector symbol set for the 4250 page printer.
ADMDHIVK	Contains the default vector symbol set for a 4224 page printer.
ADMDHIVM	Contains the default vector symbol set for a 3800 Model 3 or a 3800 Model 8 page printer.
ADMDHIVQ	Contains the default vector symbol set for a 3812 Model 2 page printer.
ADMDVIH	Contains the default vector symbol set for 3270-PC/G or 3270-PC/GX work stations.
ADMDVECP	CECP default vector symbol set
ADMDVSS	The default vector symbol set for code page 00351 (USA version). The default character codes are shown in the description of the ASTYPE call in the *GDDM Base Programming Reference, Volume 1*.
ADMDVSSB ADMDVSSD ADMDVSSE ADMDVSSF ADMDVSSG ADMDVSSI ADMDVSSN ADMDVSSS ADMDVSSV	National Language versions of the vector symbol sets for code page 00351, see note 5.
ADMDVSSK	The default vector symbol set for code page 00290. See the description of the ASTYPE call in the *GDDM Base Programming Reference, Volume 1*.
	Sample CECP vector symbol sets:
ADMU★ARP	Area Filled Roman Principal
ADMU★CIP	Complex Italic Principal
ADMU★CRP	Complex Roman Principal
ADMU★CSP	Complex Script Principal
ADMU★DRP	Duplex Roman Principal
ADMU★FSS	Filled Sans Serif
ADMU★GEP	Gothic English Principal
ADMU★GGP	Gothic German Principal
ADMU★GIP	Gothic Italian Principal
ADMU★KRF	Thick Round Filled
ADMU★KRO	Thick Round Outlined
ADMU★KSF	Thick Square Filled
ADMU★KSO	Thick Square Outlined
ADMU★MOD	Modern
ADMU★NSF	Thin Filled
ADMU★NSO	Thin Outline
ADMU★ORP	Outline Roman Principal
ADMU★SHD	Shadow
ADMU★SRP	Simplex Roman Principal
ADMU★TIP	Triplex Italic Principal
ADMU★TRP	Triplex Roman Principal
ADMU★TSS	Triplex Sans Serif
★ is U	– proportionally spaced
★ is V	– nonproportionally spaced
★ is W	– proportionally spaced – wider space for compatibility with GDDM Version 1.
	See *GDDM Typefaces and Shading Patterns* manual.
ADMVKxx	Sample double-byte character set vector characters, where "xx" is in the range X'41' through X'68'.

Notes:

1. It is not possible to use the Image Symbol Editor on the sample vector symbol sets. The Vector Symbol Editor is part of GDDM-PGF.
2. As supplied, CECP symbol sets are ordered according to the USA CECP, 00037, and are so tagged. GDDM converts them to the device code page when they are loaded by an application program.
3. All the IBM-supplied sample vector symbol sets have names starting with "ADM"; this aids identification and serviceability. However, installations may find it more convenient to generate copies of these symbol sets, using other names. If necessary, the Image or Vector Symbol Editor can be used to save the symbol sets under different names. The symbol sets are shown in *GDDM Typefaces and Shading Patterns* manual.
4. It should not normally be necessary to alter a CECP set. However, if an editor is used to change a CECP symbol set, the application code page should first be set to be the same as that of the symbol set being edited. GDDM supplies the CECP sets ordered according to code page 00037.
5. The "E" suffix character refers to UK-English, **not** US-English.

Chapter 9. Picture interchange format files

Application programs can transfer picture information between GDDM running in a host system and the 3270-PC/G or 3270-PC/GX work station as **picture interchange format (PIF) files** by using the GDDM-supplied GDF conversion utility (ADMUPCT/V) and the 3270-PC Graphics Control Program file transfer function.

A PIF file can also be generated on a work station that uses GDDM-PCLK, through "user control mode" (for details, refer to the *GDDM-PCLK Guide*).

As the PIF files on the host have different internal formats to those on a work station, when files are transferred from one system to the other, they must also be converted to the relevant format before they can be used.

This conversion can be done at the same time as the transfer operation or as a separate operation.

The methods used to process PIF files vary according to the subsystem that the GDDM host session is running under. This chapter explains:

- Processing PIF files under TSO
- Processing PIF files under VM/CMS.

Note: GDDM does not support PIF files under CICS/VS or IMS/VS.

These topics are discussed for each subsystem:

- How PIF data relates to GDF data
- How to create PIF information under GDDM
- How to create PIF information at a work station
- What a PIF file must contain if it is to be used under GDDM
- The structure of a PIF file
- Base PIF files.

The commands needed to convert and transfer PIF files are defined in the sections that follow; for more information, refer to the *GDDM Guide for Users*.

Processing PIF files under TSO

The conversion operation

The GDF file-conversion utility

The conversion utility is distributed as a module called ADMUPCT. This utility converts GDDM ADMGDF objects into PIF files, or converts PIF files from the work station into a format that is suitable for use under GDDM Release 4 (ADMGDF objects).

The conversion utility also converts files (created by applications from GSGET calls and often named GDDM Release 2 and 3 GDF files) into ADMGDF files; see "Saving GDF orders" on page 165.

Figure 3 on page 72 shows the flow of events.

When the IND$FILE CLIST executes, the ADMUPCT command is invoked to run the conversion utility if the ADMGDF option has been specified in a SEND or RECEIVE command.

The transfer operation

If the commands described in "Commands to use under TSO" on page 72 did not work, check that the IND$FILE CLIST is available at your installation, and that the library search order searches CLISTs before searching commands. Refer to the preamble to CLIST ADMUPCFT (listed under ADMUPCFT in the index) in the *GDDM Installation and System Management for MVS* manual.

GDF data files must be converted into PIF files before they can be sent from GDDM to the work station. There are four components in the procedure for transferring and converting the files:

- The SEND and RECEIVE commands that are issued at the work station.

  These commands generate the IND$FILE command on the current host session, with the first parameter set to either PUT or GET.
- The IND$FILE CLIST that is issued at the host (GDDM).

  This CLIST controls the file transfer program and the conversion utility (see below).
- The IND$FILE file transfer command.
- The GDF conversion utility, which converts GDDM ADMGDF object files to PIF files, and conversely.

Of these four components, the SEND and RECEIVE commands have already been described above. The other components are described in greater detail below.

The IND$FILE CLIST

These examples of the commands work with the IND$FILE CLIST that is supplied with GDDM.

Note: The CLIST is distributed with the name ADMUPCFT CLIST; it is recommended that it is renamed to IND$FILE CLIST by the systems programmer, after GDDM has been installed.

The IND$FILE CLIST invokes the IND$FILE file transfer program at the work station.

Notes:

1. On heavily-loaded systems, it may be advisable to perform the file transfer separately from the conversion; for details, see "Commands to use under TSO" on page 72; for further information, refer to the *GDDM Guide for Users*.
2. For GDDM Version 2 Release 1, there is a new version of the CLIST called ADMUPGT, which maintains the structure of the PIF (including default tags and segment orders), but produces ADMGDF files that may not be compatible with some GDDM Version 1 Release 4 applications.

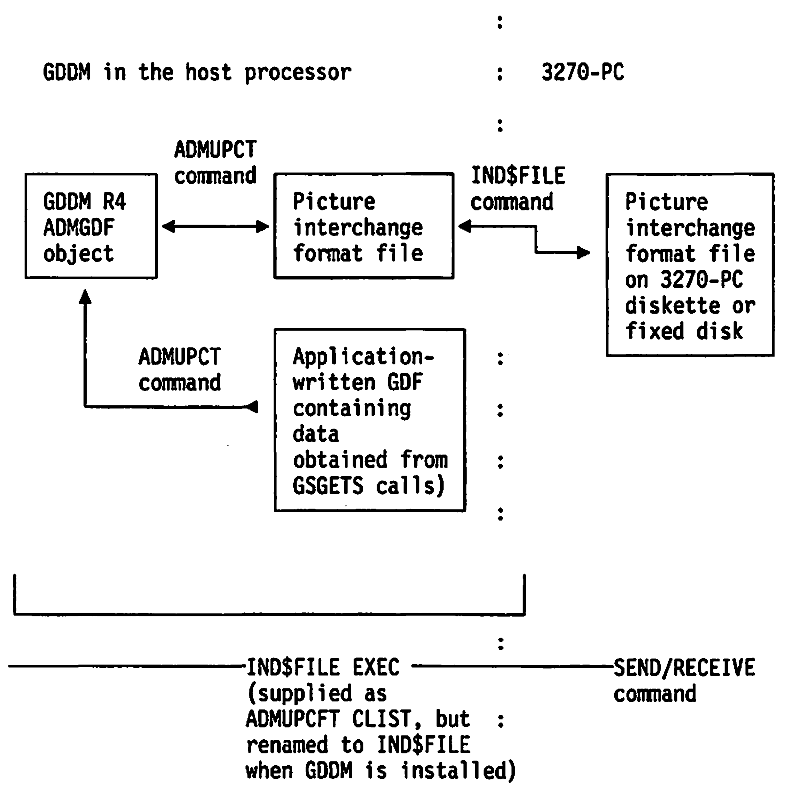

GDDM in the host processor
3270-PC
ADMUPCT command
GDDM R4 ADMGDF object
Picture interchange format file
IND$FILE command
Picture interchange format file on 3270-PC diskette or fixed disk
ADMUPCT command
Application-written GDF containing data obtained from GSGETS calls)
IND$FILE EXEC (supplied as ADMUPCFT CLIST, but renamed to IND$FILE when GDDM is installed)
SEND/RECEIVE command

Figure 3. GDF file conversion procedure under TSO

3. If the ADMUPCFT CLIST has been renamed to a name other than IND$FILE CLIST, the work station SET command can be used to invoke the appropriate CLIST when a SEND or RECEIVE command is issued. For details of the SET command, refer to the *IBM Personal Computer Disk Operating System* manual.

The IND$FILE file transfer command

This is the command that transfers files between a work station and the host processor.

Note: The file transfer command requires the 3270-PC Graphics Control Program (feature number 1507) and the File Transfer Program (licensed program number 5665-311), which runs on MVS/TSO.

Commands to use under TSO

To transfer a PIF file from the work station to host

1. Ensure that the host session is ready to receive an operator command (that is, it is in a READY state).
2. From the PC session of the work station enter:

   ```
   SEND picture.pif 'pif-dataset-name'
   ```

   The "pif-dataset-name" data set is automatically allocated if it does not already exist, and is created as a sequential data-set with fixed-length 80-byte records (unblocked). The "pif-dataset-name" if it already exists may be sequential or partitioned. If partitioned, the member-name must be included in "pif-dataset-name."

To transfer a GDDM GDF picture from the host to the work station

1. Enter the RECEIVE command from the work station (in a PC session) as follows:

   ```
   RECEIVE picture.pif 'pif-dataset-name'
   ```

   This sends the file "pif.dataset-name" from the host (GDDM) system to the current work-station directory, converting it from the ADMGDF format to a PIF format.

   If the SEND or RECEIVE command was not successful, there may be some options not set up on your system, and you should consider this:

To convert a PIF file into a GDDM ADMGDF object

1. Use the commands:

   ```
   ALLOC F(ADMPIF) DA('pif-dataset-name')SHR
   ALLOC F(ADMGDF) DA('admgdf-dataset-name')SHR
   CALL 'GDDM.OSPID.GDDMLOAD(ADMUPCT)'
        'pif-member (PUT admgdf-member options'
   ```

   Where "admgdf-dataset-name" must exist, and must be partitioned. The data set usually has the attributes LRECL(400) and RECFM(F) but these may be altered.

   If "pif-dataset-name" is sequential, pifmember should be omitted.

To convert a GDDM ADMGDF object into a PIF file

1. Use the commands:

```
ALLOC F(ADMPIF) DA('pif-dataset-name')SHR
ALLOC F(ADMGDF) DA('admgdf-dataset-name')SHR
CALL 'GDDM.OSPID,GDDMLOAD(ADMUPCT)'
     'pif-member(GET admgdf-member,options'
```

   Where "admgdf-dataset-name" must exist, and must be partitioned. The data set usually has the attributes LRECL(400) and RECFM(F) but these may be altered.

   If "pif-dataset-name" is sequential, pifmember should be omitted.

Notes:

1. The admpif-member-name is either a member name of the PIF data set or blank if a sequential data set is being used.
2. The ADMPIF data set defaults are LRECL=400 and RECFM=F, but these may be changed.
3. A user's CLIST must allocate two DDnames:
   - ADMPIF – for the PIF sequential or partitioned data set.
   - ADMGDF – for the partitioned data set with member "admgdf-name."
4. The GDDM-supplied IND$FILE CLIST accepts the SEND and RECEIVE commands from the work station, or it can run independently when invoked from GDDM in the host. See the *GDDM Installation and System Management for MVS* manual for a source listing of this CLIST.

The format of a PIF file

The format of a PIF file under GDDM in the host processor depends on the subsystem being used; under TSO, it can be a sequential data set or a member of a partitioned data set.

In a 3270-PC/G or 3270-PC/GX work station, and devices supported by GDDM-PCLK, the PIF file is a standard PC-DOS 2.1 file.

In both the host and the work station, the orders in a PIF file can span records.

Processing PIF files under VM/CMS

The conversion operation

The GDF file-conversion utility

The conversion utility is distributed as a module called ADMUPCV. This utility converts ADMGDF objects into PIF files, or converts PIF files from the work station into a format that is suitable for use under GDDM Release 4 (ADMGDF objects).

The conversion utility also converts files (created by applications from GSGET calls and often named GDDM Release 2 and 3 GDF files) into ADMGDF files; see "Saving GDF orders" on page 165.

Figure 4 on page 74 shows the flow of events.

When the IND$FILE EXEC executes, the ADMUPCV command is invoked to run the conversion utility if the ADMGDF option has been specified in a SEND or RECEIVE command.

The transfer operation

If the commands described in "Commands to use under VM/CMS" on page 74 did not work, check that the IND$FILE EXEC is available at your installation.

GDF data files must be converted into PIF files before they can be sent from GDDM to the work station. There are four components in the procedure for transferring and converting the files:

- The SEND and RECEIVE commands that are issued at the work station.

  These commands generate the IND$FILE command on the current host session, with the first parameter set to either PUT or GET.
- The IND$FILE EXEC that is issued at the host (GDDM).

  This EXEC controls the file transfer program and the conversion utility (see below).
- The IND$FILE file transfer command.
- The GDF conversion utility, which converts GDDM ADMGDF object files to PIF files, and conversely.

Of these four components, the SEND and RECEIVE commands have already been described above. The other components are described in greater detail below.

The IND$FILE EXEC

These examples of the commands work with the IND$FILE EXEC that is supplied with GDDM.

Note: The EXEC is distributed with the name ADMUPCFV EXEC; it is recommended that it is renamed to IND$FILE EXEC by the systems programmer, after GDDM has been installed.

The IND$FILE EXEC invokes the IND$FILE file transfer program at the work station.

Notes:

1. On heavily-loaded systems, it may be advisable to perform the file transfer separately from the conversion; for details, see "Commands to use under VM/CMS" on page 74; for more information, refer to the *GDDM Guide for Users*.
2. For GDDM Version 2 Release 1, there is a new version of the CLIST called ADMUPGT, which maintains the structure of the PIF (including default tags and segment orders), but produces ADMGDF files that may not be compatible with some GDDM Version 1 Release 4 applications.
3. If the ADMUPCFV EXEC has been renamed to a name other than IND$FILE EXEC, the work station SET command can be used to invoke the appropriate EXEC when a SEND or RECEIVE command is issued. For details of the SET command, refer to the *IBM Personal Computer Disk Operating System* manual.

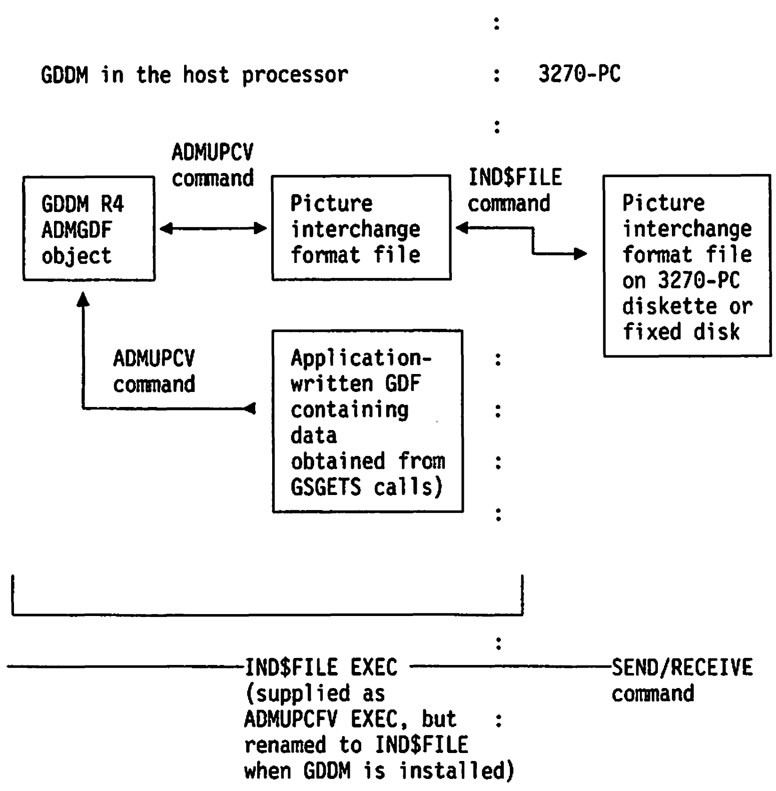

GDDM in the host processor
3270-PC
ADMUPCV command
GDDM R4 ADMGDF object
Picture interchange format file
IND$FILE command
Picture interchange format file on 3270-PC diskette or fixed disk
ADMUPCV command
Application-written GDF containing data obtained from GSGETS calls)
IND$FILE EXEC (supplied as ADMUPCFV EXEC, but renamed to IND$FILE when GDDM is installed)
SEND/RECEIVE command

Figure 4. GDF file conversion procedure under VM/CMS

The IND$FILE file transfer command

This is the command that transfers files between a work station and the host processor.

Note: The file transfer command requires the 3270-PC Graphics Control Program (feature number 1507) and the File Transfer Program (licensed program number 5664-281 for VM/SP) which runs on VM/SP Release 3.

Commands to use under VM/CMS

To transfer a PIF file from the work station to host

1. Ensure that the host session is ready to receive an operator command (for example, ensure that the host session is not running the Interactive Chart Utility).
2. Ensure that the CMS default SET IMPEX ON is in operation.
3. Enter the SEND command from the work station (in a PC session) as follows:

   ```
   SEND picture.PIF picture (ADMGDF
   ```

   This sends the file picture.PIF from the current work-station directory, converts it to GDDM format (because of the ADMGDF keyword), and stores the file as a GDDM ADMGDF picture in the host.

   If you want to transmit the file again unchanged (for back-up or transmission to another work station), do not use the keyword option ADMGDF as this option may result in some details of the picture being lost.

To transfer a GDDM GDF picture from the host to the work station

1. Ensure that the host session is ready to receive an operator command (for example, ensure that the host session is not running the Interactive Chart Utility).
2. Ensure that the CMS default SET IMPEX ON is in operation.
3. Enter the RECEIVE command from the work station (in a PC session) as follows:

   ```
   RECEIVE picture.PIF picture (ADMGDF
   ```

   This sends the GDDM ADMGDF picture file from the host ADMGDF object library to the current work-station directory.

   If you want to transmit the file again unchanged (for back-up or transmission to another work station), do not use the keyword option ADMGDF, as this option may result in some details of the picture being lost.

To convert a PIF file into a GDDM ADMGDF object

1. Use the command:

   ```
   ADMUPCV admpif-file-id (PUT admgdf-name options
   ```

   The options are:

   - *{NEWFile*|REPlace} – creates a new GDF object or replaces an existing object of the same name.
   - *{FIXed*|FLOAT} – creates the GDF object in fixed- or floating-point format.

To convert a GDDM ADMGDF object into a PIF file

1. Use the command:

   ```
   ADMUPCV admpif-file-id (GET admgdf-name options
   ```

   The options are:

   - {*NEWFile*|REPlace} – creates a new PIF file or replaces an existing file of the same name.
   - {*FIXed*|FLOAT} – creates the PIF file in fixed- or floating-point format. If the PIF file is to be sent to a work station, this parameter must be specified as FIXed.
   - LRECL {*400*|n} – specifies the length of each record for fixed-length files, or the maximum record length for variable-length files. The value of n must be in the range 16 through 2000.
   - RECFM {*F*|V} – specifies the record format as fixed length or variable length.

Note: The admpif-file-id is a standard CMS file identifier.

The format of a PIF file

The format of a PIF file under GDDM in the host processor depends on the subsystem being used; under VM/CMS, it is a normal VM/CMS file, conventionally of filetype PIF.

In a 3270-PC/G or 3270-PC/GX work station, and devices supported by GDDM-PCLK, the PIF file is a standard PC-DOS 2.1 file.

In both the host and the work station, the orders in a PIF file can span records.

Creating PIF data under GDDM

The graphics data in PIF files is essentially the same as that in fixed-point GDF files. Using GDDM's GSGETS call (see the *GDDM Base Programming Reference, Volume 1*), with the options for returning fixed-point coordinate data with a picture prolog, produces PIF orders.

Creating PIF data using GDDM-PCLK

PIF data can also be generated using GDDM-PCLK. For details, refer to the *GDDM-PCLK Guide*.

Creating PIF data at a work station

There are two ways of creating PIF data at a work station:

1. By capturing alphanumerics or alphanumerics and graphics data that is displayed on a monitor. This is done by:

   a. Pressing the Ws Ctrl key

   b. Pressing the Print or Print and Shift keys.

   This spools a file called INDPRTnn.PIF to the user's INDPRT directory for printing at the work station.

2. By writing an application program to create and save alphanumerics or graphics data, or both of these.

If they are to be transferred to GDDM, the PIF files created at a work station must contain only those drawing orders that are recognized as GDF orders; the GDF orders are listed and described in Appendix D, "GDF order descriptions" on page 165. The GDF utility converts orders where possible and diagnoses any changes made.

Note: Spooling a GDDM picture locally causes structural information to be lost because GDDM optimizes the data stream for display. Therefore, if possible you should create your PIF files at the host rather than spooling them locally into PC disk storage and retrieving them from the work station.

How PIF data relates to GDF data

The formats of data in PIF files and in files created by applications from the results of GSGET calls differ, in some respects, from those of Version 1 Release 4 GDF (ADMGDF) files created from GSSAVE calls. The conversion utility converts from one form to the other. The differences are:

- PIF files contain special control information as detailed below.
- Fixed-point GDF is, usually, a subset of PIF function. However, some GDF orders before Version 1 Release 4 are ignored by the work stations. The GDF utility makes the appropriate conversions. The orders are:

X'11'	Fractional Line Width
X'41'	Marker Scale
X'53'	Segment Position
X'71'	Segment End
X'72'	Segment Attribute
X'73'	Segment Attribute Modify.

  The work station treats all these orders as no operations.
- Fixed-point GDF End Area (X'6800') is treated as a Begin Area order by work stations. End Area should be shown using X'6000'.

For a full list of the drawing orders supported by the work station, see the *IBM 3270 Personal Computer/G or /GX: Reference Information for Picture Interchange Format* manual. See also the *IBM 3270 Personal Computer/G or /GX: Supplementary Reference Information for Picture Interchange Format* manual.

Pictures created at the work station for use under GDDM should contain only those GDF orders listed in Appendix D, "GDF order descriptions" on page 165 and should adhere to the restrictions that GDDM places on their use.

The conversion utility removes or changes orders in the PIF file that are not accepted by GDDM. In particular, note that symbol-set definitions are removed by the GDF conversion utility. For example, if a chart that uses symbol sets is created under GDDM's Interactive Chart Utility (ICU), and is stored using the Print Spool function, GDDM may use different symbol sets when the chart is sent to GDDM and displayed at the host. This is because PIF files created in this way do not reference the original symbol sets and because the symbol-set definitions in the PIF file are discarded.

Base PIF

For GDDM Version 2, there is a subset of GDF orders known as Base PIF. **All** Base PIF files can be imported into GDDM.

Restrictions and considerations

To ensure that ADMGDF files convert to Base PIF so that they can be exported, the following must be borne in mind:

Creating files

Avoid any GDDM calls involved with:

- Multiple-connected areas; for example a ring
- Image data
- Image symbols
- Loaded marker and pattern sets
- Foreground color mixing other than overpaint.

The spool print function

The same restrictions listed above must be observed when the Spool Print function is used to produce a PIF file from a picture originally created by a GDDM application.

The GDDM sample program ADMUSP4

PIF files imported into GDDM cannot be edited directly by the GDDM sample program ADMUSP4; see Appendix K, "Sample programs" on page 249.

Composed-page printing

There is no function provided, either in GDDM Base or GDDM-PGF, for sending ADMGDF files to a composed-page printer.

ADMUPCV and ADMUPCT utilities

When using these utilities to create PIF files, avoid generating files that have a floating-point format.

LCLMODE processing option

Ensure that the LCLMODE processing option is enabled. This ensures that the maximum amount of picture detail is present in a PIF file resulting from Spool Print. In the absence of local mode, GDDM optimizes the data stream (for example, an arc is expanded into a series of line segments), such that, at the original scale, a picture is displayed correctly. However, exporting the resulting PIF file to another | product such as DisplayGraphics, would not give the intended result.

GGXA file conversion

PIF files created by GGXA that contain pictures drawn with black lines will not be visible when imported into GDDM and viewed using a GDDM application, such as the ICU. They will, however, be plotted and printed successfully by GDDM.

| #### DisplayGraphics

| PIF files created by DisplayGraphics should be drawn
| white with black background. They, when imported into
| GDDM and viewed using a GDDM application, such as
| the ICU, display correctly as a white image on a black
| background, and print as black on white background.

The structure of a PIF file

A PIF file consists of the GDF orders that are listed and described in Appendix D, "GDF order descriptions" on page 165. Also, it can contain specific orders from the work station.

The structure of a PIF file created at a work station is as follows:

Table 10. The structure of a PIF file

File Descriptor order		
Begin Symbol Set Mapping order		
	Map Symbol Set Identifier order	One for each identifier
	. . .	
End Symbol Set Mapping order		
Begin Line Type Mapping order		
	Map Line Type Identifier order	
End Line Type Mapping order		
Begin Picture Prolog order		
	Set Picture Coordinates order	
	Set Picture Boundary order	
	Set Page Color order	
	"picture default" orders	
End Picture Prolog order		
Begin Segment order		Repeated for each segment of the picture
	"segment attribute" orders	
	End Segment Prolog order	
	"drawing" orders	
End Segment order		
Begin Symbol Set Definition order		See Note 2 below
	Load Symbol Set structured field	Repeated as needed for multiplane image symbol sets
	Continue Symbol Set Definition order	
	Load Symbol Set structured field	
	. . .	
End Symbol Set Definition order		
Begin Line Type Definition order		
	Load Line Type structured field	
End Line Type Definition order		

Notes:

1. Where present, the File Descriptor, Symbol Set Mapping, Line Type Mapping, Picture Prolog, Picture Segments, Symbol Set Definition, and Line Type Definition orders must be in the sequence shown.
2. The symbol-set definition orders are repeated for each internal symbol-set definition.
3. COMMENT and NOOP orders can be placed anywhere in the file except between the Begin Symbol Set Definition and End Symbol Set Definition orders, and between the Begin Line Type Definition and End Line Type Definition orders.
4. The GDDM-supplied conversion utility (ADMUPCT/V) removes these orders when the PIF file is converted to GDDM format:
   - The Line Type Mapping and Line Type Definition orders
   - The Symbol-Set Definition orders
   - The Set Page Color order.

The File Descriptor and Line Type Mapping orders, and the Set Page Color order, have no corresponding GDDM GDF orders. The format of these orders is described in the *IBM 3270 Personal Computer/G or /GX: Reference Information for Picture Interchange Format* manual.

Chapter 10. Setting up color-master tables

The GDDM page printer support provides the facility for creating a set of output files that represent the components in a subtractive (or additive) color-separation process.

In this mode, several output files are created for every picture. Each file represents one of the component colors and can be used to create the relevant printing plate.

To allow maximum flexibility, the separation process is determined by a table, ADMDJCOL, which can be constructed using a supplied macro, ADMMCOLT, and an image pattern set, which can be constructed using the Image Symbol Editor, ADMISSE.

ADMDJCOL contains multiple instances of the ADMMCOLT macro, each of which describes the specific patterns to be used for each GDDM color.

For guidance on composed-page printing and the use of color-separation masters, see the *GDDM Application Programming Guide, Volume 1*.

The ADMMCOLT macro

The syntax of the macro invocation is:

```
&NAME ADMMCOLT codes,    START | END | Pattern codes
               SETID=,   Set ID
               PATTERN=, Pattern set
               SETS=,    Number of sets defined
               COLORS=,  Number of colors
               MASTERS=  Number of masters per color
```

codes	START For initial invocation. END For final invocation. (x1,x2,x3,xi,...,xn) For all intermediate invocations. Where xi is a 2-digit hex code that identifies the pattern in the pattern symbol set to be used by color master i, and n is the total number of masters.
SETID	Name of set (up to 8 characters of the form ADMnnnnn, where nnnnn is in the range 00001 through 99999). GDDM supplies seven sets (ADM00001, ADM00002, ADM00003, ADM00004, ADM00005, ADM00006, ADM00007).
PATTERN	Name of pattern symbol set. This must be a monochrome image character set having a cell size of 32 by 32 pixels. GDDM supplies a sample called ADMDHIPK.
SETS	The number of sets defined.
COLORS	The number of colors defined for this set.
MASTERS	The number of masters to be created for this set.

You may have to contact your systems programmer to help you install the modified color-master table; however, first see the *GDDM Installation and System Management* manual that applies to the subsystem in use.

The ADMDJCOL module

The ADMDJCOL module supplied by IBM provides seven color master tables, as shown below:

```
ADMDJCOL CSECT  ,
***********************************************************************
*                                                                     *
* DESCRIPTIVE NAME:  GDDM HIGH-RESOLUTION IMAGE GENERATOR             *
*                    DEFAULT COLOR TABLES                             *
*                                                                     *
*                 5664-200,5665-356,5666-328                          *
*                 (C) COPYRIGHT IBM CORP. 1979, 1986.                 *
*                 LICENSED MATERIALS - PROPERTY OF IBM                *
*                                                                     *
* FUNCTION:                                                           *
*                                                                     *
*   THIS MODULE GENERATES THE SAMPLE COLOR TABLES FOR THE COLOR       *
*   SEPARATION PROCESS IN FAMILY-4 DEVICE SUPPORT.                    *
*                                                                     *
*   ADMDHIPK PATTERN CODES HAVE THE FOLLOWING MEANING:                *
*   ----------------------------------------------                    *
*   CODE X'41' =   0 % (NO COLOR)                                     *
*   CODE X'42' = 100 % (SOLID COLOR)                                  *
*   CODE X'43' =  50 % (1ST HALF COLOR)                               *
*   CODE X'44' =  50 % (2ND HALF COLOR)                               *
*   CODE X'45' =  25 % (1ST QUARTER COLOR)                            *
*   CODE X'46' =  25 % (2ND QUARTER COLOR)                            *
*   CODE X'47' =  25 % (3RD QUARTER COLOR)                            *
*   CODE X'48' =  25 % (4TH QUARTER COLOR)                            *
*                                                                     *
*   ADMDHIPL PATTERN CODES HAVE THE FOLLOWING MEANING:                *
*   ----------------------------------------------                    *
*                                                                     *
*   THERE ARE 33 SHADES OF GRAY.  THE PATTERN CODES START AT X'41'    *
*   (NO COLOR) AND FINISH AT X'61' (ALL BLACK).  EACH SHADE HAS       *
*   APPROXIMATELY 3% MORE PIXELS SET ON THAN ITS PREDECESSOR.         *
*                                                                     *
***********************************************************************
ADMDJCOL AMODE ANY
ADMDJCOL RMODE ANY
         ADMMCOLT START,SETS=7
*
***********************************************************************
* TABLE 1.  SUBTRACTIVE COLORS FOR PRINTERS                           *
***********************************************************************
*
ADM00001 ADMMCOLT PATTERN=ADMDHIPK,COLORS=10,MASTERS=4,SETID=ADM00001

*
*                *-------------*
* COLOR MASTER:  * 1  2  3  4  *
* COLOR SEPS:    * YY MM CC BB * (YELLOW, MAGENTA, CYAN, BLACK)
*                *-------------*
*
DEFAULT  ADMMCOLT (41,41,41,42)
BLUE     ADMMCOLT (41,43,44,41)
RED      ADMMCOLT (43,44,41,41)
PINK     ADMMCOLT (41,42,41,41)
GREEN    ADMMCOLT (43,41,44,41)
TURQSE   ADMMCOLT (41,41,42,41)
YELLOW   ADMMCOLT (42,41,41,41)
NEUTRAL  ADMMCOLT (41,41,41,42)
BACKGRD  ADMMCOLT (41,41,41,41)
ALLBLK   ADMMCOLT (42,42,42,42)
```

```
*
*************************************************************************
* TABLE 2.  ADDITIVE COLORS FOR DISPLAYS                                *
*************************************************************************
*
ADM00002 ADMMCOLT PATTERN=ADMDHIPK,COLORS=9,MASTERS=3,SETID=ADM00002

*
*                *----------*
* COLOR MASTER:  * 1  2  3  *
* COLOR SEPS:    * RR BB GG * (RED, BLUE, GREEN)
*                *----------*
*
DEFAULT  ADMMCOLT (42,42,42)
BLUE     ADMMCOLT (41,42,41)
RED      ADMMCOLT (42,41,41)
PINK     ADMMCOLT (42,42,41)
GREEN    ADMMCOLT (41,41,42)
TURQSE   ADMMCOLT (41,42,42)
YELLOW   ADMMCOLT (42,41,42)
NEUTRAL  ADMMCOLT (42,42,42)
BACKGRD  ADMMCOLT (41,41,41)
```

```
*
***************************************************************************
* TABLE 3.  GENERAL COLOR MASTER TABLE                                    *
***************************************************************************
*
ADM00003 ADMMCOLT PATTERN=ADMDHIPK,COLORS=256,MASTERS=8,SETID=ADM00003

*
*                 *-------------------------*
* COLOR MASTER:   * 1  2  3  4  5  6  7  8  *
* COLOR SEPS:     * ** ** ** ** ** ** ** ** *
*                 *-------------------------*
*
COL00    ADMMCOLT (41,41,41,41,41,41,41,41)
COL01    ADMMCOLT (42,41,41,41,41,41,41,41)
COL02    ADMMCOLT (41,42,41,41,41,41,41,41)
COL03    ADMMCOLT (42,42,41,41,41,41,41,41)
COL04    ADMMCOLT (41,41,42,41,41,41,41,41)
COL05    ADMMCOLT (42,41,42,41,41,41,41,41)
COL06    ADMMCOLT (41,42,42,41,41,41,41,41)
COL07    ADMMCOLT (42,42,42,41,41,41,41,41)
COL08    ADMMCOLT (41,41,41,42,41,41,41,41)
COL09    ADMMCOLT (42,41,41,42,41,41,41,41)
COL0A    ADMMCOLT (41,42,41,42,41,41,41,41)
COL0B    ADMMCOLT (42,42,41,42,41,41,41,41)
COL0C    ADMMCOLT (41,41,42,42,41,41,41,41)
COL0D    ADMMCOLT (42,41,42,42,41,41,41,41)
COL0E    ADMMCOLT (41,42,42,42,41,41,41,41)
COL0F    ADMMCOLT (42,42,42,42,41,41,41,41)
*
 .
 .  The first and last 16 values of a binary progression are shown
 .
*
COLF0    ADMMCOLT (41,41,41,41,42,42,42,42)
COLF1    ADMMCOLT (42,41,41,41,42,42,42,42)
COLF2    ADMMCOLT (41,42,41,41,42,42,42,42)
COLF3    ADMMCOLT (42,42,41,41,42,42,42,42)
COLF4    ADMMCOLT (41,41,42,41,42,42,42,42)
COLF5    ADMMCOLT (42,41,42,41,42,42,42,42)
COLF6    ADMMCOLT (41,42,42,41,42,42,42,42)
COLF7    ADMMCOLT (42,42,42,41,42,42,42,42)
COLF8    ADMMCOLT (41,41,41,42,42,42,42,42)
COLF9    ADMMCOLT (42,41,41,42,42,42,42,42)
COLFA    ADMMCOLT (41,42,41,42,42,42,42,42)
COLFB    ADMMCOLT (42,42,41,42,42,42,42,42)
COLFC    ADMMCOLT (41,41,42,42,42,42,42,42)
COLFD    ADMMCOLT (42,41,42,42,42,42,42,42)
COLFE    ADMMCOLT (41,42,42,42,42,42,42,42)
COLFF    ADMMCOLT (42,42,42,42,42,42,42,42)
```

```
*
***********************************************************************
* TABLE 4.  SUBTRACTIVE COLORS FOR PRINTERS WITH CLUSTER PATTERNS     *
***********************************************************************
*
ADM00004 ADMMCOLT PATTERN=ADMDHIPL,COLORS=17,MASTERS=1,SETID=ADM00004

*                             *-------*
*                             * GDDM  *
*                             * COLOR *
*                             *-------*
*
DEFAULT  ADMMCOLT (43)            0
BLUE     ADMMCOLT (59)            1
RED      ADMMCOLT (51)            2
PINK     ADMMCOLT (4A)            3
GREEN    ADMMCOLT (55)            4
TURQ     ADMMCOLT (4E)            5
YELLOW   ADMMCOLT (45)            6
NEUTRAL  ADMMCOLT (41)            7
BACKGRD  ADMMCOLT (61)            8
DKBLUE   ADMMCOLT (59)            9
ORANGE   ADMMCOLT (55)            10
PURPLE   ADMMCOLT (51)            11
DKGREEN  ADMMCOLT (59)            12
TURQSE   ADMMCOLT (45)            13
MUSTARD  ADMMCOLT (4E)            14
GRAY     ADMMCOLT (44)            15
BROWN    ADMMCOLT (57)            16
*
***********************************************************************
* TABLE 5.  SUBTRACTIVE COLORS FOR PRINTERS WITH CLUSTER PATTERNS     *
* (THIS IS THE BVBTSO DEFINITION WITH APPROX 6% BETWEEN SHADES)       *
***********************************************************************
*
ADM00005 ADMMCOLT PATTERN=ADMDHIPL,COLORS=17,MASTERS=1,SETID=ADM00005

*                             *-------*  *--------*
*                             * GDDM  *  * PIXELS *
*                             * COLOR *  *   %    *
*                             *-------*  *--------*
*
DEFAULT  ADMMCOLT (42)            0         3.1
BLUE     ADMMCOLT (5C)            1        84.3
RED      ADMMCOLT (58)            2        71.8
PINK     ADMMCOLT (54)            3        59.3
GREEN    ADMMCOLT (51)            4        50.0
TURQ     ADMMCOLT (47)            5        18.7
YELLOW   ADMMCOLT (4C)            6        34.3
NEUTRAL  ADMMCOLT (41)            7        00.0
BACKGRD  ADMMCOLT (61)            8       100.0
DKBLUE   ADMMCOLT (56)            9        65.6
ORANGE   ADMMCOLT (4F)            10       43.7
PURPLE   ADMMCOLT (4A)            11       28.1
DKGREEN  ADMMCOLT (53)            12       56.2
TURQSE   ADMMCOLT (45)            13       12.5
MUSTARD  ADMMCOLT (49)            14       25.0
GRAY     ADMMCOLT (43)            15        6.2
BROWN    ADMMCOLT (5A)            16       78.1
```

```
*
************************************************************************
* TABLE 6. COLOR TONING SET FOR 3800/3820 PRINTERS                     *
************************************************************************
*
ADM00006 ADMMCOLT PATTERN=ADMDHIPL,COLORS=17,MASTERS=1,SETID=ADM00006

*                              *-------*
*                              * GDDM  *
*                              * COLOR *
*                              *-------*
*
DEFAULT  ADMMCOLT (61)              0
BLUE     ADMMCOLT (5A)              1
RED      ADMMCOLT (54)              2
MAGENTA  ADMMCOLT (4A)              3
GREEN    ADMMCOLT (4F)              4
CYAN     ADMMCOLT (46)              5
YELLOW   ADMMCOLT (42)              6
NEUTRAL  ADMMCOLT (61)              7
BACKGRD  ADMMCOLT (41)              8
DKBLUE   ADMMCOLT (5D)              9
ORANGE   ADMMCOLT (4C)             10
PURPLE   ADMMCOLT (51)             11
DKGREEN  ADMMCOLT (56)             12
TURQSE   ADMMCOLT (47)             13
MUSTARD  ADMMCOLT (49)             14
GRAY     ADMMCOLT (44)             15
BROWN    ADMMCOLT (58)             16
*
************************************************************************
* TABLE 7. COLOR TONING SET FOR 4250 PRINTER                           *
************************************************************************
*
ADM00007 ADMMCOLT PATTERN=ADMDHIPL,COLORS=17,MASTERS=1,SETID=ADM00007

*                              *-------*
*                              * GDDM  *
*                              * COLOR *
*                              *-------*
*
DEFAULT  ADMMCOLT (61)              0
BLUE     ADMMCOLT (55)              1
RED      ADMMCOLT (4D)              2
MAGENTA  ADMMCOLT (47)              3
GREEN    ADMMCOLT (49)              4
CYAN     ADMMCOLT (44)              5
YELLOW   ADMMCOLT (42)              6
NEUTRAL  ADMMCOLT (61)              7
BACKGRD  ADMMCOLT (41)              8
DKBLUE   ADMMCOLT (59)              9
ORANGE   ADMMCOLT (48)             10
PURPLE   ADMMCOLT (4B)             11
DKGREEN  ADMMCOLT (4F)             12
TURQSE   ADMMCOLT (45)             13
MUSTARD  ADMMCOLT (46)             14
GRAY     ADMMCOLT (43)             15
BROWN    ADMMCOLT (51)             16
*
         ADMMCOLT END
         END
```

Chapter 11. Application data structure for mapping

The basic purpose of the application data structure is to define an input/output area for use in transferring data between your application program and GDDM. You include the application data structure declaration created by GDDM-Interactive Map Definition (GDDM-IMD) in your application to define the layout of one or more areas of storage. GDDM also keeps a copy in its own storage of the data area associated with each mapped field that you define, and it uses its copy to create the display that the operator sees, and to record the changes made by the operator.

Your program modifies the GDDM data area by filling in values in its own area, then passing the area to GDDM using an MSPUT call. It finds out the values in the GDDM data area by using an MSGET call, which copies the GDDM area into the program's data area. Usually, MSGET is used so that the program gets access to the operator's input, though it can be used at other times; for example, after MSDFLD, to initialize the program's data area to the default values.

When you have finished the GDDM-IMD map-definition and generation processes, you will not only have one or more generated mapgroups, but you will also have an application data structure for each map. The data structure and the fields that it defines depends on the selections made during the map-definition process. Full details of this process are given in the *GDDM Interactive Map Definition* manual.

The application data area can be used for these purposes:

- Most of an application data structure is data fields, each data field corresponding to a map-defined display field. You place into the data fields the character data that you want to be displayed.
- You can position the cursor in a display field by setting the field's **cursor adjunct**. By default, the cursor is placed under the first character of the field, but you can change this by using the MSCPOS call before you use MSPUT.
- **Selector adjuncts** provide additional control over, and information about, a field's data value. You can selectively update a field, reset a field to its map-defined default value, and determine whether a field has been modified by the operator.
- **Length Adjuncts** show the length of the data in the field. If the data in a field is shorter than the map-defined display field length, GDDM pads the data with nulls when it displays the field. After operator input the length adjunct is set to the number of characters provided by the operator.
- Usually, field attributes are specified for the various fields on a map during map definition. However, at run time the application program can change these attributes by placing attribute values in **attribute adjunct fields** in the application data structure. One or more adjunct fields can be associated with a given data field in the application data structure during map definition. Each attribute adjunct controls a different type of attribute.
- Some devices allow different attributes to be applied to individual characters in the same field. Character attributes are controlled using a separate copy of the application data area. The data fields in this copy contain the character attribute data instead of the normal character data. Each character in the character attribute data area determines the attributes of the corresponding character in the normal application data area.
- The application program can be designed to allow detection (or selection) of fields in a displayed panel by a light pen or, on some devices, the Cursor Select (CURSR SEL) key. The type of detection that occurs is determined by the first data character in the field; this character is called a **designator character**.
- If specified in the map during map definition, GDDM edits input data entered by the terminal operator. To process this edited input, you need to know how GDDM presents it in the application data structure.

This chapter gives valid settings and explanations of adjunct fields, character attributes, and designator characters, and describes the format of edited input. It also describes how to copy the application data structure into the application program.

Adjunct fields

Each data field in the application data structure may have associated adjunct fields, depending on the options selected during the Field Naming step of map definition. The possible adjunct fields for a data field are shown below. They appear in the data structure in the order given, immediately before the data field.

Adjunct	Length (bytes)
Selector	1
Cursor	1
Base attribute	2
Extended highlighting	2
Color	2
Programmed symbols (PS)	2
Validation	2
Outlining	2
Length	2

The base attribute, extended highlighting, color, PS, validation, and outlining adjunct fields shown above are each subdivided into two one-byte fields. In each case, the first byte acts as a selector to let GDDM know whether or not the value held in the second byte is to be used during program execution.

COBOL example

Suppose that in the Map Characteristics frame (2.1) of GDDM-IMD, you entered:

```
PROGRAM LANGUAGE ==> COBOL
```

Next, suppose that in the Application Structure Review frame (2.5), you are defining the characteristics of a data field that you have named SPECNAME. You want to be able to:

1. Set the cursor in the field under application program control
2. Have dynamic control of extended highlighting
3. Specify the length of data in the field.

You therefore enter "#HL" in the ADJUNCT column against the field name.

As a result of this entry, the application data structure contains, for the field SPECNAME, a cursor adjunct (1 byte), a highlighting adjunct (2 bytes), a length adjunct (2 bytes), plus the data field itself, whose length is as defined in the map (say 25 bytes).

GDDM-IMD names the adjunct fields by suffixing the data field name supplied by the user. So, for example, the cursor adjunct field is named SPECNAME-CURSOR.

The portion of the application data structure that is generated for SPECNAME is:

```
10  SPECNAME-CURSOR      PIC X.
10  SPECNAME-HI-SEL      PIC X.
10  SPECNAME-HI          PIC X.
10  SPECNAME-LENGTH      PIC 999 COMP.
10  SPECNAME             PIC X(25).
```

Assembler language example

Assume that instead of entering COBOL as the program language in the above example, you enter ASM, and, to comply with Assembler-language length restrictions, you name the data field SPEC. The generated code (assuming the other selections were the same as those given above) is:

```
SPECCR    DS    X
SPECHS    DS    X
SPECH     DS    X
SPECL     DS    AL2
SPEC      DS    XL25
```

PL/I example

Similarly, if you use PL/I as the program language and call the data field SPECNAME, the generated code is:

```
10 SPECNAME_CURSOR CHAR(1),
10 SPECNAME_HI_SEL CHAR(1),
10 SPECNAME_HI CHAR(1),
10 SPECNAME_LENGTH FIXED BIN(15),
10 SPECNAME CHAR(25),
```

Adjunct field names

The above examples show that GDDM-IMD suffixes the name you have given to a data field to create unique names for each adjunct field in the application data structure. The full set of suffixes that GDDM-IMD uses for COBOL, Assembler, and PL/I data structures is shown in Table 11.

Adjunct values

Table 12 on page 87 summarizes valid settings for adjunct fields. Details are given for each type of adjunct on the following pages.

The application program sets the values required for a send request. GDDM sets the values associated with input data returned for a receive request. On a send request, each field **must** contain one of the settings given for it in Table 12.

Table 11. Adjunct field naming conventions

Adjunct	Length	COBOL name	Assembler name	PL/I name
Selector	1	XXX-SEL	XXXS	XXX_SEL
Cursor	1	XXX-CURSOR	XXXCR	XXX_CURSOR
Base attribute	1 1	XXX-ATTR-SEL XXX-ATTR	XXXAS XXXA	XXX_ATTR_SEL XXX_ATTR
Extended highlighting	1 1	XXX-HI-SEL XXX-HI	XXXHS XXXH	XXX_HI_SEL XXX_HI
Color	1 1	XXX-COL-SEL XXX-COL	XXXCS XXXC	XXX_COL_SEL XXX_COL
PS	1 1	XXX-PS-SEL XXX-PS	XXXPS XXXP	XXX_PS_SEL XXX_PS
Validation	1 1	XXX-VAL-SEL XXX-VAL	XXXVS XXXV	XXX_VAL_SEL XXX_VAL
Outlining	1 1	XXX-OUT-SEL XXX-OUT	XXXOS XXXO	XXX_OUT_SEL XXX_OUT
Length	2	XXX-LENGTH	XXXL	XXX_LENGTH

Table 12 (Page 1 of 2). Values used in adjunct fields

Adjunct	Value (See Note 1)	Meaning
Selector	C' '	MSPUT: Any data value is ignored. The field is unchanged (see Note 2).
		MSGET: Neither the application nor the operator has put a value in it.
	C'1'	MSPUT: The field contains a value.
		MSGET: The field contains a value that the operator has just modified.
	C'2'	MSPUT: The field is to be reset to its map-defined default value. The data value is ignored. On a subsequent MSGET, the field contains its default value and the selector is C'3'.
		MSGET: not used.
	C'3'	MSPUT: The field contains a value. (that is, the same as C'1').
		MSGET: The field contains a value that has not just been modified by the operator.
Cursor	C' '	MSPUT: The cursor is not in this field.
		MSGET: The cursor is not in this field.
	C'1'	MSPUT: The cursor is in this field.
		MSGET: The cursor is in this field (set only if map is a cursor receiver).
	The position within the field can be controlled by using the MSCPOS call, and verified by using the MSQPOS call.	
Attribute Selector (first byte of adjunct)	C' '	The attribute is unchanged (see Note 2). The attribute byte (the second byte) is ignored.
	C'1'	Change the attribute to the value in the second byte.
	C'2'	Reset the attribute to the map-defined default value.
	C'3'	Change the attribute to the value in the second byte (same as C'1'). After an MSGET, the attribute selector is set to C'3' and the attribute byte set to the current attribute value.
Attribute Value (second byte of attribute adjunct)	Ignored unless the attribute selector is C'1' or C'3'. Otherwise, the valid value depends on the attribute type, as follows:	
	C' ' X'00'	Default for all attributes.
Base attribute	A valid 3270 attribute if used	These values are defined mnemonically in ADMUAIMC (Assembler), ADMUCIMC (COBOL), and ADMUPIMC (PL/I).
	For example:	C' ' Unprotected
		C'H' Unprotected, Intensified
		C'-' Protected
		C'Y' Protected, Intensified
		C'0' Autoskip
		B'xx......' Ignored (set by GDDM)
		B'..1.....' Protected
		B'..0.....' Unprotected
		B'...0....' Alphanumeric
		B'..01....' Unprotected numeric
		B'..11....' Autoskip
		B'....00..' Normal
		B'....01..' Selectable
		B'....10..' Intensified selectable
		B'....11..' Nondisplay
		B'......x.' Ignored (set by GDDM)
		B'.......1' Modified data tag set
		B'.......0' Modified data tag not set

Table 12 (Page 2 of 2). Values used in adjunct fields

Adjunct	Value (See Note 1)	Meaning
Extended highlighting attribute	C' '	No extended highlighting.
	C'1'	Blinking.
	C'2'	Reverse video.
	C'4'	Underscore.
Color attribute	C' '	Default.
	C'1'	Blue.
	C'2'	Red.
	C'3'	Magenta (pink).
	C'4'	Green.
	C'5'	Turquoise (cyan).
	C'6'	Yellow.
	C'7'	White/Neutral.
PS attribute	C' '	Default character set.
	X'41'-X'DF'	PS code of any symbol set specified in PS Set Management in GDDM-IMD, or loaded using PSDSS, PSLSS, or PSLSSC.
Validation attribute	C' '	No validation.
	X'00'	No validation.
	X'01'	Trigger.
	X'02'	Mandatory enter.
	X'04'	Mandatory fill.
	These values can be ORed together to give two or more validation attributes to the same field. For example, specify X'03' to give a field the mandatory enter and trigger attributes (X'02' OR X'01' = X'03').	
Outlining attribute	C' '	No outlining.
	X'00'	No outlining.
	X'01'	Underline.
	X'02'	Vertical line on right.
	X'04'	Overline.
	X'08'	Vertical line on left.
	These values can be ORed together to give two or more outlining attributes to the same field. For example, specify X'03' to give a field with underlining and a vertical line on the right (X'02' OR X'01' = X'03').	
Length	Binary value	Length, in characters, of the data.

Notes:

1. In Table 12 on page 87, "C" indicates character data type, "X" indicates hexadecimal, and "B" indicates bit.
2. On an MSPUT call with option 0 ("WRITE"), all fields and attributes are reset to their map-defined default value, before the application data area is processed. Therefore, an attribute selector or field selector of C' ' has the net effect of resetting the value to default, when used on an MSPUT call with option 0, or of leaving the value unchanged, when used on an MSPUT call with option other than zero.

Selector adjunct

The selector adjunct provides additional control over an individual field in the application data structure, and shows, after an MSGET, whether the data field has just been modified by the operator.

The control function is most useful when using MSPUT with option 1 (REWRITE) or 2 (REJECT), particularly if the application program does not maintain a complete copy of the application data area. A partially completed application structure can be used. Fields whose selector is blank are ignored and so need not be set by the application. Their value is unchanged. Fields whose selector is C'1' or C'3' are processed by placing the current data value in GDDM's copy of the data area with the value from the program's data area.

Note: Fields that do not have a selector are always processed.

The control function can also be used to set a field to its map-defined default value. This is the constant text placed into the field during GDDM-IMD's Field Definition or Field Initialization steps. This is the value of the field immediately after a mapped field is defined by MSDFLD. Note that if the field has no selector, any MSPUT call replaces this default value with the value from the application data area (even if the field is all blanks). If the field has a selector adjunct, its value can be reset to the map-defined default value by specifying a selector of C'2' on any MSPUT call.

Notes:

1. The map-defined default character attributes are always "default." GDDM-IMD does not support character attributes.
2. An MSPUT with option 0 (WRITE) sets all fields (attributes and so on) back to their default value before processing the application data area.

When a selector value of C'2' is specified, GDDM converts it into C'3', and places the default field value into the data field in GDDM's copy of the data area so that the program can access it using MSGET. (The program's data area is **not** modified during an MSPUT.)

After an MSGET, a selector adjunct shows whether the field has just been modified by the operator. A value of C'1' shows that the field has been modified by one of these events:

- The operator has typed into the field
- The operator has selected the field with a light-pen (if the field is selectable)
- The field has been set by AID translation.

Note: "Modified" includes the degenerate case of the operator modifying the field back to its original value.

Usually, modification indicators are reset when the operator is next given an opportunity to enter data (for example, an ASREAD). Your program can avoid this resetting by issuing an MSPUT call with option 2 (REJECT) on any map within the page, before the ASREAD call.

Cursor adjunct

The cursor adjunct is used to set the cursor in a field dynamically (thus overriding any static cursor setting specified in the map), and to show whether the cursor was left in a field on input.

Static setting of the cursor is specified in the Field Attribute Definition step of GDDM-IMD; for details, see the *GDDM Interactive Map Definition* manual.

To set the cursor in a field dynamically, the application program sets the associated cursor adjunct to 1. This causes the cursor to be placed in the field when the field is displayed. By default, the cursor is placed under the first character of the field. To position the cursor elsewhere, use the MSCPOS call to specify the position, just before issuing the MSPUT call. The position is a number between 0 and the length of the field, thus:

0	Means "under the attribute byte"
1	Means "under the first character"
2	Means "under the second character" and so on.

When the position specified is greater than the length of the field, GDDM places the cursor under the last character in the field.

GDDM places the cursor at the last dynamic setting it meets for a page. In the absence of any dynamic settings, GDDM places the cursor at the first static setting.

To determine the position of the cursor on input, the map must have been defined as a cursor-receiver map in the Map Characteristics step of map definition. If the map has been so defined, GDDM sets the cursor adjunct of the field in which the cursor lies to C'1' when the field has a cursor adjunct. The position of the cursor within the field can be found using the MSQPOS call **after** the MSGET call.

Note: The "cursor-receiver" map characteristic is provided so that applications that use cursor adjuncts only for output cursor control do not have to search for, and turn off, cursor adjuncts after an MSGET call. If the cursor adjuncts were left on, GDDM might misinterpret the application's intention when the application data area is next used in an MSPUT call.

Attribute adjuncts

An Attribute adjunct is used to change the attribute of a field from its map-defined default value. There are several types of attribute adjuncts; one "base" attribute that controls a compound set of basic field properties, and one attribute type for each of a set of "extended" properties.

Each attribute adjunct field consists of two subfields; an attribute selector byte, and an attribute value byte. The valid values for the attribute selector are the same for all attribute adjuncts (and the same as those for a Field Selector):

C' '	Ignore the value provided. Leave the attribute at its current value. If the mapped field has just been defined, or if the operator is an MSPUT with option 0 (WRITE), the current value is the map-defined value. Otherwise, the value is that set by previous MSPUT operations.

C'1' Change the attribute to that specified in the attribute adjunct.

C'2' Reset the attribute to the map-defined value.

C'3' Change the attribute to that specified in the attribute adjunct (that is, the same as C'1'). After an MSGET, all attribute adjunct selectors are set to C'3', and the attribute value byte is set to the current attribute value.

The second byte of an attribute adjunct is the value to be used (for selector value C'1' and C'3'). The range of valid values is dependent on the attribute type.

GDDM provides, as part of GDDM-IMD, a set of declarations in Assembler, COBOL, and PL/I, for the values that can be used in attribute adjuncts. These are in the files ADMUAIMC (Assembler), ADMUCIMC (COBOL) and ADMUPIMC (PL/I) in the GDDM Sample Library.

Note that all attribute adjunct types can be used on all devices supported by GDDM for mapping, but they have no effect on the presentation if the device does not support the corresponding function.

Base attribute adjunct

Base attributes are the basic (as opposed to extended) field attributes that are supported by all display devices supported by GDDM. They can be specified for individual fields on a map during map definition and reset during program execution by base-attribute adjuncts. They include:

- Protected/unprotected/autoskip
- Intensified-display/normal-display/nondisplay
- Detectable/nondetectable
- MDT bit on/off
- Alphanumeric/numeric.

The attribute adjunct value byte can contain any valid IBM 3270 basic attribute code. GDDM sets the reserved and meaningless bits of the attribute correctly, so all one-byte values are accepted.

The base attribute adjunct value byte completely specifies the combination of base attributes to be used for the field on the device. It is not merged in any way with previous base-attribute specifications for the field, or with the value specified in the associated map.

Extended highlighting adjunct

The extended highlighting adjunct can be used by the application program to override any extended highlighting attribute defined for a field in the map. Extended highlighting is available only on specific devices, and can be used in addition to the intensification control of the base attribute. It lets you specify whether a field should blink, be underscored, or be displayed in reverse video.

Possible settings in the attribute adjunct value byte are as shown in Table 12 on page 87.

Color adjunct

Possible settings in the attribute adjunct value byte are again as shown in Table 12.

Note that this adjunct cannot be used to control color on devices whose color is determined by means other than the color extended attribute. For example, it can be used to control color on seven-color display devices, but not on four-color display devices.

Programmed symbols adjunct

The programmed symbols (PS) adjunct lets you specify that the special characters and symbols defined in a given symbol set apply for the field associated with the PS adjunct in the application data structure. You can define your own symbol sets using the Image Symbol Editor, as described in the *GDDM Image Symbol Editor* manual. You can also use the predefined symbol sets supplied by IBM.

Your application program can use characters from a particular symbol set only if that symbol set is loaded into a PS store in the device. A symbol set can be loaded when defining a mapgroup containing maps that use symbol sets. You can specify that the symbol sets are to be loaded automatically by GDDM when a MSPCRT request naming the mapgroup is issued; for more details, see the *GDDM Interactive Map Definition* manual. Symbol sets loaded in this way are available to the application program for the life of the page.

The required symbol set is identified by the PS code (or PSID), which is a single-character identifier in the range X'41' through X'DF', designated by you or your installation. The PS code is designated when the mapgroup is defined, if GDDM is to handle symbol-set loading, or during the loading operation, if your application program or your installation is handling symbol-set loading directly.

Validation adjunct

The validation attribute is supported only by the IBM 8775 Display Terminal (with the appropriate feature). On all other devices it is ignored.

Possible settings in the attribute adjunct value byte are as shown in Table 12. The IBM 8775 handles operator input according to the validation attribute, as follows:

1 **Mandatory Enter Attribute**
If the operator tries to transmit data (for example, by pressing the ENTER key) while there is a mandatory enter field that has not had data entered into it, the transmission fails and input is inhibited. The cursor is repositioned to the start of the first empty mandatory enter field. The operator can proceed by pressing the RESET key. Then, the operator can either enter data in the mandatory enter field, or use the ERASE EOF or Error Override key to set the MDT. For the Error Override key, an error value (X'3F') is returned to the application program in the mandatory enter field.

2 **Mandatory Fill Attribute**
If data is entered into a mandatory fill field, the field must be completely filled before the cursor can be moved out of it. If an attempt is made to move the cursor out of the field before it has been filled, further input is inhibited.

The operator can proceed by pressing the RESET key, and completing the entry of data into the mandatory fill field. Or, the Error Override key can be used to fill the field with error values (X'3F') before continuing.

3 **Trigger Fill Attribute**
The trigger field attribute enables the application program to receive data entered into a particular field as soon as the data entry for that field is complete and the cursor leaves the field. The operator can continue keying data while the trigger field is being checked, but the data entered is placed on a queue in the device (and is not displayed).

Cursor exit from a modified trigger field causes the inbound transmission of this single field with a "trigger" AID. The application can access the trigger field data in the usual way using MSGET.

The application program must then decide whether to accept the trigger field (and hence the operator's queued keystrokes) by issuing a positive acknowledgment, or to reject the field (and lose the operator's queued keystrokes) by issuing a negative acknowledgment.

A positive acknowledgment is generated by issuing an MSPUT call specifying that the keyboard is to be unlocked. By default, this is true of options 0 (WRITE) and 1 (REWRITE).

A negative acknowledgment is generated by issuing an MSPUT call specifying that the keyboard is to remain locked. By default, this is true of option 2 (REJECT).

Note: The relationship between the MSPUT option and locking the keyboard is defined in GDDM-IMD's Map Characteristics step.

| **Field outlining**

| Outlining is only available on specific devices; if the
| device does not support outlining the adjunct is
| ignored. Possible settings in the attribute adjunct byte
| are shown in Table 12 on page 87.

Length adjunct

The length adjunct is a two-byte field that can contain values in the range 0 through the length of the field. It indicates the length of the data in the data field. GDDM treats a value greater than the field length as if it were equal to the field length.

When a field is displayed, GDDM pads the data with nulls, from the length specified in the length adjunct, to the length of the display field.

After the operator modifies a field, the length adjunct specifies the number of bytes of data placed in this field by the input operation.

If right-hand justification has been specified for the field during map definition, the length adjunct is set on input to the length of the field in the application data structure. If left-hand justification has been specified, the length adjunct is set to the number of characters in the field up to the first padding character.

Character attributes

Highlighting, color, and PS attributes can be specified for individual characters within a field. Usually, character attributes are used to emphasize a particular character string in a field.

Note: GDDM supports character attributes in mapped variable fields, but not in constant or initial values held in the map.

To control any type of character attribute, the program needs an additional application data area. This area has the same structure as the usual application data area (including adjunct fields), but the data fields are interpreted as character attributes rather than character data.

To declare several data areas using the same structure, you can use an array of structures or (in PL/I) the LIKE attribute.

COBOL

```
01  ALLAREAS.
 02  DATA-AREA OCCURS 3 TIMES.
    COPY MAP.
```

PL/I

```
Declare
    1 DATA_AREA,
      %INCLUDE MAP;
Declare
    1 COLOR_AREA LIKE DATA_AREA;
```

In the former case, the individual application areas (and fields and adjuncts within them) can be referred to using an array index. In the second case, they can be referenced using name qualification (DATA_AREA.FIELD1, COLOR_AREA.FIELD1, and so on).

The character attribute data areas are filled in the same way as are the usual application data areas, except that the data fields contain characters representing attributes. For example:

```
DATA_AREA.FIELD1 ='data value';
COLOR_AREA.FIELD1='1111111121';
```

Adjunct fields in the character attribute application data area have the same meaning as in the normal data area. Selector and Length adjuncts apply to the character attribute data field.

Each application data area is passed to GDDM with a separate MSPUT call. The character attribute type is specified as an option on MSPUT. The character attributes should be MSPUT **after** the data values, because changing the data value of any field automatically resets the character attributes of the field to the default value (C' '). Also, an MSPUT with option 0 (WRITE) resets all the character attributes of all fields in the map to default.

The allowable attribute types and attribute values are listed in Table 3 92. GDDM checks attribute types and does not transmit those that the device does not support. Invalid attribute values are rejected.

Table 13. Character attribute types and values

Type	Value	Meaning
All	X'00' C' '	Default. Take the attribute value from field's attribute.
Extended highlighting	C'1' C'2' C'4'	Blinking Reverse video Underscore
Color	C'1' C'2' C'3' C'4' C'5' C'6' C'7'	Blue Red Magenta (pink) Green Turquoise (cyan) Yellow White/Neutral
Programmed symbols	X'41' through X'DF'	PS code. Note that a symbol-set must be loaded before any reference to it is made. See "Programmed symbols adjunct".

Note: In the above table, "C" indicates character data type, and "X" indicates hexadecimal.

Setting character attributes from the terminal

If the application program uses the ASMODE call and an appropriate keyboard is in use, the terminal operator can set the attributes of data characters entered from the terminal. The program can read these attributes using MSGET with the correct option.

The procedure for setting character attributes from the terminal can be found in the appropriate terminal operator's guide.

Designator characters for light-pen or cursor selection

You specify that a field can be selected by a light pen, or, on some terminals, the CURSR SEL key, by giving it a "detectable" attribute at map-definition time. The "detectable" attribute can be defined for a field using GDDM-IMD's Field Attribute Definition step, and can be controlled dynamically using the base attribute adjunct.

However, the type of selection that occurs on using the light pen is determined by the first character (the designator character) in the data field. You must set the required designator character in the first byte of the data field. If the field contains constant data, the designator character is set in the map; otherwise, it is set in the application data structure. When the field is displayed, the designator character appears on the screen along with the rest of the data in the field.

A field having a "detectable" attribute but not starting with a valid designator character is not selectable.

The types of selection that can be set are:

1. Delayed detection. When selected by the operator, the field is marked as "modified" but nothing is transmitted until the operator performs another action associated with field modification (such as selecting an "immediate detection" field or pressing ENTER). The designator character for this type of field is "?" (X'6F'). If the field is detected, the designator character changes to ">" (X'6E'); another detection restores it to "?" and cancels the modification indication.
2. Immediate detection without data. The designator character is a blank (X'40'). Selection of this type of field causes immediate input transmission. No data from any of the fields is transmitted, however. The effect is thus:
   a. The ASREAD (or MSREAD) returns an Attention Type of 2 indicating light-pen selection.
   b. If the application issues an MSGET, any field that was modified or delay-detected has its selector set to C'1'; its data value, however, is unchanged even if the operator typed into the field.
   c. GDDM restores all display fields to their original value at the next FSFRCE, ASREAD, or GSREAD.
3. Immediate detection with data. (Not possible with the IBM 3277 Display Terminal). The designator character is "&" (X'50'). The effect is the same as pressing ENTER.

For more details of the mechanics of light-pen detection and the use of designator characters, refer to the appropriate component description manual.

Map-defined input editing

Using GDDM-IMD's Field Naming or Application Data Structure Review steps, you can specify that the following transformations are to be performed automatically by GDDM on input data passed to the application program. The transformations are specified for individual fields.

- Folding: translation to uppercase of all alphabetic input entered into the field.
- Justification and padding: right- or left-alignment and padding of data entered into the field.
- Attention identifier translation: translation of the AID associated with the input transmission into a predetermined character string.

For details of how to specify these transformations on a map, see the Application Data Structure Review step of GDDM-IMD in the *GDDM Interactive Map Definition* manual. The information given in the remainder of this section relates to the application program's view of the transformed fields returned in response to a receive request.

Notes:

1. The transformations take place on input from the operator, for receipt by the application on an MSGET. Data that is placed into the application data area by the application's MSPUT and map-defined default data is not transformed, even though it may be read back using MSGET.

The effect of the transformations is not immediately visible to the operator. However, if the application does not modify the field, delete the mapped field, or delete the page, the transformed data is displayed to the operator on the next ASREAD, FSFRCE, or GSREAD call.

2. If more than one of these transformations have been specified for a given field, processing is done in this order:

   a. AID translation
   b. Folding
   c. Justification and padding.

AID translation

At map definition time, you can associate an AID translation table with an input field on a map. This field is called an "AID receiver" field.

The translation table is set up during map definition. It defines character strings for the various terminal function keys (and the light pen, trigger fields, operator ID card reader, and magnetic slot reader, if required).

When the operator uses the corresponding key, GDDM places the corresponding character string into the designated field.

AID translation is not restricted to a single field on the map. You can associate several fields with the same or different translation tables and thus receive different character strings in the fields on input.

AIDs can be specified as "do not translate," in which case, the existing field value remains unchanged. For AIDs not explicitly named in the table, a default translation value can be specified; on the other hand, these AIDs can be specified as "do not translate."

An AID receiver field can have a corresponding display field, although this is not mandatory. If the receiver field has a corresponding unprotected display field, operator input into that field is overwritten by the translated AID value unless the operator uses an interrupt key that is designated (explicitly or implicitly) "do not translate."

Folding

When specified, folding always occurs irrespective of what other attributes have been specified for the field.

The folding transformation uses the Lowercase-to-Uppercase Translation Table in the GDDM Alphanumerics Defaults Table (ADMDATRN).

Justification and padding

During map definition, you can specify that a field should be right-justified, left-justified or not justified, and, if you want, that it should be padded with a particular character. If you do not specify a padding character, defaults are used; that is, character zero for right-justified fields, blank for left-justified fields.

For right-justified fields:

1. The rightmost significant (that is, nonblank, nonnull) character is aligned with the rightmost boundary of the field in the application data structure. Leading blanks or nulls are then changed to the padding character.
2. The length adjunct (if one was specified for the field) is set to the application data structure field length.

For left-justified fields:

1. The leftmost significant (that is, nonblank, nonnull) character is aligned with the leftmost boundary of the field in the application data structure. Trailing padding characters are then added to fill the field.
2. The length adjunct (if one was specified) is set to the number of characters in the field up to the first padding character.

For fields for which no justification is specified, the input data is left unchanged (that is, leading and trailing blanks are **not** removed), and the rest of the field is filled with blanks. The length adjunct, if specified, is set to the number of characters (including leading and trailing blanks) entered by the terminal operator.

If the input data is longer than the field in the application data structure, it is truncated on the right, irrespective of any justification specification, before leading and trailing blanks are suppressed, and a warning message is issued when MSGET is used on the map.

Copying the application data structure into the program

When you have finished the map definition and generation processes, you will have an application data structure for each map, each having the same name as the associated map. You can copy these application data structures into your application program, if it is a COBOL or PL/I program.

For an Assembler program, you must include macro instructions in your program having the same names as the maps. These expand into DSECTs at assembly time.

An example showing the code that might be used for a COBOL program is given below. For illustration, assume that there is a page that is constructed from three separate maps named HEADER, DATAREC, and TRAILER. The maps belong to a mapgroup called MAPGRP.

```
01 HEADER.
   COPY HEADER.
01 DATAREC.
   COPY DATAREC.
01 TRAILER.
   COPY TRAILER.
```

Note: As part of the application structure declaration, GDDM-IMD generates a declaration of a variable with name "mapname-ASLENGTH" (COBOL) "mapname_ASLENGTH" (PL/I) that is initialized with the length, in bytes, of the application structure. This variable can be used as the length parameter in MSPUT and MSGET calls.

Overlaying application data areas

Sometimes, for programming reasons such as conserving storage, it is convenient to overlay the storage used by one of several application data structures. Generally, the structures are not the same length. In this situation, COBOL requires that the longest record description occurs first. To avoid needing to know in advance which record is the longest, you can specify

```
LARGE STRUCTURE ===> YES
```

in frame 3.0 of the generation step of GDDM-IMD. This causes GDDM-IMD to generate an additional structure in a file with the same name as the mapgroup containing a single data item of length equal to that of the largest record.

The following code in the relevant section of the COBOL program then creates the necessary overlaid record descriptions:

```
01 MAPGRP.
   COPY MAPGRP.
01 HEADER REDEFINES MAPGRP.
   COPY HEADER.
01 DATAREC REDEFINES MAPGRP.
   COPY DATAREC.
01 TRAILER REDEFINES MAPGRP.
   COPY TRAILER.
```

COBOL also has the restriction on the placement of declarations using REDEFINES. To satisfy this restriction GDDM-IMD does not generate variables initialized to the application structure length, if you request

```
LARGE STRUCTURE=YES
```

Note: If one of the maps has a name that is the same as the mapgroup name, the application data structure for that map is expanded by a dummy data item (if necessary) to make it as long as the longest application data structure.

Double-byte character string fields

Double-byte character strings (DBCS) fields are specially treated in some cases. (Double-byte character string fields are used for Kanji and Hangeul applications).

A field can be designated as DBCS by using GDDM-IMD's Field Definition steps, or Field Attribute Definition steps, or both of these.

A field can also be changed to or from DBCS by using a PS attribute adjunct and specifying a value of X'F8' (C'8') for DBCS, or C' ' (or any other valid value) for EBCDIC.

However, the special treatment of length adjuncts and cursor positioning provided for DBCS fields depends only on how the fields were defined to GDDM-IMD. Dynamically changing a field to or from DBCS does not change this treatment.

The special treatment is:

Length adjuncts
If a field is designated **at map definition time** as a DBCS field, the field's length adjunct is always interpreted as several two-byte characters. Hence, the length of the data in bytes is twice the value of the length adjunct.

Cursor position
If a field is designated **at map definition time** as a DBCS field, the cursor position specified by MSCPOS and returned by MSQPOS is interpreted as several two-byte characters. Hence, the position within the field in bytes is twice the value specified (minus 1).

Mixed double-byte and single-byte character fields in maps

Some Asian languages, including Chinese, Kanji, and Hangeul are displayed and printed using double-byte character sets (DBCS), which means that each character is represented by two bytes. European languages use Latin single-byte character sets (SBCS). The IBM 5550 Multistation and Personal System/55 work stations will display and print both SBCS and DBCS characters.

Sometimes, the two types need to be mixed in a single alphanumeric field. The 5550 and Personal System/55 allow this.

The internal representation of mixed character strings makes use of shift-out (SO) and shift-in (SI) control characters, X'0E' and X'0F', to indicate the start and end of a DBCS substring.

There are two ways of displaying mixed character strings, called mixed-with-position and mixed-without-position. The display method to be used is specified in the map definition for each field.

- Mixed-with-position

  The SO/SI codes occupy one character position each, and are displayed as either a blank or a special character – the terminal user can select which.

- Mixed-without-position.

  The SO/SI codes do not occupy a character position on the screen.

The initial input mode of the work stations is SBCS. To enter DBCS characters, the operator presses a special shift key to change the mode. After entering the DBCS string, pressing another shift key returns the terminal to SBCS mode, so further single-byte characters can be entered.

GDDM-supplied mapping constants

This section lists the contents of the GDDM-supplied declarations that contain mapping constants. By including these declarations in your program, you can simplify the setting of the second byte of attribute adjuncts by using a mnemonic name rather than a bit value.

The declarations contain mnemonically-named variables for every attribute, and for combinations of attributes. The variables are initialized to the bit patterns required in the 3270 attribute bytes.

The method of including the declarations in your program varies according to the subsystem and programming language that are being used.

Assembler mapping constants table – ADMUAIMC

```
*****************************************************************
* TABLE NAME: ADMUAIMC                                          *
*                                                               *
* ADMUAIMC: GDDM ASSEMBLER DECLARATIONS FOR MAPPING CONSTANTS   *
*                                                               *
*                  5668-801                                     *
*                  (C) COPYRIGHT IBM CORP. 1979, 1986.          *
*                  LICENSED MATERIALS - PROPERTY OF IBM         *
*                                                               *
* FUNCTION:                                                     *
*                                                               *
*   THIS TABLE DECLARES ASSEMBLER EQUATES FOR THE               *
*   SPECIAL VALUES USED BY GDDM MAPPING.                        *
*                                                               *
*****************************************************************
* MSPUT AND MSGET OPTIONS.                                      *
*****************************************************************
WRITE     EQU   0
REWRITE   EQU   1
REJECT    EQU   2
HIGHLITE  EQU   3
COLOR     EQU   4
PS        EQU   5
*
*****************************************************************
* DATA AND ATTRIBUTE SELECTOR VALUES                            *
*****************************************************************
IGNORE    EQU   C' '
EXPLICIT  EQU   C'1'
SELECTED  EQU   C'1'
MAPDEFND  EQU   C'2'
OLDVALUE  EQU   C'3'
*
*****************************************************************
* CURSOR SELECTOR VALUES                                        *
*****************************************************************
CURSED    EQU   C'1'
*
*****************************************************************
* BASE (3270) ATTRIBUTE VALUES                                  *
*****************************************************************
*
* UNPROTECTED,NO MDT BIT.
DEFAULT   EQU   C' '
DETECTBL  EQU   C'D'
BRIGHT    EQU   C'H'
DARK      EQU   C'<'
NUMERIC   EQU   C'&&'
NUMDTCT   EQU   C'M'
NUMBRT    EQU   C'Q'
NUMDARK   EQU   C'*'
*
* PROTECTED,NO MDT BIT.
PROTECT   EQU   C'-'
PRTDTCT   EQU   C'U'
PRTBRT    EQU   C'Y'
PRTDARK   EQU   C'%'
AUTOSKIP  EQU   C'0'
SKPDTCT   EQU   C'4'
SKPBRT    EQU   C'8'
SKPDARK   EQU   C'@'
*
* UNPROTECTED,MDT BIT.
MDT       EQU   C'A'
DTCTMDT   EQU   C'E'
BRTMDT    EQU   C'I'
DARKMDT   EQU   C'('
NUMMDT    EQU   C'J'
NUMDTMDT  EQU   C'N'
NUMBRMDT  EQU   C'R'
NUMDKMDT  EQU   C')'
*
* PROTECTED,MDT BIT.
PRTMDT    EQU   C'/'
PRTDTMDT  EQU   C'V'
PRTBRMDT  EQU   C'Z'
PRTDKMDT  EQU   C'_'
SKPMDT    EQU   C'1'
SKPDTMDT  EQU   C'5'
SKPBRMDT  EQU   C'9'
SKPDKMDT  EQU   C''''
*
****************************************************************
* HIGHLIGHTING ATTRIBUTE VALUES                                *
****************************************************************
NOHIGH    EQU   X'00'
BLINK     EQU   C'1'
RVIDEO    EQU   C'2'
USCORE    EQU   C'4'
*
****************************************************************
* COLOR ATTRIBUTE VALUES                                       *
****************************************************************
MONO      EQU   X'00'
BLUE      EQU   X'F1'
RED       EQU   X'F2'
PINK      EQU   X'F3'
MAGENTA   EQU   PINK
GREEN     EQU   X'F4'
TURQ      EQU   X'F5'     TURQUOISE
CYAN      EQU   TURQ
YELLOW    EQU   X'F6'
WHITE     EQU   X'F7'
*
****************************************************************
* VALIDATION ATTRIBUTE VALUES                                  *
****************************************************************
NOVALIDN  EQU   X'00'
TRIGGER   EQU   X'01'
ENTER     EQU   X'02'
TR@EN     EQU   X'03'    TRIGGER AND ENTER
FILL      EQU   X'04'
TR@FL     EQU   X'05'    TRIGGER AND FILL
EN@FL     EQU   X'06'    ENTER   AND FILL
TR@EN@FL  EQU   X'07'    TRIGGER AND ENTER AND FILL
*
```

COBOL mapping constants table – ADMUCIMC

```
*****************************************************************
* TABLE NAME: ADMUCIMC                                          *
*                                                               *
* ADMUCIMC: GDDM COBOL DECLARATIONS FOR MAPPING CONSTANTS       *
*                                                               *
*                5668-801                                       *
*                (C) COPYRIGHT IBM CORP. 1979, 1986             *
*                LICENSED MATERIALS - PROPERTY OF IBM           *
*                                                               *
* FUNCTION:                                                     *
*                                                               *
*   THIS TABLE DECLARES COBOL VARIABLES INITIALIZED TO THE      *
*   SPECIAL VALUES USED BY GDDM MAPPING.                        *
*                                                               *
*****************************************************************
 01  ADMMAP.
*****************************************************************
* MSPUT AND MSGET OPTIONS.                                      *
*****************************************************************
         10 WRITE-OPERATION    PIC 9(8) COMP VALUE IS 0.
         10 REWRITE-OPERATION  PIC 9(8) COMP VALUE IS 1.
         10 REJECT-OPERATION   PIC 9(8) COMP VALUE IS 2.
         10 HILIGHT            PIC 9(8) COMP VALUE IS 3.
         10 COLOR              PIC 9(8) COMP VALUE IS 4.
         10 PS                 PIC 9(8) COMP VALUE IS 5.
*****************************************************************
* DATA AND ATTRIBUTE SELECTOR VALUES                            *
*****************************************************************
         10 IGNORE             PIC X VALUE IS SPACE.
         10 EXPLICIT           PIC X VALUE IS "1".
         10 SELECTED           PIC X VALUE IS "1".
         10 MAP-DEFINED        PIC X VALUE IS "2".
         10 OLD-VALUE          PIC X VALUE IS "3".
*****************************************************************
* CURSOR SELECTOR VALUES                                        *
*****************************************************************
         10 CURSED             PIC X VALUE IS "1".
*****************************************************************
* BASE (3270) ATTRIBUTE VALUES                                  *
*****************************************************************
* UNPROTECTED,NO MDT BIT.
         10 DEFAULT            PIC X VALUE IS " ".
         10 DETECTABLE         PIC X VALUE IS "D".
         10 BRIGHT             PIC X VALUE IS "H".
         10 DARK               PIC X VALUE IS "<".
         10 NUMERIC-UNPROT     PIC X VALUE IS "&".
         10 NUMERIC-DETECTABLE PIC X VALUE IS "M".
         10 NUMERIC-BRIGHT     PIC X VALUE IS "Q".
         10 NUMERIC-DARK       PIC X VALUE IS "*".

* PROTECTED,NO MDT BIT.
         10 PROTECT            PIC X VALUE IS "-".
         10 PROTECT-DETECTABLE PIC X VALUE IS "U".
         10 PROTECT-BRIGHT     PIC X VALUE IS "Y".
         10 PROTECT-DARK       PIC X VALUE IS "%".
         10 AUTOSKIP           PIC X VALUE IS "0".
         10 AUTOSKIP-DETECTABLE PIC X VALUE IS "4".
         10 AUTOSKIP-BRIGHT    PIC X VALUE IS "8".
         10 AUTOSKIP-DARK      PIC X VALUE IS "@".

* UNPROTECTED,MDT BIT.
         10 MDT                PIC X VALUE IS "A".
         10 DETECTABLE-MDT     PIC X VALUE IS "E".
         10 BRIGHT-MDT         PIC X VALUE IS "I".
         10 DARK-MDT           PIC X VALUE IS "(".
         10 NUMERIC-MDT        PIC X VALUE IS "J".
         10 NUMERIC-DETECTABLE-MDT PIC X VALUE IS "N".
         10 NUMERIC-BRIGHT-MDT PIC X VALUE IS "R".
         10 NUMERIC-DARK-MDT   PIC X VALUE IS ")".
```

```
* PROTECTED,MDT BIT.
         10 PROTECT-MDT          PIC X VALUE IS "/".
         10 PROTECT-DETECTABLE-MDT PIC X VALUE IS "V".
         10 PROTECT-BRIGHT-MDT PIC X VALUE IS "Z".
         10 PROTECT-DARK-MDT    PIC X VALUE IS "_".
         10 AUTOSKIP-MDT         PIC X VALUE IS "1̄".
         10 AUTOSKIP-DETECTABLE-MDT PIC X VALUE IS "5".
         10 AUTOSKIP-BRIGHT-MDT PIC X VALUE IS "9".
         10 AUTOSKIP-DARK-MDT   PIC X VALUE IS "'".
****************************************************************
* HIGHLIGHTING ATTRIBUTE VALUES                                *
****************************************************************
         10 NO-HIGHLIGHT         PIC X VALUE IS LOW-VALUE.
         10 BLINK                PIC X VALUE IS "1".
         10 REVERSE-VIDEO        PIC X VALUE IS "2".
         10 UNDERSCORE           PIC X VALUE IS "4".
****************************************************************
* COLOR ATTRIBUTE VALUES                                       *
****************************************************************
         10 MONOCHROME           PIC X VALUE IS LOW-VALUE.
         10 BLUE                 PIC X VALUE IS "1".
         10 RED                  PIC X VALUE IS "2".
         10 MAGENTA              PIC X VALUE IS "3".
         10 PINK                 PIC X VALUE IS "3".
         10 GREEN                PIC X VALUE IS "4".
         10 TURQUOISE            PIC X VALUE IS "5".
         10 CYAN                 PIC X VALUE IS "5".
         10 YELLOW               PIC X VALUE IS "6".
         10 WHITE                PIC X VALUE IS "7".
****************************************************************
* VALIDATION ATTRIBUTE VALUES                                  *
* (THESE ARE UNPRINTABLE CHARACTERS AND MUST BE INITIALIZED    *
* BY REDEFINING STORAGE).                                      *
****************************************************************
* NO VALIDATION:
         10 NOVALIDATN-BIN       PIC 9999 COMP VALUE IS 0.
         10 FILLER               REDEFINES NOVALIDATN-BIN.
           12 FILLER             PIC X.
           12 NO-VALIDATION      PIC X.

* TRIGGER
         10 TRIGGER-BIN          PIC 9999 COMP VALUE IS 1.
         10 FILLER               REDEFINES TRIGGER-BIN.
           12 FILLER             PIC X.
           12 TRIGGER            PIC X.

* MANDATORY ENTER:
         10 ENTER-BIN            PIC 9999 COMP VALUE IS 2.
         10 FILLER               REDEFINES ENTER-BIN.
           12 FILLER             PIC X.
           12 MANDATORY-ENTER    PIC X.

* TRIGGER AND MANDATORY ENTER:
         10 TRIGGER-ENTER-BIN    PIC 9999 COMP VALUE IS 3.
         10 FILLER               REDEFINES TRIGGER-ENTER-BIN.
           12 FILLER             PIC X.
           12 TRIGGER-ENTER      PIC X.

* MANDATORY FILL:
         10 FILL-BIN             PIC 9999 COMP VALUE IS 4.
         10 FILLER               REDEFINES FILL-BIN.
           12 FILLER             PIC X.
           12 FILL               PIC X.

* TRIGGER AND MANDATORY FILL:
         10 TRIGGER-FILL-BIN     PIC 9999 COMP VALUE IS 5.
         10 FILLER               REDEFINES TRIGGER-FILL-BIN.
           12 FILLER             PIC X.
           12 TRIGGER-FILL       PIC X.
```

```
* TRIGGER, ENTER AND FILL:
         10 TRIGGER-ENTER-FILL-BIN  PIC 9999 COMP VALUE IS 7.
         10 FILLER              REDEFINES TRIGGER-ENTER-FILL-BIN.
           12 FILLER            PIC X.
           12 TRIGGER-ENTER-FILL PIC X.
```

PL/I mapping constants table — ADMUPIMC

```
/********************************************************************/
/* TABLE NAME: ADMUPIMC                                             */
/*                                                                  */
/* DESCRIPTIVE NAME:  GDDM PL/I DECLARATIONS OF MAPPING CONSTANTS   */
/*                                                                  */
/*                5668-801                                          */
/*                (C) COPYRIGHT IBM CORP. 1979, 1986.               */
/*                LICENSED MATERIALS - PROPERTY OF IBM              */
/*                                                                  */
/* FUNCTION:                                                        */
/*                                                                  */
/*   THIS TABLE PROVIDES PL/I DECLARATION STATEMENTS FOR            */
/*   CONSTANTS USED FOR SETTING/TESTING ADJUNCT FIELDS IN A         */
/*   MAP APPLICATION DATA STRUCTURE.  IT ALSO CONTAINS DECLARATIONS */
/*   OF CONSTANTS USED FOR MSPUT/MSGET OPTIONS.                     */
/*                                                                  */
/*   THE DATA TYPE  USED FOR HARDWARE AND SOFTWARE ADJUNCTS IN THE  */
/*   GDDM-IMD-GENERATED APPLICATION DATA STRUCTURE FOR PLI IS       */
/*   CHARACTER.  SOME OF THE CODE-POINTS ARE UNPRINTABLE CHARACTERS */
/*   THAT IS, IN THE RANGE HEX'00' TO HEX'3F'.  FOR THESE CASES THE */
/*   CODE-POINTS DECLARED HERE ARE BIT(8), AND THE INTENTION IS     */
/*   THAT THE PROGRAM SHOULD USE THESE WITH UNSPEC.                 */
/*   FOR EXAMPLE, TO SET THE VALIDATION CODE TO TRIGGER FOR A       */
/*   FIELD IN THE ADS CALLED FIELDNAME, USE                         */
/*                                                                  */
/*           FIELDNAME_VAL_SEL = SELECTED                           */
/*           UNSPEC(FIELDNAME_VAL) = TRIGGER                        */
/*                                                                  */
/********************************************************************/
  DECLARE
    1 ADMMAP STATIC,
      /**************************************************************/
      /* MSPUT AND MSGET OPTIONS.                                   */
      /**************************************************************/
      2 WRITE                 FIXED BIN(31) INIT(0),
      2 REWRITE               FIXED BIN(31) INIT(1),
      2 REJECT                FIXED BIN(31) INIT(2),
      2 HILIGHT               FIXED BIN(31) INIT(3),
      2 COLOR                 FIXED BIN(31) INIT(4),
      2 PS                    FIXED BIN(31) INIT(5),
      /**************************************************************/
      /* DATA AND ATTRIBUTE SELECTOR VALUES.                        */
      /**************************************************************/
      2 IGNORE                CHAR(1) INIT(' '),
      2 EXPLICIT              CHAR(1) INIT('1'),
      2 SELECTED              CHAR(1) INIT('1'),
      2 MAP_DEFINED           CHAR(1) INIT('2'),
      2 OLD_VALUE             CHAR(1) INIT('3'),
      /**************************************************************/
      /* CURSOR SELECTOR VALUES.                                    */
      /**************************************************************/
      2 CURSED                CHAR(1) INIT('1'),
      /**************************************************************/
      /* BASE (3270) FIELD ATTRIBUTE VALUES.                        */
      /*                                                            */
      /* UNPROTECTED, NOT MODIFIED.                                 */
      /**************************************************************/
      2 DEFAULT               CHAR(1) INIT(' '),
      2 DETECT                CHAR(1) INIT('D'),
      2 BRIGHT                CHAR(1) INIT('H'),
      2 DARK                  CHAR(1) INIT('<'),
      2 NUMERIC               CHAR(1) INIT('&'),
      2 NUMERIC_DETECT        CHAR(1) INIT('M'),
      2 NUMERIC_BRIGHT        CHAR(1) INIT('Q'),
      2 NUMERIC_DARK          CHAR(1) INIT('*'),
      /**************************************************************/
      /* PROTECTED, NOT MODIFIED.                                   */
      /**************************************************************/
      2 PROTECT               CHAR(1) INIT('-'),
2 PROTECT_DETECT      CHAR(1) INIT('U'),
2 PROTECT_BRIGHT      CHAR(1) INIT('Y'),
2 PROTECT_DARK        CHAR(1) INIT('%'),
2 AUTOSKIP            CHAR(1) INIT('0'),
2 AUTOSKIP_DETECT     CHAR(1) INIT('4'),
2 AUTOSKIP_BRIGHT     CHAR(1) INIT('8'),
2 AUTOSKIP_DARK       CHAR(1) INIT('@'),
/***************************************************************/
/* UNPROTECTED, MODIFIED.                                      */
/***************************************************************/
2 MDT                 CHAR(1) INIT('A'),
2 DETECT_MDT          CHAR(1) INIT('E'),
2 BRIGHT_MDT          CHAR(1) INIT('I'),
2 DARK_MDT            CHAR(1) INIT('('),
2 NUMERIC_MDT         CHAR(1) INIT('J'),
2 NUMERIC_DETECT_MDT  CHAR(1) INIT('N'),
2 NUMERIC_BRIGHT_MDT  CHAR(1) INIT('R'),
2 NUMERIC_DARK_MDT    CHAR(1) INIT(')'),
/***************************************************************/
/* PROTECTED, MODIFIED.                                        */
/***************************************************************/
2 PROTECT_MDT         CHAR(1) INIT('/'),
2 PROTECT_DETECT_MDT  CHAR(1) INIT('V'),
2 PROTECT_BRIGHT_MDT  CHAR(1) INIT('Z'),
2 PROTECT_DARK_MDT    CHAR(1) INIT('_'),
2 AUTOSKIP_MDT        CHAR(1) INIT('1'),
2 AUTOSKIP_DETECT_MDT CHAR(1) INIT('5'),
2 AUTOSKIP_BRIGHT_MDT CHAR(1) INIT('9'),
2 AUTOSKIP_DARK_MDT   CHAR(1) INIT(''''),
/***************************************************************/
/* VALIDATION FIELD ATTRIBUTE VALUES.                          */
/***************************************************************/
2 NO_VALIDATION       BIT(8) ALIGNED INIT('00000000'B),
2 TRIGGER             BIT(8) ALIGNED INIT('00000001'B),
2 ENTER               BIT(8) ALIGNED INIT('00000010'B),
2 TRIGGER_ENTER       BIT(8) ALIGNED INIT('00000011'B),
2 FILL                BIT(8) ALIGNED INIT('00000100'B),
2 TRIGGER_FILL        BIT(8) ALIGNED INIT('00000101'B),
2 ENTER_FILL          BIT(8) ALIGNED INIT('00000110'B),
2 TRIGGER_ENTER_FILL  BIT(8) ALIGNED INIT('00000111'B),
/***************************************************************/
/* HIGHLIGHT FIELD AND CHARACTER ATTRIBUTE VALUES.             */
/***************************************************************/
2 NO_HIGHLIGHT        CHAR(1) INIT(' '),
2 BLINK               CHAR(1) INIT('1'),
2 REVERSE_VIDEO       CHAR(1) INIT('2'),
2 UNDERSCORE          CHAR(1) INIT('4'),
/***************************************************************/
/* COLOR FIELD AND CHARACTER ATTRIBUTE VALUES.                 */
/***************************************************************/
2 MONOCHROME          CHAR(1) INIT(' '),
2 BLUE                CHAR(1) INIT('1'),
2 RED                 CHAR(1) INIT('2'),
2 PINK                CHAR(1) INIT('3'),
2 MAGENTA             CHAR(1) INIT('3'),
2 GREEN               CHAR(1) INIT('4'),
2 TURQUOISE           CHAR(1) INIT('5'),
2 CYAN                CHAR(1) INIT('5'),
2 YELLOW              CHAR(1) INIT('6'),
2 WHITE               CHAR(1) INIT('7');
```

Chapter 12. Special-purpose programming in GDDM

The System Programmer Interface (SPI) is provided for programmers who want to use GDDM as the basis for a graphics system of their own. It enables GDDM functions to be written in a coded form, it gives greater control over the subsystem environment, and it allows greater programming flexibility within the subsystem environment.

This chapter describes:

- "Using the system programmer interface," below,

and

- "Specifying user exits" on page 104.

Using the system programmer interface

The system programmer interface is a special interface available to "system programming" types of applications. It is available only in reentrant form, and shares many features with the application-programmer reentrant interface. The reentrant interfaces are described in the *GDDM Base Programming Reference, Volume 1*.

In the simplest case, the system programmer interface merely provides a means of accessing a GDDM function by a function code (the Request Control Parameter, RCP) rather than by selecting an entry point. Assembler-language macros defining mnemonics for these function codes are provided.

Each call takes the form:

```
CALL ADMASP (aab,rcp,component parameters,...)
```

where ADMASP is the defined system programmer interface entry point. ADMASP is a single entry point resolved by the GDDM interface modules that are link-edited with the application.

Note: The sample PL/I declarations do not include this entry point, because it can only be called using the system programmer interface. The PL/I application programmer using this call must, therefore, supply an entry-point declaration for the system programmer interface, as described in the *GDDM Base Programming Reference, Volume 1*. For example:

```
DECLARE ADMASP EXTERNAL ENTRY OPTIONS (ASM,INTER);
```

Parameters

aab (*specified by user*) (*8-byte control block*)
An Application Anchor Block, as described in the *GDDM Base Programming Reference, Volume 1*.

rcp (*specified by user*) (*full-word integer*)
The Request Control Parameter, a 4-byte, full-word-aligned function code defining the GDDM function to be called. The GDDM RCP code is given, for each GDDM call listed and described in the *GDDM Base Programming Reference, Volume 1*, and for each GDDM-PGF call in the *GDDM-PGF Programming Reference* manual, in both hexadecimal and decimal format. Also, Appendix J, "Request control parameter codes" on page 231 contains a table defining the RCP codes for all GDDM and GDDM-PGF functions.

component parameters
The parameters for the function specified in the RCP. These are as described for the specific function being called.

Calls to the system programmer and reentrant interfaces can be mixed, provided that the same application anchor block is passed on each call.

Initialization

This interface provides an alternative initialization function (known as SPINIT) that allows control of environmental aspects. SPINIT is an alternative to FSINIT and, if used, must be the first GDDM statement to be run.

Note that your program would not use an explicit call to an entry point called SPINIT. Instead, like all other system programmer interface calls, you would code a call to ADMASP. The function is described for consistency as a SPINIT call, as it behaves like the other GDDM calls, but it can only be specified through the system programmer interface. The GDDM Assembler language tables ADMURCPB and ADMURCPO (see Appendix J, "Request control parameter codes" on page 231) include the mnemonic QQSPINIT.

SPINIT	**(spib-block)**
APL Code	115
GDDM RCP code	X'00050000' (327680)

Initializes GDDM processing, with the special processing requirements specified in the spib-block parameter.

Parameters

spib-block (specified by user) (32-byte character string)
A table giving control information. The contents of this table are processed by GDDM during initialization. Subsequent changes to the contents do not affect GDDM processing. The storage containing the table can be released after initialization has been completed.

Note: Since Version 1 Release 4, GDDM supports an abbreviated format of the SPIB. This is described below. A number of the functions that were previously available in the SPIB are now available through other GDDM calls, which can be issued immediately after the SPINIT call. For example, the functions of the SPIBOPNF, SPIBPA2F, SPIBXFBF, SPIBXFBL, and SPIBXFBP fields can now be specified as DSOPEN processing options; the functions of many other fields can be specified as input to the SPIB by means of items in a user default specification list (see Chapter 1, "Customizing your program and its environment" on page 1 for details).

The previous format of the SPIB is retained for reasons of compatibility; it does not contain or provide access to new function provided since GDDM Version 1 Release 4. It is described in the edition of the *GDDM Base Programming Reference* manual for the Release of GDDM for which your program was written.

Principal Errors

None

Format of the system programmer interface block

The labels are defined here in more detail:

SPIBLENG
Specifies the length of the SPIB. Must be in the range 16 through 32, which identifies this as a GDDM Version 1 Release 4 SPIB. The fields after offset X'10' can be omitted (and thus allowed to default) by specifying the minimum value of 16.

SPIBUDSL
Specifies the length (in bytes) of an encoded user default specification list (UDSL). Must be set to 0 if no UDSL is to be passed.

SPIBUDSP
Specifies the address of an encoded user default specification list (UDSL). Must be set to 0 if no UDSL is to be passed.

SPIBGSXP (TSO and VM/CMS only)
Specifies (if present and if not zero) the address of an application-defined storage exit to be called for GET STORAGE requests.

SPIBGSXK (TSO and VM/CMS only)
Specifies (if present) a user-defined parameter that GDDM is to pass when calling a GET STORAGE exit.

SPIBFSXP (TSO and VM/CMS only)
Specifies (if present and if not zero) the address of an application-defined storage exit to be called for FREE STORAGE requests.

SPIBFSXK (TSO and VM/CMS only)
Specifies (if present) a user-defined parameter that GDDM is to pass when calling a FREE STORAGE exit.

The interface specifications for GDDM storage exits are described under "Storage exit routines – interface specifications" on page 108.

Specifying user exits

User exits allow a system program to trap specific events whenever an application program uses a GDDM or system resource. Such events include task switching in TSO, intercepting some or all GDDM calls, and so on.

A limited number of user exits can be specified using User Default Specifications (UDSs). UDSs are described in Chapter 1, "Customizing your program and its environment" on page 1. The user exits are:

- A Task Switch exit
- A Call Intercept exit
- A Coordination exit.

This section describes how you specify user exits, the conventions that your exits must follow, and the function of each type of exit.

It also describes the storage exit routines that can be defined by using the System Programmer Interface Block (SPIB) in the SPINIT call. For details of the SPIB, see "Initialization" on page 103.

Table 15 on page 105 shows the defaults that you can specify for GDDM exits using the SPINIT call. The figure also describes the corresponding user default specifications (in source and encoded format). These UDSs must be passed to GDDM using the SPINIT call in the form of an encoded-UDS list. The last column shows where the UDS can be specified, as follows:

M	in the External Defaults Module,
F	in the External Defaults File,
S	in the SPINIT call,
C	in the ESEUDS and ESSUDS calls.

Table 14. SPIB format

Offset (hex)	Length (bytes)	Label	Usage
0	4	SPIBHEAD	Spare. Reserved for the application program to use as an eye-catcher.
4	4	SPIBLENG	Length of SPIB.
8	4	SPIBUDSL	Length of user default specification list.
C	4	SPIBUDSP	Address of user default specification list.
10	4	SPIBGSXP	Address of application-defined GET STORAGE exit.
14	4	SPIBGSXK	User-defined parameter to be passed to the application program's GET STORAGE exit.
18	4	SPIBFSXP	Address of application-defined FREE STORAGE exit.
1C	4	SPIBFSXK	User-defined parameter to be passed to the application program's FREE STORAGE exit.

Table 15. GDDM exits — options

Meaning of default	Source syntax of the ADMMEXIT macro options	Encoded values - list of full-words	Valid in: M F S C
Call intercept user exit address	CALLINT=(addr)	3,3005,A(CI-UX)	N N Y N
Call intercept user exit token value	CALLINT=(,token)	3,3006,CI-token	N N Y N
Default user exit address	DEFAULT=(addr)	3,3001,A(DFT-UX)	N N Y N
Default user exit token value	DEFAULT=(,token)	3,3002,DFT-token	N N Y N
Task switch user exit address (TSO only)	TASKSWI=(addr)	3,3003,A(TSW-UX)	N N Y N
Task switch user exit token value (TSO only)	TASKSWI=(,token)	3,3004,TSW-token	N N Y N
Note: In the source-format forms, corresponding pairs can be combined in this way: DEFAULT=(address,token).			

Exit values

The descriptions of these options are:

CALLINT=(address,token)
address gives the full-word address of the Call Intercept exit.

token provides four bytes of data that are passed from the application program to the exit.

DEFAULT=(address,token)
address gives a full-word address for all user exits. Specifying an address in this option is equivalent to specifying it for each user exit explicitly.

token provides four bytes of data that are passed from the application program to any exit. Specifying a token in this option is equivalent to specifying it for each user exit explicitly.

TASKSWI=(address,token)
address gives the full-word address of the Task Switch exit.

token provides four bytes of data that are passed from the application program to the exit.

GDDM user-exit conventions

Unless otherwise noted, user exits defined by means of UDSs must conform to these rules:

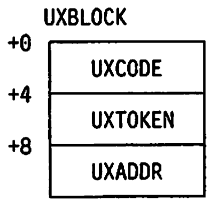
- The contents of the registers on entry to the exit are:

```
R13 -> A 72-byte save area
R14 -> The return address
R15 -> The entry point of the exit
R1  -> The parameter address list, in standard
       variable-list format:
       ADDR1 -> AAB     (Char(8))
       ADDR2 -> UXBLOCK ((3) Fixed(31))
       .
       .
       additional parameters as defined for the
       specific exit
```

AAB
The application's AAB (application anchor block) (or in the case of the coordination exit, the GDDM-provided dummy AAB if the application is using the nonreentrant interface).

The exit must not use the AAB to issue a GDDM call. That is to say, the GDDM instance that caused the exit must not be entered recursively.

UXBLOCK
A user-exit control block of this format:



The contents of UXBLOCK are:

UXCODE The full-word user-exit code. This code is the same as the UDS-code used to define the user exit address. The exit must **not** change this parameter.

UXTOKEN The full-word user-exit token. This field is initialized to 0. An explicit value for this token can be specified when the exit is specified. The exit or application program can change this parameter; in which case, subsequent calls to the exit are passed in the changed parameter.

UXADDR The full-word user-exit address. On entry to an exit, this parameter has the same value as R15 (the address of the exit entry point). The exit can change this parameter; in which case, subsequent calls to the exit are to the new address. If the address is set to 0, GDDM stops using the exit for as long as the address remains 0. If the address is subsequently reset to nonzero (by the application program or by another exit), GDDM resumes invocation of the exit.

- The parameter address list is in variable parm-list format (that is, with the high-order bit of the last address word set to "1"), and GDDM may pass parameters in addition to those defined below. Therefore, the exit must **not** rely on the high-order bit of a specific parameter address word always being set to "1."
- Unless otherwise noted, the exit must not modify any parameter passed to it. (The only exception is the UXBLOCK parameter.)

- On return, the exit must set R15 to one of the specified completion codes.

  If any other value is returned, the results are undefined (nonzero values may be diagnosed, ignored, or abended).
- It is recommended that you make the exit **reentrant** and **read-only**. Otherwise, careful thought must be given as to how the operation of GDDM and its calling application(s) is affected.
- The exit must conform to standard System/370 calling conventions (including the use of save areas and restoring registers).
- Under MVS/XA, the exit must be AMODE(ANY); that is, it must be prepared to accept control in 24-bit or 31-bit mode, and must return control in the same mode. If called in 31-bit mode, all addresses (including R13) must be treated as 31-bit addresses and may be greater than 16 megabytes.

  Under MVS/XA, a 24-bit mode application program must ensure that the top byte of an initial value for a user-exit token is cleared to zero if it intends that this token is to be interpreted as an address. GDDM considers this token to be a FIXED(31) variable, and does **not** clear the top byte of the token before invoking the exit.

The task switch exit

A Task Switch exit can be defined in an ADMMEXIT UDS. This exit is valid under TSO only. If it is specified in other environments, the results are undefined.

Function: By providing a Task Switch exit, an TSO tasking application program can call GDDM both from its main task and from any of a number of subtasks. The Task Switch exit should be coded to switch to a standard task (typically, the main task) under which specific subsystem-dependent task-sensitive functions can be performed.

If enabled, the Task Switch exit is invoked before GDDM performs selected task-sensitive functions. The Task Switch exit has passed to it the address of a GDDM subroutine to be called after switching tasks, plus the parameters to be passed to the routine.

The Task Switch exit is returned to when the GDDM subroutine has performed the task-sensitive functions. The Task Switch exit should then switch tasks back before returning to GDDM.

The system-dependent functions that are "task protected" in this manner are:

- Explicit GETMAIN and FREEMAIN requests. (Indirect requests by means of storage exits or other system-dependent functions are **not** "task protected.")
- DASD OPEN and CLOSE requests. (READ, WRITE, PUT, and GET requests are **not** "task protected.")
- Explicit LOAD and DELETE requests.

Exceptionally, some of the GETMAIN, FREEMAIN, LOAD, and DELETE requests that are issued by GDDM routines at initialization and termination are not "task protected." These requests should be separately "task protected" by the application program, by ensuring that the GDDM FSINIT (or SPINIT) and FSTERM calls are always issued from the standard task.

A Task Switch exit should be prepared to be invoked in a recursive manner in some circumstances. For example:

```
GDDM invokes the Task Switch exit before OPEN.
--> The Task Switch exit calls a GDDM subroutine.
------> The OPEN macro is called, resulting in an
        OPEN error.
----------> The DCB ABEND exit receives control.
----------> Diagnostic processing is initiated.
----------> GDDM invokes the Task Switch exit
            before a LOAD for diagnostic routines.
--------------> The Task Switch exit calls a GDDM
                subroutine.
-----------------> The LOAD macro is called for
                   diagnostic routines.
-----------------> The subroutine returns to the
                   Task Switch exit.
--------------> The Task Switch exit returns to
                GDDM after the LOAD.
----------> Diagnostic processing completes.
----------> The DCB ABEND exit returns to NSI
            after the OPEN.
------> The OPEN macro completes.
------> The subroutine returns to the Task Switch
        exit.
--> The Task Switch exit returns to GDDM after
    the OPEN.
```

However, a Task Switch exit can prevent such recursion by disabling itself on entry, by setting the UXADDR field in the UXBLOCK parameter to 0. GDDM still ensures a return through the Task Switch exit, which should then reset the UXADDR field to the address of its entry point, before returning to GDDM.

How to specify a task switch exit: A Task Switch exit is specified as follows:

```
ADMMEXIT TASKSWI=([address][,token])
```

Parameters: The parameters for Task Switch exits are as follows:

```
R13 -> A 72-byte save area
R14 -> The return address
R15 -> The entry point of the exit
R1  -> The parameter address list, in standard
       variable parm-list format:
             ADDR1 -> AAB     (Char(8))
             ADDR2 -> UXBLOCK ((3) Fixed(31))
             ADDR3 -> SUBADDR (Ptr(31))
             ADDR4 -> SUBPARM (Format reserved to
                              GDDM)
```

Parameters AAB and UXBLOCK are described under "GDDM user-exit conventions" on page 105. Additional parameters are as follows:

SUBADDR The address of the GDDM subroutine to be called after switching tasks.

The GDDM subroutine must be called according to full System/370 calling conventions. Specifically, Register 13 on entry to the subroutine must locate a register save area, which must **not** be the same as that passed to the exit by GDDM. Also, Register 1 on entry to the subroutine must be the same as was passed to the exit by GDDM.

The GDDM subroutine saves and restores the exit's registers as normal, but does **not** necessarily conform to other System/370 calling conventions.

On return from the subroutine, the exit must return to GDDM according to full System/370 calling conventions. Specifically, the exit **must** reload Register 14 from GDDM's save area in order to return. The exit must **not** rely on the contents of GDDM's save area being the same as on entry (specifically, all saved registers, including Register 14, and the RSA forward chain, may have been modified by the GDDM subroutine).

SUBPARM Additional parameter(s) that may be supplied by GDDM, for the use of the GDDM subroutine.

The exit should not assume the existence of, nor try to examine, these parameters. The exit should call the GDDM subroutine with Register 1 locating the **same** parameter address-list as that passed to the exit by GDDM.

The exit must be AMODE(ANY); that is, it must be prepared to accept control in 24-bit or 31-bit mode, and must return control in the same mode. Also, it must call the GDDM subroutine in the same mode.

Feedback values: On return, the exit must set R15 as follows:

0	Successful completion.

The call intercept exit

A Call Intercept exit may be defined by using an ADMMEXIT UDS. This exit is valid in all environments.

Function: The Call Intercept exit provides a mechanism whereby a controlling process can monitor the calls issued by an application program. Other than for its specification by means of the SPIB, this exit is transparent to an application program at the API.

The Call Intercept exit is invoked from within GDDM, **before** each application-program call is processed (though after some housekeeping has been performed). Application-program calls that are grossly in error may be rejected without giving control to the exit.

The exit has passed to it the parameters provided by the application program. It cannot change the request or the parameters, but it can have some control over the subsequent execution, as described below.

The exit could operate in a pass-through mode, whereby it passes the specified requests through to a secondary instance of GDDM that had been separately initialized. In this mode, the exit could change or add more calls to the secondary instance of GDDM in response to a single call from the application program. However, in this mode the exit may have difficulty passing-back error diagnostics from the GDDM secondary instance.

How to specify a call intercept exit: The Call Intercept exit is specified as follows:

```
ADMMEXIT CALLINT=([address][,token])
```

Parameters: The parameters for the Call Intercept exit are as follows:

```
R13 -> A 72-byte save area
R14 -> The return address
R15 -> The entry point of the exit
R1  -> The parameter address list, in standard
       variable parm-list format:
         ADDR1 -> AAB     (Char(8))
         ADDR2 -> UXBLOCK ((3) Fixed(31))
         ADDR3 -> RCP     (Fixed(31))
         ADDR4 -> NPARMS  (Fixed(31))
         ADDR5 -> PLIST(NPARMS) (Array of Ptr(31))
```

Parameters AAB and UXBLOCK are described under "GDDM user-exit conventions" on page 105. Additional parameters are as follows:

RCP	The RCP code defining the call issued by the application program.
NPARMS	The number of functional parameters provided by the application program (excluding the AAB for RACI, and the AAB and RCP for SPI).
PLIST(NPARMS)	The addresses of the functional parameters provided by the application program. These addresses are **not** in variable parameter-list format. These addresses should be treated as read-only. ADDR5 is undefined (and hence PLIST is not addressable) if NPARMS = 0.

Feedback values: On return, the exit must set R15 as follows:

0	GDDM is to continue processing the call
8	GDDM is to ignore the call, with no message
12	GDDM is to reject the call, issuing the message:

```
ADM0056 E  REQUEST REJECTED BY USER EXIT.
           REASON n
```

If R15 = 12, the exit should set R0 as follows:

n	The reason-code to be inserted into message ADM0056.

Otherwise, R0 should be restored to its value on entry.

The coordination exit

A coordination exit can be defined by specifying the coordination exit address in the **array** parameter of the WSCRT call; for a description of this, see the *GDDM Base Programming Reference, Volume 1.*

Function: By providing a coordination exit when creating an operator window, a task manager allows the use of that window by independent applications running their own instances of GDDM.

How to specify a coordination exit: A coordination exit is specified as part of the WSCRT call. For details, see the description of the WSCRT call in the *GDDM Base Programming Reference, Volume 1.*

Parameters: The parameters for coordination exits are as follows:

```
R13 -> A 72-byte save area
R14 -> The return address
R15 -> The entry point of the exit
R1  -> The parameter address list, in standard
       variable parm-list format:
             ADDR1 -> AAB     (Char(8))
             ADDR2 -> UXBLOCK ((3) Fixed(31))
             ADDR3 -> DIRECTN (Fixed(31))
```

Parameters AAB and UXBLOCK are described under "GDDM user-exit conventions" on page 105. Additional parameters are as follows:

DIRECTN The direction in which the exit is to pass control. Possible values are:

- **0** Pass control from the sub-task to the main task
- **1** Pass control from the main task to the sub-task.

The exit may not change this parameter.

Feedback values: On return, the exit must set R15 as follows:

- **0** Successful completion
- **8** Sub-task terminated abnormally.

Storage exit routines – interface specifications

Storage exit routines can be defined using explicit fields in the System Programmer Interface Block (SPIB) passed as a parameter to GDDM in the SPINIT call.

The following section references fields defined in the Version 1 Release 4 format of the SPIB, but equivalent fields exist in the pre-Version 1 Release 4 format. For details, see "Initialization" on page 103.

Under VM/CMS and TSO, GDDM calls application exit routines, identified by fields SPIBGSXP and SPIBFSXP (if defined and nonzero), to GET and FREE storage. The interface to these storage exits is as follows:

Register 0 contains the number of bytes of storage requested (GET) or to be released (FREE). The high-order bit of this register is set to indicate a conditional request. This value is passed to the storage exits for both GET and FREE.

Register 1 contains the address of the block of storage. This address is returned by the application exit on GET and passed to the application exit on FREE.

Register 14 contains the GDDM return address.

Register 15 contains the user-defined parameter specified in either field SPIBGSXK (GET) or field SPIBFSXK (FREE). This is passed by GDDM to the appropriate application exit on each call. Before returning to GDDM, the application exit should set a return code in register 15: 0 indicating that the request was successful, and, for conditional requests only, 4 indicating that the request was unsuccessful.

All other registers must be preserved across the call.

Application storage exits must operate without corrupting any of the registers on entry other than as described above. On entry to the exit routines, register 13 does not locate a register save area. If necessary, the exits should provide for their own save area, possibly by "anchoring" a user area by means of the SPIBGSXK or SPIBFSXK, or both, fields passed in register 15.

Application storage exits must not assume that their entry point is located by register 15 on entry. Register 15 is set as described above.

The application GET storage exit must return storage that is double-word aligned.

GDDM abnormally ends on receiving a return code other than as described above.

GDDM requests for blocks of local, last-in-first-out, or instance storage are restricted to a maximum length of 32K bytes. When storage and exit routines are defined (that is, "active"), this restriction also applies to extended storage requests. GDDM never releases "merged" or "split" blocks; storage is always released in blocks as acquired from the application GET exit routine.

Under MVS/XA, the top bit of the specified storage exit address is taken to identify the AMODE of the exit and causes the exit to be called accordingly (that is, a top bit of '1'B causes the corresponding exit to be called in 31-bit addressing mode).

Chapter 13. GDDM high-performance alphanumerics

High-performance alphanumerics (HPA) is another way of doing alphanumerics in GDDM, and is intended for complex applications which require minimum instruction path length within GDDM.

The application program may not mix mapped and procedural alphanumeric field definitions with HPA field definitions on the same GDDM page.

The style of application programming interface used by HPA differs from that used by other parts of GDDM, such as procedural alphanumerics. When using procedural alphanumerics, application programs use many API calls to describe the data to GDDM for output, and also to determine the data input by the device operator. In contrast, the HPA application builds a data structure to describe all the data, and passes that to GDDM for output. Also, the data input by the device operator is returned to the HPA application in the same data structure. Changes to the data are indicated through status indicators which are part of the structure.

HPA data structure

The data structure consists of three distinct objects. These are:

The field list
The data buffer
The bundle list.

The field list

The **field list** groups together all information about the layout of alphanumeric data on one GDDM page. New fields can be added to an existing GDDM page, or old ones deleted, by modifying the field list. To give additional flexibility, there may be more than one field list in any GDDM page, so that if an existing field list is used up, further field definitions can be added by creating a new one.

A field list consists of a header followed by field definitions.

The header contains:

The status of the field list
The number of field definitions in the list
The size of the field definitions
The cursor position on the page.

Each field definition contains:

The status of the field definition
The size and position of the field on the GDDM page
A reference to the field attribute bundle definition in the bundle list
A reference to the character data
Optional length of character data
Optional references to character attributes.

The field list is represented as a rectangular array of half-word integers, in which the first row is the header and the following rows contain field definitions. It can be declared as a structure, or as a two-dimensional array stored in row-major order. Programming languages which use column-major ordering of two dimensional arrays will have to exchange rows and columns in the description which follows. Below is a sample PL/I declaration for a field list, where "depth" and "width" are the array dimensions used in the API call APDEF:

```
DCL FIELD_LIST(depth,width) FIXED BIN(15);
```

The numbers beside each component description below are the indices of each item in the row. See Figure 5 on page 110.

The field list header row

1 – List Status
The status of the field list.

Values that can be assigned to list status are the same as field status; in fact, list status must always be equal to the value obtained by ORing together the values of all the field statuses in the field list. For example, if any field has the indicator set to indicate that the field is to be "output" because the character data has been changed by the application, the corresponding indicator in list status must also be set. This means that whenever the application changes a field status indicator, it must ensure that the list status indicator is correct. Whenever GDDM changes a field status indicator it will also do this.

2 – Used depth
The number of rows in the field list used by GDDM.

This value must be in the range 1 through list depth. It may be changed by the application in order to add new fields or to remove deleted fields from the list.

Note: If this number is increased to add new fields to the list, the create indicator must be set in the new field-definition status elements. Also, if deleted fields are removed from the list, the deletions must first have been processed by GDDM, which sets the status element in the field definitions to zero.

3 – Used width
The number of elements in the header and each field definition used by GDDM.

This value must be less than or equal to the list width, and must be in the range 6 through 10. If the value is less than the list width, then any extra elements in the header and each field definition are ignored by GDDM, and may be used by the application to record its own data. It may be changed by the application in order to extend or reduce the field definitions. An example of this might be increasing the used width to 9 in order to specify character color. If the used width is changed, the output indicator must be set in the field definition status elements of all the field definitions altered by this change.

If this value is less than 10, then the omitted parts of the field definition are described as being "not present," and assume default values.

Note: Even though GDDM may not use as many elements in the header as in the field definitions, only those elements beyond the used width may be used for application data. The rest must be zero.

4 – Cursor row
This is the row position of the alphanumeric cursor on the GDDM page.

When used, it must be in the range 1 through page depth, otherwise it must be zero. If the field list is designated as the one used for cursor positioning, then the cursor row and cursor column are used to position the alphanumeric cursor on output, and also to return its position on input. This designation is made by setting the mode parameter of the APDEF or APMOD call.

This cursor position overrides any cursor position specified by calling ASFCUR. During I/O, if the cursor position specified lies outside the page window, then the cursor is placed at the closest position within the page window.

5 – Cursor column
This is the column position of the alphanumeric cursor on the GDDM page.

When used it must be in the range 1 through page width, otherwise it must be zero.

The field definition row

1 – Field Status
The status of the field definition. The list of values below shows both numerical value and corresponding bit position of the indicator. If your use of HPA requires complex testing and setting of these status indicators then you may wish to declare the status element as a bit string.

Values that can be assigned to the field status are:

1 — Bit 15 — Process
If this indicator is not set, none of the other indicators in the field status element may be set.

Only those field definitions that have this indicator set are processed. This allows space for future field definitions to be reserved in the field list, in which case the application program must set both this indicator and the create indicator before the first use of the field. If a field has been indicated to be deleted, GDDM sets the field status element to zero on the next I/O to the primary device involving the GDDM page.

Note: An I/O involving the page is any I/O operation, ASREAD, FSFRCE, and so on, for the primary device to which the page belongs during which the page is the current one for its partition, and the partition set is the current one for the device.

2 — Bit 14 — Create
Indicates a new field to be created.

If it is set, GDDM resets it on the next I/O to the primary device involving the GDDM page. When a field list is first defined to GDDM all its fields are assumed to be new, so this indicator need not be set.

4 — Bit 13 — Delete
Indicates a field to be deleted.

When the application sets this indicator, it informs GDDM that the field is to be deleted. GDDM resets the entire status element, including the Process indicator, on the next I/O to the primary device involving the GDDM page. The field definition may not be reused to define another field until after GDDM has reset this indicator.

8 — Bit 12 — Output
Indicates a field to be output.

It must be set by the application whenever it changes one of the following:

Character data
Character attributes
Character index
Color index
Highlight index
Symbol-set index
Actual-length
Bundle-row.

This indicator is reset by GDDM on the next I/O to the primary device involving the GDDM page.

Column (width)

Row (depth)	1	2	3	4	5	6	7	8	9	10 ...
1	List status	Used-depth	Used-width	Cursor row	Cursor column					
2	Field status	Field row	Field column	Field width	Bundle row	Char index	Actual length	Color index	Highlt index	SS index
3	Field status	Field row	Field column	Field width	Bundle row	Char index	Actual length	Color index	Highlt index	SS index
4	Field status	Field row	Field column	Field width	Bundle row	Char index	Actual length	Color index	Highlt index	SS index
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Figure 5. Field list array

Notes:

1. This indicator is set by GDDM if the device operator updated the field. This causes the field to be output on the next I/O to ensure that any input data editing is reflected back on the device.
2. This indicator should not be set to indicate changes in the bundle definition, it only indicates changes in the field definition.

16 — Bit 11 — Input
Indicates a field has been input.

This indicator is set by GDDM, during input involving the GDDM page, to indicate changes to character data and possibly character attributes, made by the device operator. It should be reset by the application once the changes have been processed.

If more than one status indicator is required, the element must be set to the sum of the numbers corresponding to the indicators required.

2 – Row
This is the row for the top left-hand corner of the field within the GDDM page.

Rows are numbered from top to bottom of the page, starting with 1. This is the position of the field contents, not the field attribute. Once the field has been defined the application may not change the field row until the field has been deleted. For best performance it is recommended that fields are defined in order of their positions on the page.

3 – Column
This is the column for the top left-hand corner of the field within the GDDM page. Columns are numbered from left to right across the page, starting with 1. This is the position of the field contents, not the field attribute. Once the field has been defined, the application may not change the field column until the field has been deleted.

4 – Width
This is the number of columns that the field occupies.

The width may cause the field to extend beyond the right-hand side of the page, in which case it wraps to the left-hand side of the page on the next row. A field may not extend below the bottom of the page, neither may fields overlap. Once the field has been defined, the application may not change the field width until the field has been deleted.

Width also defines the data-area length. For mixed-without-position fields the data-area length is twice Width bytes, and for other fields the data-area length is Width bytes. The data-area length is the length of the data areas in the data buffer, where the data for the field is held. There may, optionally, be data areas for:

Character data
Character color attributes
Character highlight attributes
Character symbol-set attributes.

The data areas as defined by the character index and width, the color index and width, the highlight index and width, and the symbol-set index and width, must be contained totally within the data buffer.

5 – Bundle row
This is the row number in the bundle list of the field attribute bundle definition. It must be in the range 2 through the number of rows in the bundle list.

6 – Character Index
This is the index in the data buffer of the data area containing the characters that occupy the field. An index of 0 indicates that there are no character codes for the field. The character data area must be present if color, highlight, or symbol-set data areas are present. The character data area must also be present if the field is unprotected or has the MDT attribute.

Note: It is possible for more than one field to be associated with the same data area or overlapping data areas, within the data buffer. This does not cause any difficulty if all the fields are protected.

In the instance where one or more of the fields is unprotected, the application must set the output indicators of **all** the fields involved if the data area has been changed as a result of device operator input. If this is not done, the corresponding fields on the screen may not be updated on the next I/O.

In the instance where two or more unprotected fields share the same data area, and the device operator enters updates into two or more such fields in the same I/O operation, the resulting contents of the data area are undefined.

7 – Actual Length
This is the length of the data in the data area(s).

When the application changes the data, it must set this to the length of data in the character, color, highlight, and symbol-set data areas in the data buffer. If not present an actual length of data-area length is assumed. If a value greater than data-area length is specified, then only data area length bytes are output. If the number of bytes output does not fill the field, then the rest of the field is filled with the pad character. (The pad character is null for character data and blank, meaning inherit the field attributes, for character attributes.)

If the device operator enters data into the field, GDDM sets actual length to the length of data, in bytes, now in the field, up to and including the last nonpad character.

GDDM only sets actual length if the field status indicates that changes to field contents have been input.

8 – Color Index
This is the index in the data buffer of the data area containing the color codes for individual characters that occupy the field. If not present, an index of 0 is assumed. An index of 0 indicates that there are no character color codes for the field.

9 – Highlight Index
This is the index in the data buffer of the data area that contains the highlight codes for individual characters that occupy the field. If not present an index of 0 is assumed. An index of 0 indicates that there are no character highlight codes for the field.

10 – Symbol-set Index
This is the index in the data buffer of the data area that contains the symbol-set codes for individual characters that occupy the field. If not present, an index of 0 is assumed. An index of 0 indicates that there are no character symbol-set codes for the field.

Fields that do not have character attributes should specify indices of 0. Omitting character attribute data areas, when not required, significantly improves the performance characteristics of an application.

Example

This is an example of a field list declaration in PL/I (compare with Figure 5 on page 110).

```
DCL FL(5,10) FIXED BIN(15) STATIC INIT

/*STA  DEP  WID  CSR  CSC                         */
 ( 1,   5,  10,   2,   5,   0,   0,   0,   0,   0,
/*STA  ROW  COL  WID  BLR  CHI  ACT  COI  HII  SSI*/
   1,   2,   5,   4,   2,   1,   4,   0,   0,   0,
   1,   4,  10,  11,   3,   5,  11,   0,   0,   0,
   1,   6,  15,  13,   4,  16,  13,   0,   0,   0,
   1,   8,  20,   3,   5,  29,   3,  32,  35,  38);
```

The data buffer

The **data buffer** consists of data areas containing the data and character attributes for each field defined in the field list. The position and size of each data area within the data buffer is defined in the field list. Each field-list entry contains the length and index into the data buffer of its character-data area. Optionally, it may also contain indexes to a character color data area, a character highlight data area, and a character symbol set data area.

Mixed double-byte and single-byte character fields

The internal representation of mixed character strings makes use of shift-out (SO) and shift-in (SI) control characters, X'0E' and X'0F', to indicate the start and end of a DBCS substring.

There are two ways of displaying mixed character strings, called mixed-with-position and mixed-without-position. The display method to be used is specified in the bundle definition for each field.

- Mixed-with-position

  The SO/SI codes occupy one character position each, and are displayed as either a blank or a special character – the terminal user can select which.

- Mixed-without-position.

  The SO/SI codes do not occupy a character position on the screen.

Character attributes

Character attributes are represented by these codes:

Color

blank	X'40'	Inherit the field color (the default)
1	X'F1'	Blue
2	X'F2'	Red
3	X'F3'	Magenta (pink)
4	X'F4'	Green
5	X'F5'	Turquoise (cyan)
6	X'F6'	Yellow
7	X'F7'	Neutral (white on displays, black on printers).

Highlight

blank	X'40'	Inherit the field highlight (the default)
1	X'F1'	Blink
2	X'F2'	Reverse video
4	X'F4'	Underscore.

Symbol-set

X'00' or X'40'	Inherit the field symbol set (the default)
X'01' through X'03'	Loadable symbol set (3800 system printer)
X'41' through X'DF'	Loadable symbol set (3270 family devices)
X'F1'	Alternative nonloadable symbol set (3270-family devices).

Notes:

1. The two character attributes, corresponding to the two bytes of a DBCS character, must both be the same.
2. Symbol-set character attributes, corresponding to DBCS characters, must be blank.

Example

The data buffer to go with the field lists in the earlier example might be:

```
DCL DB CHAR(40) STATIC INIT
     ('HighPerformanceAlphanumericsAPI356124 &&');
     /*↑   ↑          ↑              ↑  ↑  ↑  ↑  */
```

The field list has four fields defined, corresponding to the words `High`, `Performance`, `Alphanumerics`, and `API`. No color, highlight, or symbol set indexes have been specified for the first three fields. The field definition for the fourth defines a color index that selects the '356', a highlight index that selects the '124', and a symbol-set index that selects the ' &&' in the data buffer. (The blank specifies inheritance of the field symbol set, and the two '&' characters (X'50', decimal 80) request the use of a symbol set with identifier 80.)

The bundle list

The field attributes that are used with the alphanumeric fields defined in the field list, are themselves defined in the **bundle list**. Each field definition in the field list contains a bundle row, which is the row number of the bundle definition in the bundle list.

The first row of the bundle list is a header, and following rows contain field attribute bundle definitions. Each bundle definition consists of a status element, and the number of type-and-value pairs in the definition, followed by pairs of attribute types and attribute values describing the attributes of the bundle. It may also contain application data.

Figure 6 on page 114 illustrates the layout of a bundle list.

The bundle list can be declared as a structure, or as a two-dimensional array stored in row-major order. Programming languages which use column-major ordering of two dimensional arrays will have to exchange rows and columns in the description which follows. Below is a sample PL/I declaration for a bundle list, where "depth" and "width" are the array dimensions used in the API call APDEF:

```
DCL BUNDLE_LIST(depth,width) FIXED BIN(15);
```

The components of the bundle list are:

Bundle list header row

1 – List Status
The status of the bundle list.

Values that can be assigned to list status are the same as bundle status; in fact list status must always be equal to the value obtained by ORing together the values of all the bundle statuses in the bundle list. For example, whenever the application changes a bundle status indicator it must also change the list status.

2 – Used depth
The number of rows that GDDM uses in the bundle list.

Its value must be in the range 1 through list depth. It may be changed by the application in order to add new definitions or to remove unused definitions from the list. If this value is increased, the new bundle definitions must have the bundle changed indicator set in the bundle definition status element.

3 – Used width
The maximum number of elements in the header and each bundle definition used by GDDM.

This value must be less than or equal to the list width, and the minimum value is 4. If the value is less than the list width, then extra elements in the header and each bundle definition will be ignored by GDDM, and may be used by the application to record its own data. It may be changed by the application in order to extend or reduce the maximum number of type-and-value pairs in the bundle definitions.

Note: Although GDDM may not use as many elements in the header as in the bundle definitions, only those elements beyond the used width may be used for application data, the rest must be zero.

Bundle definition row

1 – Bundle status
The status of the bundle defintion. The list of values below shows both numerical value and corresponding bit position of the indicator.

1 — Bit 15 — Bundle changed
This must be set by the application to tell GDDM of changes made to the bundle definition, and, if set by the application, is reset by GDDM on the next I/O to the primary device involving the GDDM page. Set it if the number of pairs, the attribute types, or the attribute values, have been changed.

Note: All the other status indicators in the halfword must be zero.

2 – Pairs
The number of type-and-value pairs in the bundle definition.

The minimum value is 0 and the maximum value is (Used_width–2)/2. Elements in the bundle definition beyond this specified number are ignored by GDDM.

3 – Type-and-value pairs
Type is a code for the attribute type, such as "color" and value is a code for the corresponding value such as "blue".

The permitted type codes and their associated value codes are:

0 Dummy

This is a special type code that causes the type-and-value pair is to be ignored by GDDM. It effectively reserves space within the bundle definition for future use by the application. The associated value is ignored.

8 Field type

The permitted values are:

0 Unprotected alphanumeric (the default)
1 Alphanumeric output, numeric input
2 Protected alphanumeric.

16 Intensity

The permitted values are:

0 Invisible
1 Normal (the default)
2 Bright.

24 Color

The permitted values are:

0 Default
1 Blue
2 Red
3 Magenta (Pink)
4 Green
5 Turquoise (cyan)
6 Yellow
7 Neutral (white on color displays, black on printers).

32 SBCS Primary symbol set alias

The permitted values are:

0 Default. For a 3270-family device, the base nonloadable symbol set; for a 3800-system printer, the first loadable symbol set (use the CHARS parameter to specify the loaded symbol sets when printing).
1 through 3. For a 3800-system printer, the second, third, and fourth loadable symbol sets respectively (use the CHARS parameter to specify the loaded symbol sets when printing).
65 through 223. For a 3270 family device, loadable symbol sets corresponding to X'41' through X'DF'. The alias must be made known to GDDM with a call to PSDSS, PSLSS, or PSLSSC to load the symbol set.

40 Highlight

The permitted values are:

0 Normal (the default)
1 Blink
2 Reverse video
4 Underscore.

48 End

The permitted values are:

0 Autoskip (the default)
1 Notautoskip.

56 Transparency

The permitted values are:

0 Opaque (the default)
1 Transparent.

64 SBCS/DBCS

The permitted values are:

0 SBCS (the default)
1 Mixed-with-position
2 Mixed-without-position
3 DBCS.

Note: On 5550-family displays all unprotected fields on the device that are not DBCS, or mixed-with-position are enabled for mixed-without-position input if any bundle list on the device specifies mixed-without-position. If the device operator enters mixed-without-position data into a field, GDDM only places the correct shift-in, shift-out, and DBCS characters into the data buffer if mixed-without-position is specified for the field.

72 Outlining

The permitted values are:

0 None (the default)
1 Underline
2 Vertical line on right
4 Overline
8 Vertical line on left.

For an outlining attribute that is composed of more than one of these lines, specify the sum of the numbers corresponding to the lines required.

80 Modified data tag (MDT)

This defines the field MDT setting. It causes the physical MDT bit to be set so that the fields can be returned as input to a subsequent application program after GDDM terminates. This function is intended primarily for use under CICS/VS and IMS/VS.

The permitted values are:

0 Reset the MDT (the default)
1 Set the MDT.

88 Reply

This defines the character reply attribute. It specifies whether the device operator is able to enter color, highlight, or symbol-set character attributes into the field. If the field definition also specifies data areas for character attributes, GDDM will update the data areas with the attributes input.

The permitted values are:

0 Character reply mode off (the default)
1 Enable color character reply mode
2 Enable highlight character reply mode
4 Enable symbol-set character reply mode.

To enable combinations of color, highlight, and symbol-set character reply modes, specify the sum of the numbers corresponding to the enablements required.

Note: On 3270-family displays all unprotected fields in the real partition (or on the real screen if emulated partitions are being used) are enabled for character-attribute input if any bundle list on the page sets this attribute. If the device operator enters character attributes into a field, GDDM only places the character attributes input into the data buffer if the appropriate reply mode is enabled for the field.

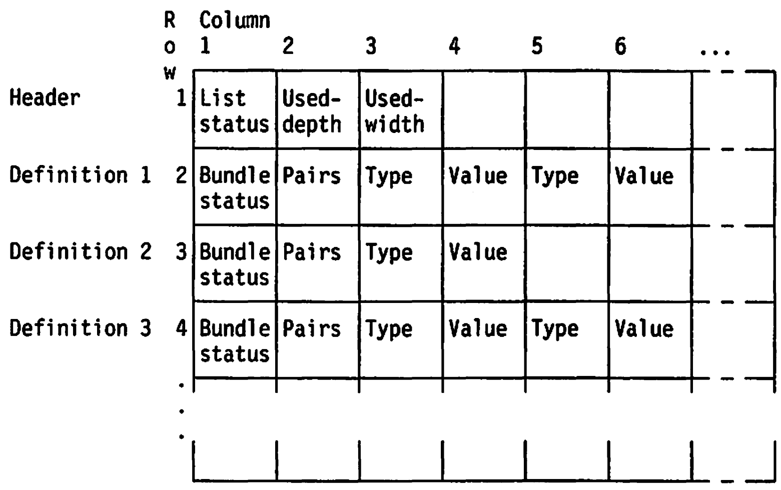
96 Pen detectable

This attribute permits selection of fields by a light pen or cursor select key.

The permitted values are:

0 Not pen detectable (the default)
1 Pen detectable.

The type of selection that occurs is determined by the first data character in the field; this character is called a designator character. A field having a "pen detectable" attribute but not starting with a valid designator character is not selectable.



	Row	Column 1	2	3	4	5	6	...
Header	1	List status	Used-depth	Used-width				
Definition 1	2	Bundle status	Pairs	Type	Value	Type	Value	
Definition 2	3	Bundle status	Pairs	Type	Value			
Definition 3	4	Bundle status	Pairs	Type	Value	Type	Value	
	. . .							

Figure 6. The bundle list array

The types of selection that can be set are:

Delayed detection. When selected by the device operator, the field is marked as "modified" but nothing is transmitted until the device operator performs another action associated with field modification (such as selecting an "immediate detection" field or pressing ENTER). The designator character for this type of field is "?" (X'6F'). If the field is selected the designator character changes to ">" (X'6E'); another selection restores it to "?" and cancels the modification indication.

Immediate detection without data. The designator character is a blank (X'40'). Selection of this type of field causes immediate input transmission. No data from any of the fields is transmitted, however. The effect is thus:

1. The ASREAD returns an Attention Type of 2 indicating light pen selection. All changes typed in by the device operator are lost.
2. GDDM restores all fields on the display to their original value at the next ASREAD (or other I/O call).

Immediate detection with data. The designator character is "&" (X'50'). The effect is the same as pressing ENTER. (Not possible with the IBM 3277 display terminal.)

Apart from dummy, the same type may not appear more than once in the same bundle definition.

Example

Below is an example of a declaration for a bundle list in PL/I:

```
DCL BL( 5,10) FIXED BIN(15) STATIC INIT

  /*STA DEP  WID                         */
  ( 0,  5, 10,  0,  0,  0,  0,  0,  0,  0,
  /*STA PRS TYP VAL COL VAL BDY VAL PSS VAL*/
     0,  3,  8,  0, 24,  3, 72,  1,  0,  0,
     0,  3,  8,  0, 24,  5, 72,  3,  0,  0,
     0,  4,  8,  0, 24,  6, 72, 15, 32, 80,
     0,  4,  8,  0, 24,  3, 72,  7, 88,  7);
```

How to use high-performance alphanumerics

Move mode and locate mode

There are two modes in which data can be transferred between GDDM and the application program, which are the **move** and **locate** modes. The mode is specified through the "mode" parameter of the APDEF call.

If move mode is specified, the field list, data buffer, and bundle list are copied by GDDM when APDEF is called. Subsequent output and input processing, done by GDDM, use the GDDM copies. When the application needs to retrieve updates made by the device operator, or modify the fields, it must query the field list, data buffer, and bundle list by calling APQRY. This returns copies of the field list, data buffer, and bundle list held by GDDM. When the application has modified the field list, data buffer, and bundle list, it must pass the modified versions back to GDDM by calling APMOD.

If locate mode is specified, GDDM does not copy the field list, data buffer, or bundle list. Subsequent output and input processing, by GDDM, use the copies in application storage. The application must not release the storage that these objects occupy until the field list has been deleted. The contents of the field list, data buffer, and bundle list must be valid whenever GDDM is called. When using locate mode, it is not necessary to call APQRY to determine device operator updates, nor to call APMOD in order to inform GDDM of changes made by the application.

The choice of move mode or locate mode will affect any application data embedded in the field list, data buffer, or bundle list. If move mode is used, this application data is copied by GDDM on APDEF and subsequent calls to APMOD. The value copied on the most recent APDEF or APMOD call is returned by GDDM on APQRY. This means that any changes made after APDEF or APMOD will be lost on the next call to APQRY. If locate mode is used this application data is not altered by GDDM.

Output

To display a page of alphanumeric fields proceed as follows:

- Construct the field list and associated data buffer and bundle list to describe the page of alphanumerics. The field definition statuses for all the fields to be shown should be set to 1. The field list status should be set to 1. The bundle list status, and all bundle definition statuses should be set to 0.
- Call APDEF to define the field list and associated data buffer and bundle list to GDDM.
- Call ASREAD, or another GDDM I/O call as required.

Input

To retrieve device operator updates to the page of alphanumeric fields following an I/O operation, proceed as follows:

- If using move mode, retrieve the field list, data buffer, and bundle list from GDDM by calling APQRY.
- Test the field list status input indicator to determine if any fields have been updated by the device operator. If they have, then test the field definition input indicators to determine which fields have been changed, and process the input found in the data buffer.
- If the alphanumerics are not to be reshown they should be cleared by calling APDEL.

Reshow

The application may need to reshow the page of alphanumeric fields just input, which should be done as follows:

- Reset the field list status input indicator and the field definition input indicators.
- Change the data or character attributes in the data buffer as required, and set the corresponding

output indicators in the field definition and header status.

- Change the bundle definitions in the bundle list as required, and set the corresponding bundle definition and header status indicators.
- Change the field definitions in the field list as required, and set the corresponding status indicators to specify what has changed.
- If using move mode, return the modified field list, data buffer, and bundle list to GDDM by calling APMOD.
- Call ASREAD, or another GDDM I/O call as required.

Field list update rules

The rules for altering a field list are:

- The input indicators, which indicate device operator updates, should be reset by the application after each I/O. If this is not done, the application will not be able to detect further updates on a subsequent I/O.
- Field row, field column, and field width may not be changed, except when using a previously-unused field definition entry to define a new field. Fields may be defined in any order, but must not overlap. They may wrap from row to row, but must not extend beyond the end of the page.
- Bundle row may be changed by the application, in which case the application must also set the output indicator to indicate to GDDM that this is changed. It is not necessary to set this indicator if only the bundle definition has changed and the field definition has not changed.
- If the character index, color index, highlight index, symbol-set index or actual length are changed, then the application must set the Output indicator to indicate to GDDM that the field has changed and is therefore to be output on the next I/O.
- When a previously unused field definition is activated, the process indicator and the create indicator must be set by the application. These indicators should never be reset by the application, only by GDDM.
- If an existing field is to be deleted, the field delete indicator should be set by the application. This indicator should never be reset by the application, only by GDDM, and the field definition entry may only be reused to define a new field after GDDM has reset the entire field status element.
- Changes to any field definition status indicator may also require changes to the corresponding header status indicator. The header status must always be set to the value obtained by ORing together all the field status elements.

Data buffer update rule

The rule for altering a data buffer is:

- If a character data area, or a character attribute data area is modified, then the output indicators in the corresponding field definition status and field list status must be set.

Bundle list update rule

The rule for altering a bundle list is:

- If a bundle definition is modified, the bundle changed indicator in the bundle definition status and bundle list status must be set.

Dynamic fields

Dynamic alphanumeric fields, using HPA, may be obtained by reserving space in the field list, data buffer, and bundle list for the fields to be added later. Reserved field definitions in the field list may be made by leaving the process indicator off. Reserved space may be left in the data buffer by not referring to it in existing field definitions. Reserved bundle definitions in the bundle list may be made by setting the number of type-and-value pairs to zero, or by using the dummy attribute type.

It may become necessary at some stage to enlarge the structures. When this happens, the APMOD call may be used to change the size of the field list, data buffer, or bundle list and also their location if using locate mode. The application must allocate new-larger data structures to replace the old ones, initialize them from the old ones (or by calling APQRY), call APMOD to define the enlarged versions to GDDM, and throw the old ones away.

Note: If APMOD is used in this way, any differences between the contents of the old and new structures must be indicated by change indicators as defined in the rules above.

Interpreted languages

In general, locate mode cannot be used by applications written in interpreted languages such as APL, BASIC, and REXX. When using these languages move mode must be used. See also the restrictions on shared storage below.

Read-only storage

In certain circumstances it may be desirable to use HPA with the field list, data buffer, or bundle list in read-only storage. An example might be an application that is used by many users at the same time. In this instance, it would be more efficient if fixed panel layouts were placed in shared storage. To use HPA from read-only storage, ensure that GDDM does not write to it by adhering to the rules below:

- Neither APDEF nor APMOD alters the storage of the field list, data buffer, or bundle list.
- In move mode, ASREAD does not alter the objects in user storage.
- In locate mode, ASREAD only alters:

The field list	If any of the create, delete, or output indicators are set, or if any field is unprotected or has the MDT attribute
The data buffer	If any field is unprotected or has the MDT attribute
The bundle list	If any status indicators are set.

Shared storage

When using locate mode, it is possible for an application to define more than one field list using the same storage. Field lists, data buffers, and bundle lists could all share storage. The rules for sharing storage are:

- Field lists may not share storage unless they are read only. See the section on Read-only storage on page 116.
- Bundle lists may be shared between more than one field list on the same device. They may not be shared between field lists on different devices unless they are read only.
- Data buffers may be shared between more than one field list only if unprotected data areas (that is, data areas corresponding to fields that are unprotected or have the MDT attribute) are not shared.

Note: Violations of these rules are not detected, and the results of such a violation are undefined.

Validation

To enable GDDM to be used as the device driver for fully tested program products, it is necessary to be able to run HPA without validation. (Validation is not necessary for tested applications and the performance advantages are significant.)

Validation checks the API parameters such as identifiers and lengths, as well as the field list, data buffer, and bundle list. The field list, data buffer, and bundle list are not validated during the API call processing as other parameters are, instead they are validated during processing for each I/O call involving the GDDM page.

If the writers of an application choose to use HPA without validation, they do so at their own risk. Incorrect use may result in device checks.

Validation is controlled by an external default as follows:

FRCEVAL — Force validation.
The default is NO. When FRCEVAL=YES is specified, the validation indicator in the mode parameter is overridden so that validation is always performed. The other indicators in the mode parameter are not affected.

For example, when a tested application (for instance, a shipped program product that does not use validation), is suspected of a bug, validation can be turned on to determine whether the application or GDDM is at fault by specifying:

```
ADMMDFT FRCEVAL=YES
```

in the external defaults file. This default may not be specified in the external defaults module, on SPINIT calls, or by API call.

Alternatively validation may be controlled by the mode parameter on the APDEF and APMOD API calls. It may be used during application development, but once an application is fully tested validation should be turned off.

Example program

The following sample program illustrates the use of HPA in locate mode. It displays a page with four fields, one of which uses character attributes. When the ENTER key is pressed the color of the first field is changed. When PF3 is pressed the program terminates.

```
/* EXAMPL    - SAMPLE CHARACTER ATTRIBUTES    */
EXAMPL:    PROC;

DCL TYP      FIXED BIN(31) STATIC INIT(1);
DCL VAL      FIXED BIN(31) STATIC INIT(3);
DCL CNT      FIXED BIN(31) STATIC INIT(0);
DCL ENDKEY   BIT(1) INIT('0'B);

DCL FL( 5,10)FIXED BIN(15) STATIC INIT
             /*STA  DEP  WID  CSR  CSC                          */
              ( 1,   5,  10,   2,   5,   0,   0,   0,   0,   0,
             /*STA  ROW  COL  WID  BLR  DAI  ACT  COI  HII  SSI*/
                1,   2,   5,   4,   2,   1,   4,   0,   0,   0,
                1,   4,  10,  11,   3,   5,  11,   0,   0,   0,
                1,   6,  15,  13,   4,  16,  13,   0,   0,   0,
                1,   8,  20,   3,   5,  29,   3,  32,  35,  38);
DCL BL( 5,10) FIXED BIN(15) STATIC INIT
             /*STA DEP  WID                         */
              ( 0,  5, 10,  0,  0,  0,  0,  0,  0,  0,
             /*STA PRS TYP VAL COL VAL BDY VAL PSS VAL*/
                0,  3,  8,  0, 24,  3, 72,  1,  0,  0,
                0,  3,  8,  0, 24,  5, 72,  3,  0,  0,
                0,  4,  8,  0, 24,  6, 72, 15, 32, 80,
                0,  4,  8,  0, 24,  3, 72,  7, 88,  7);
DCL DB     CHAR(40) STATIC INIT
           ('HighPerformanceAlphanumericsAPI356124 &&');

CALL FSINIT;

CALL PSLSS(0,'ADMITALC',80);    /*   load a symbol set              */

/*   Define a field list for the panel                              */
CALL APDEF(1,DIM(FL,1),DIM(FL,2),FL,LENGTH(DB),DB,
             DIM(BL,1),DIM(BL,2),BL,6);
     /* This uses the built in DIM feature of PL/I. */
     /* where DIM is the dimension of the array.    */
     /* It could have been coded as:     */
     /* CALL APDEF(1,5,10,FL,40,DB,5,10,BL,6);     */


/*   Display panel and process selection until END key pressed      */
DO   UNTIL(ENDKEY);
     CALL ASREAD(TYP,VAL,CNT);  /* Display panel                    */
     SELECT;                    /* Process selection                */
          WHEN(TYP=1 & (VAL=3 | VAL=15)) /* END key                 */
               ENDKEY = '1'B;

          WHEN(TYP=0) DO;   /* ENTER key alone - change field colour */
               BL(1,1) = 1; /* Set bundle list status */
               BL(2,1) = 1; /* Set bundle definition status */
               BL(2,6) = MOD(BL(2,6)+1,8); /* change color value */
               END;

          OTHERWISE CALL FSALRM; /* Error condition                 */
          END;
     END;
```

```
| CALL PSRSS(80);              /* Release a symbol set          */

| CALL FSTERM;

| %INCLUDE ADMUPINA;
| %INCLUDE ADMUPINF;
| %INCLUDE ADMUPINP;

| END EXAMPL;
```

Chapter 14. Country-extended code pages

Alphanumeric and graphics text characters are represented in main storage, disk files, and data streams as hexadecimal codes. Each code generally occupies one byte, although GDDM does support double-byte character sets also.

IBM devices attached to host computers use the Extended Binary Coded Decimal Interchange Code (EBCDIC) as the standard way of representing single-byte characters. However, problems arise because EBCDIC allows the characters represented by some of the codes to vary from one device to another.

The codes for the Latin letters (A through Z) in upper and lower case, and for Arabic numerals (0 through 9), are consistent across devices. But other codes, designated as being for national use, vary in the characters assigned to them, especially between devices made for different countries. Other codes are entirely device-dependent. Older EBCDIC devices generally supported 80 standard characters, 14 national-use assignments, and up to 96 device-dependent character codes.

For example, X'5B' is a national-use code, and on terminals made for the USA, it normally represents the dollar sign, $. But UK terminals normally use this code to represent the pound sign, £. And in the USA, X'4A' normally represents the cent sign, ¢, whereas in the UK it normally represents the dollar sign. So if you enter this data on a USA terminal:

```
Price: $1 large or 50¢ small
```

then store it in a file and redisplay it on a UK terminal, you would probably see:

```
Price: £1 large or 50$ small
```

Another type of problem is that a code generated by one device may have no corresponding character defined for it on another device. In other words, data may contain characters that are nondisplayable (or nonprintable) on some devices.

To overcome such problems, extensions to EBCDIC, called Country Extended Code Pages (CECPs), are supported. A *code page* is a mapping between hexadecimal codes and characters. The codes are often called, in this context, *code points*.

All CECPs contain the same set of characters and the same set of code points; there are 190 characters, and 190 code points ranging from X'41' through X'FE' (X'40' is not defined in the CECPs, but always represents a blank). The difference between one CECP and another is in the assignment of characters to code points, that is, the order in which the 190 characters are mapped onto the 190 code points.

The 190 characters are illustrated in Figure 7. Their mapping onto code points in all the CECPs supported by GDDM are shown in *GDDM Typefaces and Shading Patterns*. The supported CECPs are listed in Figure 9 on page 124.

CECP devices, such as the 3179, 3192, and 3193, can be queried from the host computer to discover the code pages they implement. GDDM takes advantage of this to make programs code-page independent.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
4x			â	ä	à	á	ã	å	ç	ñ	¢	.	<	(	+	|
5x	&	é	ê	ë	è	í	î	ï	ì	ß	!	$	*	)	;	¬
6x	-	/	Â	Ä	À	Á	Ã	Å	Ç	Ñ	¦	,	%	_	>	?
7x	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	@	'	=	"
8x	Ø	a	b	c	d	e	f	g	h	i	«	»	ð	ý	þ	±
9x	°	j	k	l	m	n	o	p	q	r	ª	º	æ	¸	Æ	¤
Ax	µ	~	s	t	u	v	w	x	y	z	¡	¿	Ð	Ý	Þ	®
Bx	^	£	¥	·	©	§	¶	¼	½	¾	[	]	¯	¨	´	×
Cx	{	A	B	C	D	E	F	G	H	I	-	ô	ö	ò	ó	õ
Dx	}	J	K	L	M	N	O	P	Q	R	¹	û	ü	ù	ú	ÿ
Ex	\	÷	S	T	U	V	W	X	Y	Z	²	Ô	Ö	Ò	Ó	Õ
Fx	0	1	2	3	4	5	6	7	8	9	³	Û	Ü	Ù	Ú	

Figure 7. CECP character set

The CECP character set and code points have been chosen to make migration as easy as possible. All standard EBCDIC characters and country-specific national-use characters are included in the CECP character set. Also the 80 standard EBCDIC characters have the same code points in all CECPs, and each country's CECP uses the same code points for its 14 EBCDIC national-use characters as before.

So, for example, the capital letter A is represented by X'C1' and the numeral 1 by X'F1' in the older scheme and in CECPs. The dollar sign is represented by X'5B' and cent sign by X'4A' in both the USA CECP and the older EBCDIC national-use assignments for the USA. In both the UK CECP and the older UK EBCDIC national-use assignments, the pound sign is represented by X'5B' and the dollar sign by X'4A'.

Because CECPs are generally supersets of each country's older EBCDIC codes, existing EBCDIC data usually appears on CECP devices with the correct characters. This is why CECPs were often transparent to users and programmers under releases of GDDM before Version 2 Release 2. GDDM now more fully exploits the capabilities of CECP devices.

GDDM code page concepts and facilities

A typical GDDM application program reads output data from files and sends it to terminals, printers, or plotters, and reads input data from terminals and stores it in files. The presentation of the output data on the screen or paper will contain the correct characters if the data in the files has the same code page as the output device. Otherwise, to be sure of obtaining the correct characters, the code page of the data must be converted before being transmitted. Similarly, input data may need to be converted after receipt from the terminal to be sure that all the data in the files uses the same code page.

GDDM performs these conversions on behalf of application programs. A programmer or end user can specify an *application code page* in which applications require to pass data to, and receive data from, GDDM. GDDM itself can determine the *device code page* (although this too can be specified). Output data passed by API calls is converted by GDDM from the application code page to the device code page before being sent to the device; and input data is converted from the device code page to the application code page before being returned by API calls to the application.

Code pages can be specified in nickname statements. No additional programming is usually required, and existing programs can benefit from the new function without being recompiled or re-link-edited. The default action for Version 2 Release 2 is to perform no code page conversion. Unless at least one GDDM default statement or API call explicitly specifying code page conversion is supplied, programs will run exactly as they did under releases of GDDM before Version 2 Release 2.

Conversion is carried out on all types of character data handled by the GDDM Base products, whether alphanumeric or graphics text, and whether generated by the application program, the end user, or GDDM itself. Besides input and output data, this includes, for example, window names and GDDM messages.

Besides devices and application programs, there are two other sources and targets for character data handled by GDDM: the operating system and GDDM object files. Each can employ its own particular code page. Altogether, GDDM can process four different types of code page on behalf of an application program, as shown in Figure 8 on page 123. The four types are:

- The device code page: the one used by a terminal, printer, or plotter for input or output.
- The application code page: the one used by the data passing between application programs and GDDM. It applies to *all* data flowing between a program and GDDM, whether passed to, or returned by, GDDM, and whether passed as a parameter on a call or in a data structure. It therefore applies to any character data that an application program reads from user files and passes to GDDM, in addition to characters coded literally in the program source.
- The GDDM object code page: the one used by data stored in a GDDM object file – such as a generated mapgroup, or a graphics data format (GDF) file – including any description that may be stored with the object. The object code page also applies to ADMPRINT files.
- The installation code page: the one used by data passed between GDDM and the operating system. Principally, this means the names of files.

As an example of code page conversion, consider an application program running with the USA CECP as the application code page, and the UK CECP as the device code page. If the application passes GDDM a code point of X'C1' in some output data, this is transmitted unchanged to the terminal: it represents the character A in both code pages. But if the application passes a code point of X'5B', this is converted to X'4A' before being transmitted to the terminal, because X'5B' represents the dollar sign in the USA CECP, and X'4A' represents the dollar sign in the UK CECP.

In addition to implicit conversion, GDDM applications can explicitly convert data from one specified code page to another using a GDDM Base call, FSTRAN. Calls providing code page functions are:

- ESQCPG – Query Code Page Of a GDDM Object
- ESQEUD – Query Encoded User Default Specification
- ESSCPG – Set Code Page Of a GDDM Object
- FSTRAN – Translate Character String.

They are described in *GDDM Base Programming Reference, Volume 1.*

A PL/I sample program, called ADMUSP7, is supplied that uses a selection of these calls.

The standard code pages supported by GDDM are listed in Figure 9 on page 124. The figure shows the global identifiers that uniquely identify the code pages.

What you should consider doing

These possible actions should be carefully considered by all installations:

1. Do nothing about CECP.

   This enables applications to run exactly as they did under releases of GDDM before Version 2 Release 2. However, it may mean that end-users are missing benefits from the CECP facilities and so this action is not recommended.

2. Specify the local national CECP as the default installation and application code pages for GDDM programs.

   These are the recommended minimum actions. They have the effect of switching on support for code-page conversion. Data from GDDM applications and GDDM objects is displayed, printed, and plotted as far as possible in the correct characters on any device.

   "Installation code page" and "Application code page" on page 124 explain how to specify these defaults.

3. In future, specify a single application code page within each new application.

   This ensures that the application always uses the specified code page, whatever the installation in which it is run, and whatever the code pages of the devices it uses. The code page could be a national CECP, or the multilingual code page (global code page identifier 00500).

   If such an application was exported to other countries, it would still use the same code page. Any files it created containing data passed to it from terminals would use the same code points, irrespective of the country in which it was executed.

   "Application code page" on page 124 explains how an application program can set its code page.

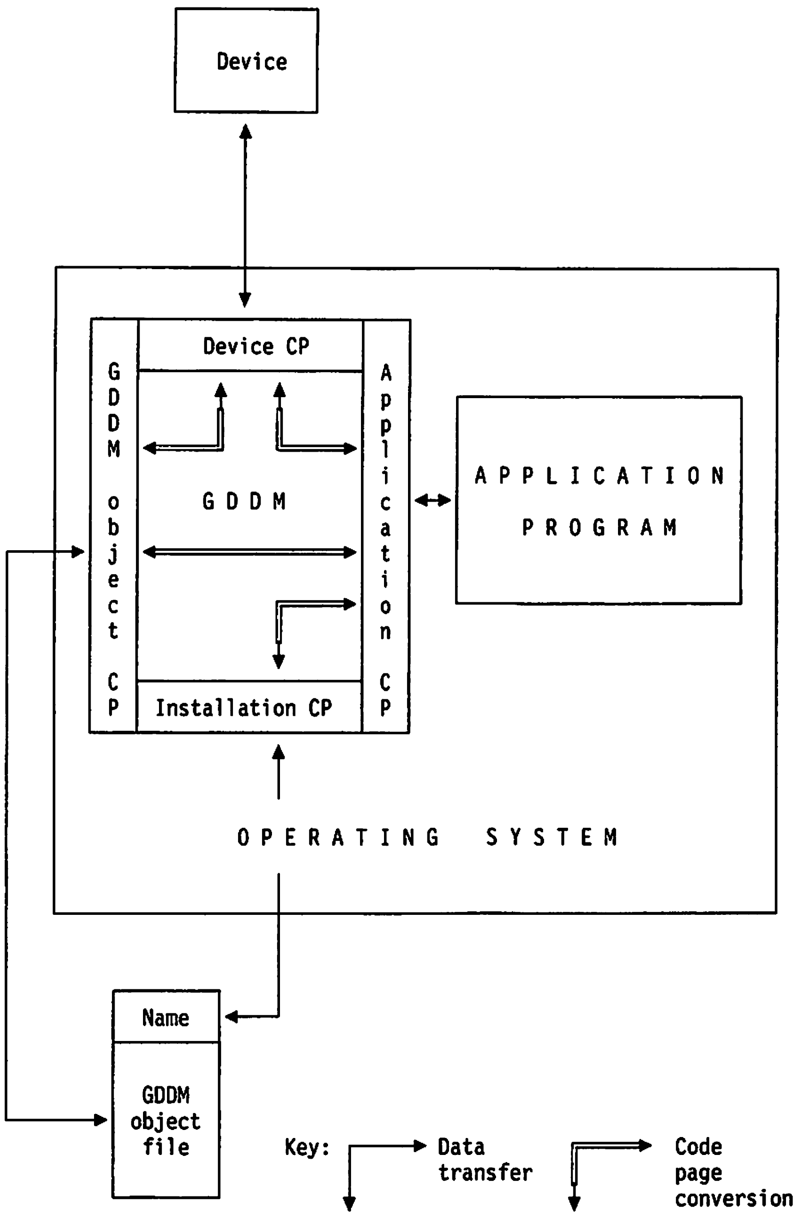


Figure 8. Code page conversion

4. In appropriate future applications, convert code pages explicitly.

   Programs that do this can handle data in a multiplicity of code pages. For example, a multinational enterprise might want to produce an international telephone directory by merging files created in several different countries. Using the FSTRAN call, a program can convert data from the various national code pages to a single standard one.

Code pages supported by GDDM

The code pages supported by GDDM are defined in translation tables contained in the alphanumeric defaults module, ADMDATRN. Those defined in the standard module are listed in Figure 9 on page 124. The module can be modified – see the *GDDM Installation and System Management* manual.

The two GDDM code pages, default EBCDIC (00351) and Katakana (00290), are special. If either the source or the target for any of the possible conversions shown in Figure 8 is a GDDM code page, *no conversion takes place*. The uses of these code pages are given in "Compatibility with releases of GDDM before Version 2 Release 2" on page 124.

Code pages 00351 and 00290 are illustrated in the description of the ASTYPE call in *GDDM Base Programming Reference, Volume 1*; CECPs are illustrated in the *GDDM Typefaces and Shading Patterns* booklet.

Name	Global code page identifier
CECP USA/Canada/Portugal[1]/ Netherlands	00037
CECP Austria/Germany	00273
CECP Brazil	00275
CECP Denmark/Norway	00277
CECP Finland/Sweden	00278
CECP Italy	00280
CECP Japan(Latin)	00281
CECP Spain/Latin America	00284
CECP United Kingdom/Ireland	00285
GDDM Katakana	00290
CECP France	00297
GDDM default EBCDIC	00351
CECP multi-lingual page (MLP)/ Switzerland/Belgium[2]	00500
CECP Iceland	00871

[1] 00037 has superseded 00282 for Portugal
[2] 00500 has superseded 00274 for Belgium

00282 and 00274 are supported for compatibility. They can be specified when an application needs to run with CECP devices or data that employ these early CECPs. Otherwise, neither should be specified.

Figure 9. Code pages supported as standard by GDDM

Specifying code pages

Application code page

This is specified in the GDDM default APPCPG. The application code page used if no APPCPG default is coded explicitly on an ADMMDFT macro, namely 00351, is the GDDM default EBCDIC code page, which has the effect of preventing code page conversion of all data passed to and from the application program.

Device code page

GDDM queries the code page support of devices when they are opened, that is, when an explicit or implicit DSOPEN call is executed. For devices that do not return this information, the device code page processing option (DEVCPG) may be used. A CECP identifier returned by the device is used as the device code page, unless a DEVCPG processing option has been specified for the device, in which case the processing option overrides the query reply. If no processing option has been specified, and the device does not return the information, the installation code page is used as the device code page.

Installation code page

This is specified in the GDDM default, INSCPG. The installation code page used if no INSCPG default is coded explicitly on an ADMMDFT macro, is 00037. This is the CECP for USA, Canada, Portugal, and the Netherlands.

More information about GDDM default statements is given in Appendix A, "GDDM's default values" on page 127.

Object code page

All GDDM object files saved after Version 2 Release 2 of a GDDM Base product has been installed are tagged with the application code page that is current at the time of saving. When an object is loaded, GDDM inspects its tag and uses the code page it contains as the object code page for that object.

If an object being loaded does not have a valid tag – usually this is because it was created under a release of GDDM before Version 2 Release 2 – GDDM uses the current application code page as the object code page.

The GDDM objects and their file-types are:

Graphics data format (GDF) files (ADMGDF)
Chart format files (ADMCFORM)
Chart data files (ADMCDATA)
Chart definition files (ADMCDEF)
Image data files (ADMIMG)
Projection definition files (ADMPROJ)
Saved pictures (FSSAVE) files (ADMSAVE)
GKS metafiles (ADMGKSM)
Symbol sets (ADMSYMBL)
Generated mapgroups (ADMGGMAP).

ADMGDF object files converted from PIF by the ADMUPCx utilities are tagged with an object code page equal to the current application code page when they are created. However, the objects' contents are not converted.

ADMPRINT files are also tagged with an object code page when they are created, for use by the GDDM Print Utility.

There is a GDDM Base call and an end-user utility for explicitly tagging GDDM objects. The call, ESSCPG, is described in *GDDM Base Programming Reference, Volume 1*, and the utility, ADMUOT, is described below.

Compatibility with releases of GDDM before Version 2 Release 2

GDDM code pages

In some circumstances, an installation may want programs to continue to operate without code page conversion. For this purpose, GDDM provides a special code page called GDDM default EBCDIC (00351). If it is specified as the application code page, it has the effect of preventing all CECP code page conversion, apart from explicit conversions using the FSTRAN call.

For Version 2 Release 2, if no installation or application code page is specified, and no tagged GDDM objects are used, GDDM default EBCDIC is used by GDDM as the application and object code pages, which prevents implicit conversion. Thus, if no action is taken to specify code pages, programs will run as they did under releases of GDDM before Version 2 Release 2.

For Katakana applications, there is a GDDM Katakana code page, 00290. When this is specified as the application code page, it prevents CECP-type code page conversion. However, data transmitted to and from devices is converted as if an ASTYPE call with a parameter value of 3 had been executed.

Inhibiting input of extended code points

Some terminals, such as the 3179-G, allow the host computer to specify whether keyboard input of all 190 CECP code points is to be allowed. If disallowed, only a base set (of, typically, 94 code points) can be entered. Attempting to enter one of the new CECP code points puts the terminal into the input-inhibit state.

An external default, CECPINP, controls this function. It enables existing applications to be protected from new code points. It does not affect the use of the new code points in output data, nor the display and printing of the full range of CECP characters.

Code page conversion in GDDM objects

Graphics data format files: The code page of graphics text in ADMGDF-type files is converted.

Symbol sets: These are converted only if they contain a character in every CECP code point – or, more precisely, in X'41' through X'FE' for image symbol sets, and X'42' through X'FE' for vector symbol sets.

Generated mapgroups: The code page of alphanumeric data in ADMGGMAP-type files is converted.

ADMIMG, and ADMPROJ files: These contain character data in the description only; this data is converted.

Interactive Chart Utility (ICU): This is part of GDDM-PGF, which has not been changed for GDDM Version 2 Release 2. It therefore does not have any extended code page functions, so no conversion of data in ADMCFORM, ADMCDATA, and ADMCDEF files is carried out by the ICU. However, when the ICU saves these files, they are tagged with the current application code page, because the ICU uses GDDM Base facilities to do the saving.

Converting ICU charts

The character data in chart format and data files can be converted explicitly using the FSTRAN call. A sample application program, called ADMUSP7, is supplied that does this. It is written in PL/I, and the source code is supplied. It converts the data from the object code page to the current application code page.

Editing symbol sets

Before using the Vector or Image Symbol Editor, the user should ensure that the application code page is the same as that of the symbol set being edited.

If a new symbol set is created, or if an existing symbol set with a code page equal to the installation code page is edited, the application code page can be allowed to default. However, if a foreign symbol set – one that has a code page different from the installation – is to be edited, the application code page must be set explicitly before the editor is invoked. Here is an example default statement for doing this:

```
DEFAULT APPCPG=nnnnn
```

where nnnnn is the global code page identifier of the symbol set to be edited.

If a foreign symbol set is edited without doing this, the wrong symbols may be displayed during editing, and the symbol set, when saved, will be tagged with the wrong code page.

Symbol set code pages can be queried by a program executing the ESQCPG call (refer to the *GDDM Base Programming Reference, Volume 1*.)

All the GDDM symbol sets containing the CECP set of characters are supplied in the order of code page 00037.

Utility program for tagging GDDM object files (ADMUOT)

This utility uses the ESSCPG call to tag a specified GDDM object file with a specified code page. It runs under TSO or CMS only. CMS end users invoke it from their terminals by entering the following (assuming the utility module has not been renamed):

```
ADMUOT objname objtype cpgid
```

TSO end users enter the following (assuming the module has not been renamed and is stored in SYS1.PROCLIB):

```
CALL 'SYS1.PROCLIB(ADMUOT) objname objtype cpgid'
```

where

objname is the name of the object
objtype is an integer identifying the type of object. The valid values and their meanings are the same as for the ESSCPG call (refer to the *GDDM Base Programming Reference, Volume 1*).
cpgid is the code page identifier. The standard GDDM identifiers are listed on page 124.

The name of the object is converted by the utility from the application code page to the installation code page. Normally these are the same.

Code page conversion by GDDM Print Utility

The object code page tag in GDDM print files (type ADMPRINT) is used by the GDDM Print Utility.

The utility converts any CECP data in the file from the object code page to the device code page.

APL characters

The CECP character set does not include APL characters. Applications requiring APL characters can either use GDDM default EBCDIC (00351) as the application code page, or they can use the ASCSS or GSCS call to specify the alternative nonloadable symbol set.

4250 printer code page function

This function, in which a code page is specified using a GSCPG call or CPN4250 external default parameter, applies only when the current device is a 4250 printer. If a 4250 (type 5) symbol set is specified in a GSLSS call, the specified symbol set will be loaded using the code page defined in the GSCPG call or in the external default. It will not be affected by the current GDDM device or application code page. This function therefore remains independent of the code page functions.

Symbol sets

Most of the GDDM sample symbol sets, which contain single-byte Latin characters, contain the full set of 190 CECP characters. All the CECP symbol sets are listed in Chapter 8, "Symbol sets" on page 65.

ADMDVECP will be used as the default vector symbol set provided a CECP has been specified as the application code page. If the GDDM default EBCDIC code page, 00351, is the application code page, the vector symbol set ADMDVSS will be used instead.

Appendix A. GDDM's default values

This Appendix contains information on the following:

- Changing GDDM's default values
- GDDM external defaults, listed by subsystem:
  - CICS/VS on pages 128 through 131
  - IMS/VS on pages 131 through 133
  - TSO on pages 134 through 136
  - VM/CMS on pages 137 through 139
  - VSE/Batch on pages 140 through 141.
- Alphabetic list of default descriptions on pages 142 through 148.

GDDM's default values, listed by subsystem

This section describes the options you can specify to change defaults for your GDDM and subsystem environment. The information is presented in tabular form and is organized in alphabetic order of subsystem, thus; CICS/VS, IMS/VS, TSO, VM/CMS, and VSE/Batch.

Full descriptions of the defaults are given under "Alphabetic list of GDDM default values" on page 142, where they are listed in alphabetic order of the user default specification parameter.

The first four columns of each table give a brief meaning of the option, the source format of the user default specification (UDS) to change that option, the GDDM default for that option, and the encoded format of the UDS. The final column shows the methods of implementing the UDS you have specified; it shows where the UDS can be specified, as follows:

M	in the External Defaults Module,
F	in the External Defaults File,
S	in the SPINIT call,
C	in the ESEUDS and ESSUDS calls.

Note that not all defaults can be specified by all of the methods; some defaults can be specified by only one of the methods.

Changing GDDM's default values

The default values supplied by GDDM can be changed to allow for variations in such things as specific operating environments, equipment availability, or user requirements. For full details, see Chapter 1, "Customizing your program and its environment" on page 1.

If a default keyword is specified without a value, the current default value is not changed. For example, in:

```
DEFAULT ERRTHRS=,NATLANG=F
```

the ERRTHRS keyword has no effect.

A default value of blanks can be defined by specifying it as a null string enclosed in parentheses. For example:

```
DEFAULT TSOS99U=()
```

The tables that follow list, in alphabetic order of default function, the GDDM defaults you can change for each subsystem environment, together with their source-format and equivalent encoded-format user default specifications.

Note that in defaults files, the "ADMMDFT" keyword can be replaced by "DEFAULT".

GDDM external defaults – CICS/VS

Table 16 (Page 1 of 3). GDDM defaults — options for CICS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Alphanumeric defaults module control	DATRN=addr	ADMDATRN	3,118,addr	Y N Y Y
Always-unlock-keyboard	AUNLOCK= {NO|YES}	NO	3,10,{0|1}	Y Y Y Y
Application code page	APPCPG=n	00351	3,125,n	Y Y Y Y
Audit trail anchor block addresses: storage: program:	CICAUD=(stg-addr, pgm-addr)	0,0 (none)	4,1201, A(STGANCH), A(PGMANCH)	N N Y N
Call information feed-back block: length: address:	CALLINF= (len,addr)	0,0 (none)	4,1101, L(CIB), A(CIB)	N N Y N
CECP keyboard input	CECPINP={YES|NO}	YES	3,126,{1|0}	Y Y N N
CICS device query temporary storage prefix	CICTQRY=aaaa	ADMQ	3,211,aaaa	Y Y Y Y
Comments for module identification	COMMENT= (ccccccc, cccccccc ,.......)	N/A	1-8000,0,cccc, cccc,....	Y Y Y Y
Compressed PS loads	IOCOMPR= {NO|YES}	YES	3,9,{0|1}	Y Y Y Y
Date convention	DATEFRM= {1|2|3|4}	4	3,5,{1|2|3|4}	Y Y Y Y
DBCS default selection	DBCSDFT= {GDDM|NO|YES}	GDDM	3,18,{0|1|2}	Y Y Y Y
DBCS SO/SI emulation character	SOSIEMC=c	"	3,110,X'xx000000'	Y Y Y Y
DBCS strings with shift-out/shift-in	MIXSOSI= {NO|YES}	NO	3,17,{0|1}	Y Y Y Y
DBCS symbol set component in-core threshold	DBCSLIM=n	4	3,113,n	Y Y Y Y
DBCS symbol set language	DBCSLNG=c	K	3,111,X'xx000000'	Y Y Y Y
Deck output transient data name	CICDECK=aaaa	ADMD	3,202,aaaa	Y Y Y N
Defaults file temporary storage prefix	CICDFPX=aaaa	ADMD	3,210,aaaa	Y N Y N
Device attachment	AM3270= ({LOCREM| REMOTE| LOCAL} ,{SNANOSNA| NONSNA| SNA})	LOCREM SNANOSNA	4,12,{0| 1| 2} {0| 1| 2}	Y Y Y Y

Table 16 (Page 2 of 3). GDDM defaults — options for CICS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Error exit: use GDDM-supplied feed-back block	ERRFDBK= (GDDMDFLT)	GDDMDFLT	3,1102,0	Y N Y Y
Error exit: use user-supplied feedback block	ERRFDBK= (USERAREA, addr,len	—	5,1102,2,addr,len	Y N Y Y
Error threshold value	ERRTHRS=n	4	3,101,n	Y Y Y Y
Force validation of HPA	FRCEVAL={NO|YES}	NO	3,127,{0|1}	N Y N N
Form feed	FF3270P={NO| AFTER| BEFORE| BOTH}	AFTER	3,11,0 3,11,1 3,11,2 3,11,3	Y Y Y Y
FSSAVE buffer size	SAVBFSZ=n	1024	3,105,n	Y Y Y Y
ICU isolate value	ICUISOL={0|1|2}	0	3,112,{0|1|2}	Y Y Y Y
ICU panel color	ICUPANC= {TURQUOISE |BLUE}	TURQ	3,120,{5|1}	Y Y Y Y
ICU symbol sets	ICUFMSS={0|1|2}	0	3,122,{0|1|2}	Y Y Y Y
ICU format	ICUFMDF={0|1|2}	0	3,121,{0|1|2}	Y Y Y Y
GDDM-IMD ADMGIMP file-control name	CICGIMP=aaaaaaaa	ADMGIMP	4,203,aaaa,aaaa	Y Y N N
GDDM-IMD ADS output transient data name	CICIADS=aaaa	ADMG	3,207,aaaa	Y Y N N
GDDM-IMD staged data file-type	CICIFMT=aaaaaaaa	ADMIFMT	4,208,aaaa,aaaa	Y Y N N
GDDM-IMD staging file file-control name	CICSTGF=aaaaaaaa	ADMX	4,209,aaaa,aaaa	Y Y N N
Installation code page	INSCPG=n	00037	3,124,n	Y N N N
Mapgroup storage threshold	MAPGSTG=n	8192	3,106,n	Y Y Y Y
National language	NATLANG=c	A	3,4,X'xx000000'	Y Y Y N
No operation	—	—	{0|1}	Y N Y Y
Number convention	NUMBFRM={1|2|3}	1	3,7,{1|2|3}	Y Y Y Y
Parameter verification (SPI)	PARMVER={NO|YES}	NO	3,1,{0|1}	N N Y N
Print Utility Temporary Storage prefix	CICTSPX=aaaa	ADMT	3,204,aaaa	Y Y Y N
Print Utility transaction name	CICPRNT=aaaa	ADMP	3,205,aaaa	Y Y Y N
Short-on-storage proc-essing	STGRET={NO|YES}	NO	3,2,{0|1}	N N Y N
Synchronized I/O	IOSYNCH={NO|YES}	NO	3,8,{0|1}	Y Y Y Y

Table 16 (Page 3 of 3). GDDM defaults — options for CICS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values — list of full-words	Valid in: M F S C
System printer output transient data name	CICSYSP=aaaa	ADMS	3,206,aaaa	Y Y Y N
Time convention	TIMEFRM={1|2|3|4}	1	3,6,{1|2|3|4}	Y Y Y Y
Trace table size, in-core	TRTABLE=n	100	3,103,n	Y Y Y N
Trace output transient data name	CICTRCE=aaaa	ADMT	3,201,aaaa	Y Y Y N
Trace control	TRCESTR='aaaaaaa'	(none)	3,114,aaaa,bbbb,..	Y Y Y Y
Trace output width	TRCEWID={SINGLE| DOUBLE}	SINGLE	3,115,{0|1}	Y Y Y Y
Trace word value	TRACE={0|n}	0	3,102,n	Y Y Y Y
Transaction independence	CICTIF={NO|YES}	NO	3,14,{0|1}	N N Y N
Transmission buffer size	IOBFSZ=n	1536	3,104,n	Y Y Y Y
User Control SAVE function control	CTLSAVE={YES|NO}	NO	3,119,{0|1}	Y Y Y Y
VSAM data-set names for:	OBJFILE=(aaaaaaaa, bbbbbbbb ,...)		4-16,107,	Y Y Y Y
Symbol sets		ADMF	aaaa,aaaa,	
Generated mapgroups		ADMF	bbbb,bbbb,	
Saved pictures		ADMF	cccc,cccc,	
Chart formats		ADMF	dddd,dddd,	
Chart data		ADMF	eeee,eeee,	
GDDM-IMD tutorial pages		ADMGIMP	ffff,ffff,	
GDF files		ADMF	gggg,gggg	
(reserved)		—	hhhh,hhhh	
(reserved)		—	iiii,iiii	
Projection definitions		ADMF	jjjj,jjjj	
Image data		ADMF	kkkk,kkkk	

GDDM external defaults – IMS/VS

Table 17 (Page 1 of 3). GDDM defaults — options for IMS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Alphanumeric defaults module control	DATRN=addr	ADMDATRN	3,118,addr	Y N Y Y
Always-unlock-keyboard	AUNLOCK={NO|YES}	YES	3,10,{0|1}	Y N Y Y
Application code page	APPCPG=n	00351	3,125,n	Y Y Y Y
Call information feedback block:	CALLINF=(len,addr)	0,0 (none)	4,1101,	N N Y N
length:			L(CIB),	
address:			A(CIB)	
CECP keyboard input	CECPINP={YES|NO}	YES	3,126,{1|0}	Y Y N N
Comments for module identification	COMMENT=(cccccccc, cccccccc, ...)	N/A	1-8000,0,cccc,cccc ,....	Y N Y Y
Compressed PS loads	IOCOMPR={NO|YES}	YES	3,9,{0|1}	Y N Y Y
Data base DBD names for:	OBJFILE=(aaaaaaaa, bbbbbbbb ,...)		4-16,107,	Y N Y N
Symbol sets		ADMOBJ1	aaaa,aaaa,	
Generated mapgroups		ADMOBJ1	bbbb,bbbb,	
Saved pictures		ADMOBJ1	cccc,cccc,	
Chart formats		ADMOBJ1	dddd,dddd,	
Chart data		ADMOBJ1	eeee,eeee,	
(reserved)		—	ffff,ffff,	
GDF files		ADMOBJ1	gggg,gggg	
(reserved)		—	hhhh,hhhh	
(reserved)		—	iiii,iiii	
Projection definition		ADMOBJ1	jjjj,jjjj	
Image data		ADMOBJ1	kkkk,kkkk	
Date convention	DATEFRM={1|2|3|4}	4	3,5,{1|2|3|4}	Y N Y Y
DBCS default selection	DBCSDFT= {GDDM| NO|YES}	GDDM	3,18,{0|1|2}	Y Y Y Y
DBCS SO/SI emulation character	SOSIEMC=c	"	3,110,X'xx000000'	Y N Y Y
DBCS strings with shift-out/shift-in	MIXSOSI={NO|YES}	NO	3,17,{0|1}	Y N Y Y
DBCS symbol set component in-core threshold	DBCSLIM=n	4	3,113,n	Y Y Y Y
DBCS symbol set language	DBCSLNG=c	K	3,111,X'xx000000'	Y N Y Y
Deck output LTERM name	IMSDECK=aaaaaaaa	ADMDECK	4,302,aaaa,aaaa	Y N Y N
Device attachment	AM3270= ({LOCREM| REMOTE| LOCAL},	LOCREM	4,12, {0| 1| 2}	Y N Y Y
	{SNANOSNA| NONSNA| SNA})	SNANOSNA	{0| 1| 2}	

Table 17 (Page 2 of 3). GDDM defaults — options for IMS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Error exit: use GDDM-supplied feedback block	ERRFDBK= (GDDMDFLT)	GDDMDFLT	3,1102,0	Y N Y Y
Error exit: use user-supplied feedback block	ERRFDBK= (USERAREA, addr,len)	—	5,1102,2,addr,len	Y N Y Y
Error threshold value	ERRTHRS=n	8	3,101,n	Y N Y Y
Force validation of HPA	FRCEVAL={NO|YES}	NO	3,127,{0|1}	N Y N N
Form feed	FF3270P= {NO| AFTER| BEFORE| BOTH}	AFTER	3,11,0 3,11,1 3,11,2 3,11,3	Y N Y Y
FSSAVE buffer size	SAVBFSZ=n	1024	3,105,n	Y N Y Y
GDDM message output descriptor (MOD) name	IMSMODN=aaaaaaaa	DFS.EDT	4,317,aaaa,aaaa	Y N Y Y
GDDM system definition data base DBD name	IMSSDBD=aaaaaaaa	ADMSYSDF	4,307,aaaa,aaaa	Y N Y N
ICU isolate value	ICUISOL={0|1|2}	0	3,112,{0|1|2}	Y N N N
ICU panel color	ICUPANC= {TURQUOISE |BLUE}	TURQ	3,120,{5|1}	Y Y Y Y
ICU symbol sets	ICUFMSS={0|1|2}	0	3,122,{0|1|2}	Y Y Y Y
ICU format	ICUFMDF= {0|1|2}	0	3,121, {0|1|2}	Y Y Y Y
Input area size	IMSUISZ=n	3000	3,310,n	Y N N N
Installation code page	INSCPG=n	00037	3,124,n	Y N N N
Interactive Utility exit character string	IMSEXIT=aaaaaaaa	EXIT	4,311,aaaa,aaaa	Y N N N
Interactive Utility shutdown LTERM name	IMSMAST=aaaaaaaa	MASTER	4,313,aaaa,aaaa	Y N N N
Interactive Utility shutdown string	IMSSHUT=aaaaaaaa	SHUTDOWN	4,312,aaaa,aaaa	Y N N N
Mapgroup storage threshold	MAPGSTG=n	8192	3,106,n	Y Y Y Y
Maximum number of users	IMSUMAX=n	5	3,309,n	Y N N N
National language	NATLANG=c	A	3,4,X'xx000000'	Y N Y N
No operation	—	—	{0|1}	Y N Y Y
Number convention	NUMBFRM={1|2|3}	1	3,7,{1|2|3}	Y N Y Y
Parameter verification (SPI)	PARMVER={NO|YES}	NO	3,1,{0|1}	N N Y N

Table 17 (Page 3 of 3). GDDM defaults — options for IMS/VS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values — list of full-words	Valid in: M F S C
Print Utility transaction name	IMSPRNT=aaaaaaaa	ADMPRINT	4,303,aaaa,aaaa	Y N Y N
Segment/Key field names:	IMSSEGS= (aaaaaaaa , bbbbbbbb ,...)		14,308,	Y N Y N
Object data base root segment		ADMOBROO	aaaa,aaaa	
Object data base dependent segment		ADMOBDEP	bbbb,bbbb	
Object data base root key field		ADMOBRKY	cccc,cccc	
Object data base dependent key field		ADMOBDKY	dddd,dddd	
System definition data base segment		ADMSDSGM	eeee,eeee	
System definition data base key field		ADMSDKEY	ffff,ffff	
Short-on-storage processing	STGRET={NO|YES}	NO	3,2,{0|1}	N N Y N
System printer output destination name	IMSSYSP=aaaaaaaa	ADMLIST	4,314,aaaa,aaaa	Y N Y N
Time convention	TIMEFRM={1|2|3|4}	1	3,6,{1|2|3|4}	Y N Y Y
Trace output ddname	IMSTRCE=aaaaaaaa	ADMTRACE	4,301,aaaa,aaaa	Y N Y N
Trace table size, in-core	TRTABLE=n	100	3,103,n	Y N Y N
Trace control	TRCESTR='aaaaaaa'	(none)	3,114,aaaa,bbbb	Y Y Y Y
Trace output width	TRCEWID={SINGLE| DOUBLE}	SINGLE	3,115,{0|1}	Y Y Y Y
Trace word value	TRACE={0|n}	0	3,102,n	Y N Y Y
Transaction name for Image Symbol Editor	IMSISE=aaaaaaaa	ISSE	4,304,aaaa,aaaa	Y N N N
Transaction name for Interactive Chart Utility	IMSICU=aaaaaaaa	CHART	4,306,aaaa,aaaa	Y N N N
Transaction name for Vector Symbol Editor	IMSVSE=aaaaaaaa	VSSE	4,305,aaaa,aaaa	Y N N N
Transmission buffer size	IOBFSZ=n	1536	3,104,n	Y N Y Y
User Control SAVE function control	CTLSAVE={YES|NO}	NO	3,119,{0|1}	Y Y Y Y
Write-to-operator descriptor codes	IMSWTOD=(n,n,n,..)	(7)	3,316,X'xxxx0000'	Y N Y N
Write-to-operator routing codes	IMSWTOR=(n,n,n,..)	(2)	3,315,X'xxxx0000'	Y N Y N

GDDM external defaults – TSO

Table 18 (Page 1 of 3). GDDM defaults — options for TSO

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Alphanumeric defaults module control	DATRN=addr	ADMDATRN	3,118,addr	Y N Y Y
Always-unlock-keyboard	AUNLOCK={NO|YES}	NO	3,10,{0|1}	Y Y Y Y
APL default specification	TSOAPLF= {DATAANAL| APLTEXT}	DATAANAL	3,16,{0|1}	Y Y Y Y
Application code page	APPCPG=n	00351	3,125,n	Y Y Y Y
Call information feedback block: length: address:	CALLINF=(len,addr)	0,0 (none)	4,1101, L(CIB), A(CIB)	N N Y N
CECP keyboard input	CECPINP={YES|NO}	YES	3,126,{1|0}	Y Y N N
Comments for module identification	COMMENT= (cccccccc, cccccccc, ...)	N/A	1-8000,0,cccc,cccc ,....	Y Y Y Y
Compressed PS loads	IOCOMPR={NO|YES}	YES	3,9,{0|1}	Y Y Y Y
Date convention	DATEFRM={1|2|3|4}	4	3,5,{1|2|3|4}	Y Y Y Y
DBCS default selection	DBCSDFT= {GDDM| NO|YES}	GDDM	3,18,{0|1|2}	Y Y Y Y
DBCS SO/SI emulation character	SOSIEMC=c	"	3,110,X'xx000000'	Y Y Y Y
DBCS strings with shift-out/shift-in	MIXSOSI={NO|YES}	NO	3,17,{0|1}	Y Y Y Y
DBCS symbol set component in-core threshold	DBCSLIM=n	4	3,113,n	Y Y Y Y
DBCS symbol set language	DBCSLNG=c	K	3,111,X'xx000000'	Y Y Y Y
ddnames for:	OBJFILE= (aaaaaaaa, bbbbbbbb, ...)		4-16,107,	Y Y Y Y
Symbol sets		ADMSYMBL	aaaa,aaaa,	
Generated mapgroups		ADMGGMAP	bbbb,bbbb,	
Saved pictures		ADMSAVE	cccc,cccc,	
Chart formats		ADMCFORM	dddd,dddd,	
Chart data		ADMCDATA	eeee,eeee,	
GDDM-IMD tutorial pages		ADMGIMP	ffff,ffff,	
GDF files		ADMGDF	gggg,gggg	
(reserved)		—	hhhh,hhhh	
Chart data definition		ADMCDEF	iiii,iiii	
Projection definition		ADMPROJ	jjjj,jjjj	
Image data		ADMIMG	kkkk,kkkk	
Deck output ddname	TSODECK=aaaaaaaa	ADMDECK	4,402,aaaa,aaaa	Y Y Y N
Defaults file ddname	TSODFTS=aaaaaaaa	ADMDEFS	4,411,aaaa,aaaa	Y N Y N

Table 18 (Page 2 of 3). GDDM defaults — options for TSO

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values — list of full-words	Valid in: M F S C
Device attachment	AM3270=({LOCREM| REMOTE| LOCAL} ,{SNANOSNA| NONSNA| SNA})	LOCREM SNANOSNA	4,12, {0|, 1| 2} {0| 1| 2}	Y Y Y Y
Error exit: use GDDM-supplied feedback block	ERRFDBK=(GDDMDFLT)	GDDMDFLT	3,1102,0	Y N Y Y
use user-supplied feedback block	ERRFDBK=(USERAREA) addr,len)	—	5,1102,2,addr,len	Y N Y Y
Error threshold value	ERRTHRS=n	4	3,101,n	Y Y Y Y
Force validation of HPA	FRCEVAL={NO|YES}	NO	3,127,{0|1}	N Y N N
Form feed	FF3270P= {NO| AFTER| BEFORE| BOTH}	AFTER	3,11,0 3,11,1 3,11,2 3,11,3	Y Y Y Y
FSSAVE buffer size	SAVBFSZ=n	1024	3,105,n	Y Y Y Y
High-resolution image generation; color ddname or high-level qualifier	TSOCOLM=aaaaaaaa	ADMCOL+	4,409,aaaa,aaaa	Y Y Y N
monochrome ddname or high-level qualifier	TSOMONO=aaaaaaaa	ADMIMAGE	4,408,aaaa,aaaa	Y Y Y N
ICU isolate value	ICUISOL={0|1|2}	0	3,112,{0|1|2}	Y Y Y Y
ICU panel color	ICUPANC= {TURQUOISE |BLUE}	TURQ	3,120,{5|1}	Y Y Y Y
ICU symbol sets	ICUFMSS={0|1|2}	0	3,122,{0|1|2}	Y Y Y Y
ICU format	ICUFMDF={0|1|2}	0	3,121,{0|1|2}	Y Y Y Y
GDDM-IMD ADMGIMP ddname	TSOGIMP=aaaaaaaa	ADMGIMP	4,403,aaaa,aaaa	Y Y N N
GDDM-IMD ADS output ddname	TSOIADS=aaaaaaaa	ADMGNADS	4,406,aaaa,aaaa	Y Y N N
GDDM-IMD Export data ddname	TSOIFMT=aaaaaaaa	ADMIFMT	4,407,aaaa,aaaa	Y Y N N
Installation code page	INSCPG=n	00037	3,124,n	Y N N N
Map group storage threshold	MAPGSTG=n	8192	3,106,n	Y Y Y Y
National language	NATLANG=c	A	3,4,X'xx000000'	Y Y Y N
No operation	—	—	{0|1}	Y N Y Y
Number convention	NUMBFRM={1|2|3}	1	3,7,{1|2|3}	Y Y Y Y

Table 18 (Page 3 of 3). GDDM defaults — options for TSO

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Parameter verification (SPI)	PARMVER={NO|YES}	NO	3,1,{0|1}	N N Y N
Print data-set qualifier	TSOPRNT=aaaaaaaa	ADMPRINT	3,404,aaaa,aaaa	Y Y Y N
Short-on-storage processing	STGRET={NO|YES}	NO	3,2,{0|1}	N N Y N
SVC99 allocation size	TSOS99S=n	742710	3,410,n	Y Y Y Y
SVC99 unit specification	TSOS99U=aaaaaaaa	SYSDA	4,412,aaaa,aaaa	Y Y Y Y
Synchronized I/O	IOSYNCH={NO|YES}	NO	3,8,{0|1}	Y Y Y Y
System printer output ddname	TSOSYSP=aaaaaaaa	ADMLIST	4,405,aaaa,aaaa	Y Y Y N
Time convention	TIMEFRM={1|2|3|4}	1	3,6,{1|2|3|4}	Y Y Y Y
Trace output ddname	TSOTRCE=aaaaaaaa	ADMTRACE	4,401,aaaa,aaaa	Y Y Y N
Trace table size, in-core	TRTABLE=n	100	3,103,n	Y Y Y N
Trace control	TRCESTR='aaaaaaa'	(none)	3,114,aaaa,bbbb,..	Y Y Y Y
Trace output width	TRCEWID={SINGLE| DOUBLE}	SINGLE	3,115,{0|1}	Y Y Y Y
Trace share	TRCESHR={NO|YES}	NO	3,117,{0|1}	Y Y Y Y
Trace word value	TRACE={0|n}	0	3,102,n	Y Y Y Y
Transmission buffer size	IOBFSZ=n	1536	3,104,n	Y Y Y Y
User Control SAVE function control	CTLSAVE={YES|NO}	YES	3,119,{0|1}	Y Y Y Y
4250 code-page name	CPN4250=aaaaaaaa	AFTC0395	4,109,aaaa,aaaa	Y Y Y Y
TSO Emulation	TSOEMUL={NO|YES}	NO	3,413	Y Y Y Y

GDDM external defaults – VM/CMS

Table 19 (Page 1 of 3). GDDM defaults — options for VM/CMS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
Abend-return processing	ABNDRET={NO|YES}	NO	3,3,{0|1}	N N Y N
Alphanumeric defaults module control	DATRN=addr	ADMDATRN	3,118,addr	Y N Y Y
Always-unlock-keyboard	AUNLOCK={NO|YES}	NO	3,10,{0|1}	Y Y Y Y
APL default specification	CMSAPLF= {DATAANAL| APLTEXT}	APLTEXT	3,15,{0|1}	Y Y Y Y
Application code page	APPCPG=n	00351	3,125,n	Y Y Y Y
Call information feedback block: length: address:	CALLINF=(len,addr)	0,0 (none)	4,1101, L(CIB), A(CIB)	N N Y N
CECP keyboard input	CECPINP={YES|NO}	YES	3,126,{1|0}	Y Y N N
Comments for module identification	COMMENT= (cccccccc, cccccccc, ...)	N/A	1-8000,0,cccc,cccc ,....	Y Y Y Y
Compressed PS loads	IOCOMPR={NO|YES}	YES	3,9,{0|1}	Y Y Y Y
Date convention	DATEFRM={1|2|3|4}	4	3,5,{1|2|3|4}	Y Y Y Y
DBCS default selection	DBCSDFT= {GDDM| NO|YES}	GDDM	3,18,{0|1|2}	Y Y Y Y
DBCS SO/SI emulation character	SOSIEMC=c	"	3,110,X'xx000000'	Y Y Y Y
DBCS strings with shift-out/shift-in	MIXSOSI={NO|YES}	NO	3,17,{0|1}	Y Y Y Y
DBCS symbol set component in-core threshold	DBCSLIM=n	4	3,113,n	Y Y Y Y
DBCS symbol set language	DBCSLNG=c	K	3,111,X'xx000000'	Y Y Y Y
Deck output filetype	CMSDECK=aaaaaaaa	ADMDECK	4,503,aaaa,aaaa	Y Y Y N
Defaults file: - filename - filetype	CMSDFTS= (aaaaaaaa, bbbbbbbb)	PROFILE ADMDEFS	6,511, aaaa,aaaa, bbbb,bbbb	Y N Y N
Device attachment	AM3270=({LOCREM| REMOTE| LOCAL} ,{SNANOSNA| NONSNA| SNA})	LOCREM SNANOSNA	4,12, {0|, 1| 2} {0| 1| 2}	Y Y Y Y

Table 19 (Page 2 of 3). GDDM defaults — options for VM/CMS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values — list of full-words	Valid in: M F S C
Error exit:				
use GDDM-supplied feedback block	ERRFDBK= (GDDMDFLT)	GDDMDFLT	3,1102,0	Y N Y Y
use user-supplied feedback block	ERRFDBK= (USERAREA,addr,len)	—	5,1102,2,addr,len	Y N Y Y
Error threshold value	ERRTHRS=n	4	3,101,n	Y Y Y Y
Filetypes for:	OBJFILE= (aaaaaaaa, bbbbbbbb ,...)		4-16,107,	Y Y Y Y
Symbol sets		ADMSYMBL	aaaa,aaaa,	
Generated mapgroups		ADMGGMAP	bbbb,bbbb,	
Saved pictures		ADMSAVE	cccc,cccc,	
Chart formats		ADMCFORM	dddd,dddd,	
Chart data		ADMCDATA	eeee,eeee,	
GDDM-IMD tutorial pages		ADMTUTPG	ffff,ffff,	
GDF files		ADMGDF	gggg,gggg	
(reserved)		—	hhhh,hhhh	
Chart data definition		ADMCDEF	iiii,iiii	
Projection definition		ADMPROJ	jjjj,jjjj	
Image data		ADMIMG	kkkk,kkkk	
Force validation of HPA	FRCEVAL={NO|YES}	NO	3,127,{0|1}	N Y N N
Form feed	FF3270P=	AFTER		Y Y Y Y
	{NO|		3,11,0	
	AFTER|		3,11,1	
	BEFORE|		3,11,2	
	BOTH}		3,11,3	
FSSAVE buffer size	SAVBFSZ=n	1024	3,105,n	Y Y Y Y
High-resolution image generation;				
color filetype	CMSCOLM=aaaaaaaa	ADMCOL+	4,510,aaaa,aaaa	Y Y Y N
monochrome filetype	CMSMONO=aaaaaaaa	ADMIMAGE	4,509,aaaa,aaaa	Y Y Y N
ICU isolate value	ICUISOL={0|1|2}	0	3,112,{0|1|2}	Y Y Y Y
ICU panel color	ICUPANC= {TURQUOISE |BLUE}	TURQ	3,120,{5|1}	Y Y Y Y
ICU symbol sets	ICUFMSS={0|1|2}	0	3,122,{0|1|2}	Y Y Y Y
ICU format	ICUFMDF={0|1|2}	0	3,121,{0|1|2}	Y Y Y Y
GDDM-IMD ADS output filetype	CMSIADS=aaaaaaaa	COPY	4,506,aaaa,aaaa	Y Y N N
GDDM-IMD Export data filetype	CMSIFMT=aaaaaaaa	ADMIFMT	4,507,aaaa,aaaa	Y Y N N
GDDM-IMD MSL filetype	CMSMSLT=aaaaaaaa	ADMMSL	4,508,aaaa,aaaa	Y Y N N
Installation code page	INSCPG=n	00037	3,124,n	Y N N N
Mapgroup storage threshold	MAPGSTG=n	8192	3,106,n	Y Y Y Y
National language	NATLANG=c	A	3,4,X'xx000000'	Y Y Y N

Table 19 (Page 3 of 3). GDDM defaults — options for VM/CMS

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values – list of full-words	Valid in: M F S C
No operation	—	—	{0|13}	Y N Y Y
Number convention	NUMBFRM={1|2|3}	1	3,7,{1|2|3}	Y Y Y Y
Parameter verification (SPI)	PARMVER={NO|YES}	NO	3,1,{0|1}	N N Y N
Queued printer output filetype	CMSPRNT=aaaaaaaa	ADMPRINT	4,504,aaaa,aaaa	Y Y Y N
Short-on-storage processing	STGRET={NO|YES}	NO	3,2,{0|1}	N N Y N
System printer output filetype	CMSSYSP=aaaaaaaa	ADMLIST	4,505,aaaa,aaaa	Y Y Y N
Time convention	TIMEFRM={1|2|3|4}	1	3,6,{1|2|3|4}	Y Y Y Y
Trace output: filename filetype	CMSTRCE= (aaaaaaaa, bbbbbbbb)	ADM00001 ADMTRACE	6,502, aaaa,aaaa, bbbb,bbbb	Y Y Y N
Trace table size, in-core	TRTABLE=n	100	3,103,n	Y Y Y N
Trace control	TRCESTR='aaaaaaa'	(none)	3,114,aaaa,bbbb,....	Y Y Y Y
Trace output width	TRCEWID={SINGLE| DOUBLE}	SINGLE	3,115,{0|1}	Y Y Y Y
Trace share	TRCESHR={NO|YES}	NO	3,117,{0|1}	Y Y Y Y
Trace word value	TRACE={0|n}	0	3,102,n	Y Y Y Y
Transmission buffer size	IOBFSZ=n	1536	3,104,n	Y Y Y Y
User Control SAVE function control	CTLSAVE={YES|NO}	YES	3,119,{0|1}	Y Y Y Y
Work-file filetype	CMSTEMP=aaaaaaaa	ADMUT1	4,501,aaaa,aaaa	Y Y Y N
4250 code-page name	CPN4250=aaaaaaaa	AFTC0395	4,109,aaaa,aaaa	Y Y Y Y

GDDM external defaults – VSE/Batch

Table 20 (Page 1 of 2). GDDM defaults — options for VSE/Batch

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values - list of full-words	Valid in: M F S C
Alphanumeric defaults module control	DATRN=addr	ADMDATRN	3,118,addr	Y N Y Y
Application code page	APPCPG=n	00351	3,125,n	Y Y Y Y
Call information feedback block: length: address:	CALLINF= (len,addr)	0,0 (none)	4,1101, L(CIB), A(CIB)	N N Y N
Comments for module identification	COMMENT= (ccccccc, cccccccc ,.......)	N/A	1-8000,0,cccc, cccc,....	Y Y Y Y
Composed-page printer files for image generation; monochrome file name	VSEMONO=aaaaaaaa	ADMIMGE	3,602,aaaa,aaaa	Y Y Y Y
Composed-page printer files for image generation; color file name	VSECOLM=aaaaaaaa	ADMCOL+	3,603,aaaa,aaaa	Y Y Y Y
Date convention	DATEFRM= {1|2|3|4}	4	3,5,{1|2|3|4}	Y Y Y Y
DBCS default selection	DBCSDFT= {GDDM|NO|YES}	GDDM	3,18,{0|1|2}	Y Y Y Y
DBCS strings with shift-out/shift-in	MIXSOSI= {NO|YES}	NO	3,17,{0|1}	Y Y Y Y
DBCS symbol set component in-core threshold	DBCSLIM=n	4	3,113,n	Y Y Y Y
DBCS symbol set language	DBCSLNG=c	K	3,111,X'xx000000'	Y Y Y Y
Defaults file name	VSEDFTS=aaaaaaaa	SYSIPT	3,604,aaaa,aaaa	Y N Y N
Error exit: use GDDM-supplied feedback block	ERRFDBK= (GDDMDFLT)	GDDMDFLT	3,1102,0	Y N Y Y
Error exit: use user-supplied feed-back block	ERRFDBK= (USERAREA, addr,len	—	5,1102,2,addr,len	Y N Y Y
Error threshold value	ERRTHRS=n	4	3,101,n	Y Y Y Y
Force validation of HPA	FRCEVAL={NO|YES}	NO	3,127,{0|1}	N Y N N
FSSAVE buffer size	SAVBFSZ=n	1024	3,105,n	Y Y Y Y
ICU format	ICUFMDF={0|1|2}	0	3,121,{0|1|2}	Y Y Y Y

Table 20 (Page 2 of 2). GDDM defaults — options for VSE/Batch

Meaning of default	Source syntax of the ADMMDFT options	GDDM default	Encoded values - list of fullwords	Valid in: M F S C
Installation code page	INSCPG=n	00037	3,124,n	Y N N N
Mapgroup storage threshold	MAPGSTG=n	8192	3,106,n	Y Y Y Y
National language	NATLANG=c	A	3,4,X'xx000000'	Y Y Y N
No operation	—	—	{0|1}	Y N Y Y
Number convention	NUMBFRM={1|2|3}	1	3,7,{1|2|3}	Y Y Y Y
Parameter verification (SPI)	PARMVER={NO|YES}	NO	3,1,{0|1}	N N Y N
Short-on-storage processing	STGRET={NO|YES}	NO	3,2,{0|1}	N N Y N
Time convention	TIMEFRM={1|2|3|4}	1	3,6,{1|2|3|4}	Y Y Y Y
Trace table size, in-core	TRTABLE=n	100	3,103,n	Y Y Y N
Trace file name	VSETRCE=aaaaaaaa	ADMTRCE	3,601,aaaa,aaaa	Y Y Y N
Trace control	TRCESTR='aaaaaaa'	(none)	3,114,aaaa,bbbb,..	Y Y Y Y
Trace output width	TRCEWID={SINGLE| DOUBLE}	SINGLE	3,115,{0|1}	Y Y Y Y
Trace word value	TRACE={0|n}	0	3,102,n	Y Y Y Y
VSAM data-set names for:	OBJFILE=(aaaaaaaa, bbbbbbbb ,...)		4-16,107,	Y Y Y Y
Symbol sets		ADMF	aaaa,aaaa,	
Generated mapgroups		ADMF	bbbb,bbbb,	
Saved pictures		ADMF	cccc,cccc,	
Chart formats		ADMF	dddd,dddd,	
Chart data		ADMF	eeee,eeee,	
GDDM-IMD tutorial pages		ADMGIMP	ffff,ffff,	
GDF files		ADMF	gggg,gggg	
(reserved)		—	hhhh,hhhh	
(reserved)		—	iiii,iiii	
Projection definitions		ADMF	jjjj,jjjj	
Image data		ADMF	kkkk,kkkk	

Alphabetic list of GDDM default values

This section lists the GDDM default values in alphabetic order of the user default description parameter. For example, for the "always-unlock-keyboard" default you would look up AUNLOCK in this list.

Note: Where an operand is defined as a 4- or 8-character string, it may be specified as a shorter value, in which case the string is left-justified and padded with blanks to 4 or 8 characters.

ABNDRET = {NO|YES}
Shows whether, in a controlled abnormal-end (abend) condition, GDDM should immediately return control to the application program with a corresponding error code and message. The message includes an indication of the abend code that GDDM would otherwise have issued.

This default applies to VM/CMS only.

Note: Requesting this function causes GDDM to return only in controlled abend situations. Uncontrolled abends, such as program checks and abends issued by underlying subsystem services, cannot be returned in this manner. Also, an abend situation may be indicative of a major internal error; hence, successful return to the application cannot be ensured.

GDDM does not try to correct the abend situation or to release resources before returning to the application. Successful continuation of the GDDM session after return cannot be ensured.

AM3270 = ({LOCAL|REMOTE|LOCREM},{SNA|NONSNA|SNANOSNA})
Shows the attachment mode of 3270-family devices. All devices can be local, all remote, or a mixture of both. All can be SNA devices, all non-SNA, or a mixture of both.

This default identifies known device characteristics that GDDM may not otherwise be able to deduce, and allows GDDM to optimize its device processing.

If GDDM can deduce that all devices are locally attached, it does not usually generate "compressed PS load" data streams, even if the device shows that it supports compression and even if the IOCOMPR=YES default has been specified.

If GDDM can deduce that all devices are either locally-attached or SNA, it does not constrain "PS load" data streams to conform to the 3K transmission limit required for remote non-SNA devices.

| **APPCPG = n**
| The code-page to be used by GDDM applications. (See Figure 9 on page 124.)

AUNLOCK = {NO|YES}
Shows whether GDDM is, by default, to operate in always-unlock-keyboard mode. This is defined in the explanation of the AUNLOCK processing option in Appendix B, "Processing option groups and name-lists" on page 149.

CALLINF = (length,address)
Specifies two 4-byte fields containing the length and address of a call information feedback block provided by the application program.

The area passed by the application must be at least eight bytes long. The first four bytes receive the address of the call formats descriptor module. See Appendix H, "Call format descriptor module" on page 209. The second four bytes receive the address of the APL request code module. See Appendix I, "APL request codes module" on page 213.

If either call information module cannot be located, the 8-byte call information feedback block is set to binary zeros.

| **CECPINP = {YES|NO}**
| Specifies whether the full range of CECP code points is to be allowed in alphanumeric input data from the keyboard of a family 1 device. (See "Inhibiting input of extended code points" on page 125.)

CICAUD = (stg-addr,pgm-addr)
Specifies two 4-byte fields, each containing the address of a 4-byte anchor by which GDDM locates a record of currently acquired storage resources and currently acquired program resources, respectively. For a full explanation of this processing, see "Using the resource audit trails" on page 14.

CICDECK = aaaa
A 4-character string that is the transient data destination used by GDDM for object module output resulting from requests through the Image Symbol Editor or the GDDM-PGF Vector Symbol Editor.

CICDFPX = aaaa
A 4-character string containing the 4-byte prefix used by GDDM to determine the CICS/VS Temporary Storage names used for external defaults files. This option is intended for use in problem determination only. For the details of how to use it in that context, see the *GDDM Diagnosis and Problem Determination Guide*.

CICGIMP = aaaaaaaa
An 8-character string that is the CICS/VS File Control data-set name used by GDDM for retrieving the generated mapgroups required for the operation of GDDM-IMD.

CICIADS = aaaa
A 4-character string that is the default Transient Data destination used by GDDM for the output of ADSs (application data structures) resulting from the use of GDDM-IMD.

CICIFMT = aaaaaaaa
An 8-character string that is a default "file-type" assigned to data exported to a VSAM "staging" data set, as a result of using GDDM-IMD's Export Utility.

CICPRNT = aaaa
A 4-character string that is the transaction name assigned to the GDDM CICS/VS Print Utility; see "CICS/VS print utility" on page 48.

CICSTGF = aaaaaaaa
An 8-character string that is the default CICS/VS File Control data-set name of the VSAM "staging" data set to be used with GDDM-IMD.

CICSYSP = aaaa
A 4-character string that is the default transient data destination used by GDDM for output resulting from system printers. Such devices are defined as described in Chapter 2, "Using GDDM under CICS/VS" on page 7.

CICTIF = {NO|YES}
Shows whether GDDM is to use transaction-independent services. For a full description of this processing, see Chapter 2, "Using GDDM under CICS/VS" on page 7.

CICTQRY = aaaa
A 4-character string that is the prefix for the CICS/VS temporary storage queue names used for saving device query information.

CICTRCE = aaaa
A 4-character string that is the transient data destination used by GDDM for diagnostic trace output.

CICTSPX = aaaa
A 4-character string that is the 4-byte prefix used by GDDM to construct CICS/VS Temporary Storage names for passing data to the GDDM CICS/VS Print Utility; see "CICS/VS print utility" on page 48.

CMSAPLF = {DATAANAL|APLTEXT}
Shows the APL feature that is installed on **nonqueriable** IBM 3270 printer devices.

DATAANAL
GDDM is to assume that any APL feature installed on any printer of the above type is the Data Analysis – APL feature, unless specific application program device-definition information shows otherwise. The Data Analysis – APL feature applies to such printers as the IBM 3284, 3286, and 3288.

APLTEXT
GDDM is to assume that any APL feature installed on any printer of the above type is the APL/Text feature, unless specific application program device-definition information shows otherwise. The APL/Text feature applies to such printers as the IBM 3287 and 3289.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

CMSCOLM = aaaaaaaa
An 8-character string defining the default filetypes used by GDDM under VM/CMS for multicolored output resulting from high-resolution image devices. For details of how to define these devices, see Appendix B, "Processing option groups and name-lists" on page 149.

The character string must contain a "+" substitution character.

CMSDECK = aaaaaaaa
An 8-character string that is the filetype used by GDDM under VM/CMS for object module output resulting from requests through the Image Symbol Editor or the GDDM-PGF Vector Symbol Editor.

CMSDFTS = (aaaaaaaa,bbbbbbbb)
Two 8-character strings that are the filename and filetype of the External Defaults File under VM/CMS.

CMSIADS = aaaaaaaa
An 8-character string that is the default filetype used by GDDM under VM/CMS for the output of ADSs (application data structures) resulting from the use of GDDM-IMD.

CMSIFMT = aaaaaaaa
An 8-character string that is the default filetype used by GDDM under VM/CMS for exporting data as a result of using GDDM-IMD's Export Utility.

CMSMONO = aaaaaaaa
An 8-character string that is the default filetype used by GDDM under VM/CMS for monochrome output resulting from high-resolution image devices. For details of how to define these devices, see Appendix B, "Processing option groups and name-lists" on page 149.

CMSMSLT = aaaaaaaa
An 8-character string that is the filetype used by GDDM under VM/CMS for GDDM-IMD map specification libraries (MSLs).

CMSPRNT = aaaaaaaa
An 8-character string that is the filetype used by GDDM under VM/CMS for generating files to be printed by the GDDM VM/CMS Print Utility; see "VM/CMS print utility" on page 55.

CMSSYSP = aaaaaaaa
An 8-character string that is the default filetype used by GDDM under VM/CMS for disk file output resulting from system printer devices. For details of how to define these devices, see Appendix B, "Processing option groups and name-lists" on page 149.

CMSTEMP = aaaaaaaa
An 8-character string that is the filetype used by GDDM under VM/CMS for intermediate file operations.

CMSTRCE = (aaaaaaaa,bbbbbbbb)
Two 8-character strings that are the filename and filetype used by GDDM under VM/CMS for trace output.

COMMENT = (cccccccc,cccccccc,........)
Specifies a comment as a list of strings of 8 or less nonblank characters, which are ignored by GDDM default processing. The list must not contain more than 8000 such strings. This UDS can be used to imbed a comment into an encoded UDSL for documentation purposes.

CPN4250 = aaaaaaaa
An 8-character string that is the default code page name used for a 4250 printer. For a list of possible values, see Appendix B, "Processing option groups and name-lists" on page 149.

CTLSAVE = {YES|NO}
Shows whether GDDM is, by default, to allow the application to control the picture-saving facilities offered in the User Control environment.

The default value varies according to the subsystem:
On CICS/VS it is NO
On VM/CMS and TSO it is YES
On IMS it is not available.

DATEFRM = {1|2|3|4}
The date convention to be used by GDDM and GDDM-PGF:

1	MM/DD/YYYY	(US convention)
2	DD.MM.YYYY	(European convention)
3	YYYY-MM-DD	(ISO and Japanese convention)
4	DD MMM YYYY	(MMM are the first 3 characters of the month name).

Note that GDDM-IMD always displays the date in an abbreviated form, that is, the first two digits of the year (YYYY) are omitted.

DATRN = addr
Provides a means by which a program can pass to GDDM the address of an alphanumeric defaults module to be used instead of ADMDATRN.

DBCSDFT = {GDDM|NO|YES}
This default, which only has meaning when the NATLANG default specifies a double-byte character set (DBCS) language, introduces the concept of the default error message destination, and gives the user control over DBCS support for it. DBCSDFT allows the user to specify, or to ask GDDM to specify, whether the default error message destination can support DBCS languages. The default is that GDDM should determine this.

The values are:

GDDM	GDDM must determine whether the device can support DBCS
NO	The device cannot support DBCS
YES	The device can support DBCS.

Some examples of default error message destinations are:
- The user screen (for TSO)
- Transaction-initiating terminals (for CICS and IMS)
- FSQERR destination
- FSEXIT destination.

DBCSLIM=n
An integer, in the range 1 through 16, that is the DBCS symbol set component in-core threshold. GDDM usually optimizes DBCS symbol set functions by retaining loaded DBCS symbol set components in main storage up to the specified number of components.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

DBCSLNG=c
The language used for DBCS symbol sets.

The DBCSLNG character informs GDDM which symbol set to load to retrieve the symbol definitions. The naming convention for DBCS symbol sets is ADMxcp1p2, where:

x	I or V for mode-2 or mode-3 text respectively,
c	Language (for example, K for Kanji),
p1p2	Page (that is, the first two digits of the DBCS character).

These symbol sets are loaded as required by GDDM while processing GSCHAP, GSCHAR, or GSQTB calls. The default DBCSLIM=n specifies the limit on the number of pages that can be loaded concurrently.

In the encoded UDS format, the default value must be coded as X'xx000000', where "xx" is the hexadecimal equivalent of the character "c".

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

DFTXTNA=aaaaaaaa
The label on the first ADMMDFTX macro that defines the Job Control Language (JCL) to be used for batch printing. See the *GDDM Installation and System Management for VSE* manual for further information.

ERRFDBK=(GDDMDFLT)
Shows that the GDDM-supplied default error feedback block is used. This default can only be specified in encoded format and cannot, therefore, be specified in an ESSUDS call or in an External Defaults File.

ERRFDBK=(USERAREA,addr,len)
Shows that a user or application program-supplied error feed-back block is used. The arguments are the address and length of an error feed-back block provided by the application program. This default can only be specified in encoded format and cannot, therefore, be specified in an ESSUDS call or in an External Defaults File.

If an application program error feedback block is located in this manner, GDDM's default error exits do not send error messages to the user's terminal device. Rather, these default error exits return error details in the application program error feedback block. The format of the information returned in the feedback block is defined in the *GDDM Base Programming Reference, Volume 1*. GDDM never clears this error feedback block; it is set only as a result of a GDDM default error exit being invoked.

Note that the ERRFDBK option establishes the **default** error action. The FSEXIT(0,n) call shows that the **default** error action is to be taken. FSEXIT(addr,n) shows that the FSEXIT-defined user error exit is to be used. A subsequent FSEXIT(0,n) restores the **default** error action.

ERRTHRS=n
A nonnegative integer that is the default error threshold value. This value has the same meaning as the error severity value specified in the FSEXIT call. However, the specified threshold can have effect from the start of initialization.

The error threshold value can also be changed in the FSEXIT call.

FRCEVAL={NO|YES}
Allows the user to control the validation of high-performance alphanumerics data.

For example, when a tested application (for example, a shipped program product that does not use validation), is suspected of a bug, validation can be turned back on to determine whether the application or GDDM is at fault by specifying:

```
ADMMDFT FRCEVAL=YES
```

in the external defaults file. This default may not be specified in the external defaults module, on SPINIT calls, or by API call.

FF3270P={NO|AFTER|BEFORE|BOTH}
Shows whether GDDM, including the GDDM Print Utility, by default, performs a form feed (page eject) at the start, end, or start **and** end of processing on a 3270-family printer.

ICUFMDF={0|1|2}
Allows the user to control the use of chart format defaults in the Interactive Chart Utility of GDDM-PGF. **All** applications on the system (new, old, or standalone ICU) have their chart format defaults controlled by this one parameter. The values that can be specified are:

0 Release-dependent ICU choice.

Allows the ICU to choose the chart format defaults — the actual defaults may change from one release of GDDM to the next. This value is usually the same as choosing "2" except when the ICU is invoked by CHART with FORMNAME=★ and DISPLAY≠1 or ≠2; in this case ICUFMDF is set as if "1" had been chosen.

1 Use the chart format defaults as specified in GDDM Version 1 Release 4.

2 Use the chart format defaults as specified in GDDM Version 2 Release 1.

ICUFMSS = {0|1|2}
Specifies the default use of symbol sets in formats value in the Interactive Chart Utility of GDDM-PGF.

The values that can be specified in the defaults option are:

0	Release-dependent ICU choice (same as 2).
1	Use an asterisk (★) for all symbol sets named in format defaults.
2	Use Vector Symbol Sets as named in the format defaults.

ICUISOL = {0|1|2}
Specifies the default isolate value for the Interactive Chart Utility of GDDM-PGF. This value is inspected only if the chart-control parameter of the GDDM-PGF CHART call has the isolate value set to zero.

The values that can be specified in the defaults option are:

0	The Save, Restore, and Directory panels of the ICU are made available to the operator.
1	The Save, Restore, and Directory panels are not made available to the operator.
2	The Save and Restore panels are made available to the operator, but the Directory panel is not.

ICUPANC = {TURQUOISE|BLUE}
Specifies the default use of the basic panel color for the Interactive Chart Utility of GDDM-PGF.

The values that can be specified in the defaults option are:

TURQUOISE	The default.
BLUE	

IMSDECK = aaaaaaaa
An 8-character string that is the logical terminal name (LTERM) used by GDDM for object module output resulting from requests through the Image Symbol Editor or the GDDM-PGF Vector Symbol Editor.

IMSEXIT = aaaaaaaa
An 8-character string used as a parameter to the GDDM interactive utility transaction to cause exit processing for all conversations from a particular LTERM.

IMSICU = aaaaaaaa
An 8-character string that is the transaction name for requesting the Interactive Chart Utility of GDDM-PGF.

IMSISE = aaaaaaaa
An 8-character string that is the transaction name for requesting the Image Symbol Editor.

IMSMAST = aaaaaaaa
An 8-character string that is the LTERM name of the only LTERM allowed to issue the shutdown request to the GDDM interactive utility transaction.

IMSMODN = aaaaaaaa
An 8-character string that is the message output descriptor (MOD) name used by GDDM for sending non-conversational messages to 3270-family displays.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

IMSPRNT = aaaaaaaa
An 8-character string that is the transaction name assigned to the GDDM IMS/VS Print Utility; see "IMS/VS print utility" on page 49.

IMSSDBD = aaaaaaaa
An 8-character string that is the DBD name by which the GDDM system definition data base is accessed.

IMSSEGS = (aaaaaaaa,bbbbbbbb,cccccccc,dddddddd, eeeeeeee,ffffffff)
Six 8-character strings, which are the names of the IMS/VS segments and key fields:

aaaaaaaa	object data base root-segment name
bbbbbbbb	object data base dependent segment name
cccccccc	object data base root-segment key field name
dddddddd	object data base dependent segment key field name
eeeeeeee	system definition data base segment name
ffffffff	name of the key field in the above segment.

IMSSHUT = aaaaaaaa
An 8-character string used as a parameter to the GDDM interactive utility transaction to cause immediate termination of the transaction.

IMSSYSP = aaaaaaaa
An 8-character string that is the default destination for output from a system printer device. For details of how to define system printer devices, see Appendix B, "Processing option groups and namelists" on page 149.

IMSTRCE = aaaaaaaa
An 8-character string that is the ddname used by GDDM for trace output.

IMSUISZ = n
An integer, in the range 1 through 32000, which is the size of the data area reserved to contain the MFS Bypass input to the GDDM interactive utility transaction.

IMSUMAX = n
An integer, in the range 1 through 32765, which is the maximum number of concurrent conversations to be supported by the GDDM interactive utility transaction.

IMSVSE = aaaaaaaa
An 8-character string that is the transaction name for requesting the GDDM-PGF Vector Symbol Editor.

IMSWTOD = (n,n,n,n,....)
The descriptor codes for a write-to-operator (WTO) macro. This is used by GDDM to issue error messages if all other methods fail. For a description of valid descriptor codes, see the *OS/VS2 MVS Supervisor Services and Macro Instructions* manual.

In the encoded-UDS format, the default value should be coded as X'xxxx0000', in which bit n=1 (n=1 through 32) corresponds to descriptor code "n" being requested.

IMSWTOR = (n,n,n,n,...)
The routing codes for a write-to-operator (WTO) macro. This is used by GDDM to issue error messages if all other methods fail. For a description of valid routing codes, see the *OS/VS2 MVS Supervisor Services and Macro Instructions* manual.

In the encoded-UDS format, the default value should be coded as X'xxxx0000', in which bit n=1 (n=1 through 32) corresponds to routing code "n" being requested.

| **INSCPG = n**
| The code-page to be used by GDDM as the default for
| the installation. (See Figure 9 on page 124.)

IOBFSZ = n
An integer, in the range 1024 through 32000, which is the transmission buffer size used by GDDM for 3270-family devices. GDDM splits **outbound** terminal transmissions to fit within this buffer size. Under IMS/VS, this is the size of segments, excluding the

LLZZ prefix, that are inserted into the Message Queue.

On a non-SNA connection, for a 3179-G or 3192-G color display station, a 3270-PC/G or 3270-PC/GX work station, or a device supported by GDDM-PCLK, the outbound transmission size is restricted to approximately 3500 bytes to avoid possible controller timeouts.

Inbound transmission sizes are regulated according to the system you are using:

CICS/VS	Maximum **inbound** transmission size is regulated by CICS/VS system generation (specifically, the Terminal I/O Area lengths defined in the Terminal Control Table (TCT)), and is not affected by the value of IOBFSZ.
IMS/VS	User transactions cannot receive **input**; therefore, this field does not apply to input processing. The size of the input area allocated in the GDDM interactive utility transaction is defined in the IMSUISZ option.
TSO	The maximum **inbound** transmission size is regulated by TSO and VTAM system and network definition. Within this bound, IOBFSZ determines the size of an individual work buffer but does not otherwise affect or limit inbound transmission processing.
VM/CMS	IOBFSZ determines the **default inbound** transmission buffer size used by GDDM. GDDM acquires temporary buffers of 32000 bytes for larger inbound terminal data streams (resulting from 3270 READ MODIFIED commands).

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

IOCOMPR = {NO|YES}

Shows whether GDDM is to create compressed PS load data streams. See also the description of the AM3270 default option on page 142.

Some IBM 3270 series terminals optionally support compression of programmed symbol (PS) data streams. If such compression is to be inhibited, it is generally recommended that this be done on a specific basis through device configuration parameters. However, the IOCOMPR option can be used to inhibit compression, on a global basis, of all PS load data streams generated by GDDM.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

IOSYNCH = {NO|YES}

Shows whether GDDM is to perform synchronized terminal I/O. Usually, the use of synchronized terminal I/O implies longer transmission times and increased processing overhead. It may be useful to prevent jamming a network with large data streams used for graphics. In this context, this control might be used with a smaller value of IOBFSZ and SAVBFSZ.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

The meaning of synchronized terminal I/O differs according to the subsystem in use:

CICS/VS	Each GDDM outbound terminal transmission which expects input to be received, specifies "definite," requiring that the terminal returns a definite response, where applicable, before GDDM continues with the next transmission. Each GDDM outbound terminal transmission which does not expect input to be received, specifies "wait", requiring that the application program waits until the transmission has been completed.
TSO	Each GDDM outbound terminal transmission (using TPUT) specifies "hold," requiring that the transmission physically arrives at the terminal, where applicable, before GDDM continues with the next transmission.

MAPGSTG = n

An integer defining the mapgroup storage threshold. GDDM usually optimizes mapping functions by retaining loaded mapgroups in main storage up to the specified threshold value.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

MIXSOSI = {NO|YES}

Shows whether alphanumeric and graphic character strings may be "mixed" that is, may contain shift-out (SO) (X'0E') and shift-in (SI) (X'0F') characters to mix one-byte characters with two-byte DBCS characters.

Except on devices that support mixed alphanumeric fields (such as the IBM 5550 and 5553), alphanumeric fields that are to contain mixed strings must also be defined as "mixed" by the ASFSEN call. On devices that support mixed alphanumeric fields, it is not necessary to specify MIXSOSI=YES, unless mixed graphic character string support is also required. (See also the SOSIEMC external default on page 147.)

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

NATLANG = c

The language used by GDDM, the GDDM-PGF Interactive Chart Utility, and Presentation Graphics routines in generating messages, control-mode panels, Menu Panels, Help Panels, and generated charts. The meanings of "c" are defined as:

A	American-English
B	Brazilian
C	Simplified Chinese (People's Republic of China)
D	Danish
F	French
G	German
H	Korean (Hangeul)

I	Italian
K	Japanese (Kanji)
N	Norwegian
Q	Canadian French
S	Spanish
T	Traditional Chinese (Taiwan – Republic of China)
V	Swedish.

Languages other than American-English are supported only if the corresponding National Language Support special feature is available and installed. American-English language support is provided as part of GDDM-PGF.

In the encoded-UDS format, the default value must be coded as X'xx000000', where "xx" is the hexadecimal equivalent of the character "c".

NUMBFRM = {1|2|3}
The number representation convention to be used by GDDM and GDDM-PGF is:

1	N,NNN,NNN.MMM	(Period decimal convention)
2	N.NNN.NNN,MMM	(Comma decimal convention)
3	N NNN NNN,MMM	(French decimal convention).

OBJFILE = ([aaaaaaaa],[bbbbbbbb],...)
Up to eleven 8-character strings that show the default file-types (VM/CMS), default ddnames (TSO), default File Control data-set names (CICS/VS), or default DBD names (IMS/VS):

aaaaaaaa	symbol sets
bbbbbbbb	generated mapgroups
cccccccc	saved pictures
dddddddd	chart formats
eeeeeeee	chart data
ffffffff	GDDM-IMD tutorial pages
gggggggg	GDF files
hhhhhhhh	Reserved
iiiiiiii	Chart data definition (under TSO and VM/CMS) Reserved (under CICS/VS and IMS)
jjjjjjjj	Projection definition
kkkkkkkk	Image data.

PARMVER = {NO|YES}
Shows whether all calls through the system programmer interface should be verified for correctness of function code and number of parameters. Requesting this function incurs additional processing overheads.

SAVBFSZ = n
An integer, in the range 1024 through 32000, which is the FSSAVE transmission buffer size used by GDDM. The FSSAVE function stores preformatted data streams ready for subsequent recall and display by FSSHOW. SAVBFSZ determines the transmission buffer size used by such a saved data stream. The value of SAVBFSZ at the time of the FSSAVE call must not exceed the value of IOBFSZ at the time of the FSSHOW call.

For maximum efficiency, the FSSAVE buffer size should be chosen so that the value 4088/(FSSAVE buffer size + 5) is greater than 2 and close to an integer.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

| For 3179-G or 3192-G color display stations, 3270-PC/G or 3270-PC/GX work stations, and devices supported by GDDM-PCLK, the size saved is restricted to approximately 3500 bytes to avoid possible controller timeouts when subsequently showing the saved file.

SOSIEMC = c
Shows the character that is used as the shift-out/shift-in emulation character in mixed character strings. The character can be any keyable character that is consistent with the syntax of GDDM defaults; however, the character specified **must not then be used for any other purpose** (for example, as its own keyable value) in a mixed-string field.

The emulation character is ignored unless the MIXSOSI = YES default is specified and the device is a family-1 display other than an IBM 5550.

In the encoded-UDS format, the default value must be coded as X'xx000000', where "xx" is the hexadecimal equivalent of the character "c".

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

STGRET = {NO|YES}
Shows whether not-enough-storage or short-on-storage conditions should be processed by GDDM, and whether control should be returned immediately to the application program with a corresponding error code. Otherwise, storage requests are unconditional, with subsequent action determined by the subsystem.

Note: Requesting this function causes GDDM to issue conditional storage requests only where these are available in the subsystem. Some subsystem requests are implicitly unconditional; in these cases, subsequent action is determined by the subsystem.

TIMEFRM = {1|2|3|4}
The time convention to be used by GDDM and GDDM-PGF is:

1	HH:MM xM	(U.S. convention; XM = AM or PM)
2	HH.MM	(International convention)
3	-HH.MM.SS	(ISO convention)
4	,HH,MM,SS	(Japanese convention).

Note that GDDM-IMD always displays the time using the International convention (format 2).

TRACE = {0|n}
An integer that is the default value of the trace control word at initialization. The value may be specified either as a decimal integer or as an Assembler-language hexadecimal constant. The use of trace is described in the *GDDM Diagnosis and Problem Determination Guide*.

TRCESHR = {NO|YES}
Shows whether the trace output file is to be shared between more than one instance of GDDM. This default is only available on TSO and VM/CMS. The use of trace is described in the *GDDM Diagnosis and Problem Determination Guide*.

TRCESTR = 'aaaaaaaaaaaa'
Shows the default value of the trace control word at initialization, which is no trace. The alphanumeric string **aaaaaaaaaaaa**, which can be from 1 through 256 characters long, indicates the type of trace; the use of trace is described in the *GDDM Diagnosis and Problem Determination Guide*.

TRCEWID = {SINGLE|DOUBLE}
Shows the default value of the trace output width control word at initialization.
SINGLE
GDDM is to produce the trace output as 4-word hexadecimal.
DOUBLE
GDDM is to produce the trace output as 8-word hexadecimal, thus saving space.
The use of trace is described in the *GDDM Diagnosis and Problem Determination Guide*.

TRTABLE = n
An integer, in the range 5 through 1000, defining the number of trace entries to be held in the cyclic in-core trace table.

TSOAPLF = {DATAANAL|APLTEXT}
Shows the APL feature that is installed on **nonqueriable** IBM 3278, and 3279 Model 2 displays.

DATAANAL
GDDM is to assume that any APL feature installed on any display of the above type is the Data Analysis – APL feature, unless specific application program device-definition information shows otherwise. The Data Analysis – APL feature applies to such terminals as the IBM 3279.

APLTEXT
GDDM is to assume that any APL feature installed on any display of the above type is the APL/Text feature, unless specific application program device-definition information shows otherwise. The APL/Text feature applies to such terminals as the IBM 3278 and 3279.

The default for a specific device is established at the time of the DSOPEN call for that device. Subsequent specifications of this default in ESSUDS or ESEUDS calls do not influence the operation of a device unless it is closed (by a DSCLS call) and reopened (by a DSOPEN call).

TSOCOLM = aaaaaaaa
An 8-character string defining the default ddnames or high-level qualifiers used by GDDM for multicolored output resulting from high-resolution image devices. For details of how to define these devices, see the *GDDM Base Programming Reference, Volume 1.*

The character string must contain a "+" substitution character.

TSODECK = aaaaaaaa
An 8-character string that is the ddname used by GDDM for object module output resulting from requests through the Image Symbol Editor or the GDDM-PGF Vector Symbol Editor.

TSODFTS = aaaaaaaa
An 8-character string that is the ddname used by GDDM to access an External Defaults File.

TSOEMUL = {NO|YES}
This specifies whether, when operating in the MVS batch environment, TSO terminal I/O supervisor calls are emulated through the MVS screening facility. The emulation routines are compatible with the current version of TSO. For details of MVS SVC screening see the *OS/VS2 System Programming Library: Supervisor Manual*, and for TSO details see the OS/VS2 *TSO Guide to Writing a Terminal Monitor Program or a Command Processor*.

TSOIADS = aaaaaaaa
An 8-character string that is the default ddname used by GDDM for the output of ADSs (application data structures) resulting from the use of GDDM-IMD.

TSOIFMT = aaaaaaaa
An 8-character string that is the default ddname used by GDDM for exporting data as a result of using GDDM-IMD's Export Utility.

TSOGIMP = aaaaaaaa
An 8-character string that is the ddname used by GDDM for retrieving the generated mapgroups required for the operation of GDDM-IMD.

TSOMONO = aaaaaaaa
An 8-character string that is the default ddname or high-level qualifier used by GDDM for monochrome output resulting from high-resolution image devices. For details of how to define these devices, see the *GDDM Base Programming Reference, Volume 1.*

TSOPRNT = aaaaaaaa
An 8-character string used to generate a name of the form "aaaaaaaa.REQUEST.QUEUE" to identify the Print Utility Master Print Queue data set, where this has not otherwise been identified by DD statement. This string is also used to generate names of the form

```
[dsn-prefix.][userid.]aaaaaaaa.REQUEST.#nnnnn,
```

which are assigned to intermediate data sets required for queued printer support.

TSOSYSP = aaaaaaaa
An 8-character string that is the default ddname used by GDDM for output resulting from system printer devices. For details of how to define system printer devices, see the *GDDM Base Programming Reference, Volume 1.*

TSOS99S = n
An integer defining the size (in bytes) of the intermediate data sets that are dynamically allocated for queued printer support. The IBM-supplied default of 742710 is approximately equivalent to three 3330 cylinders.

TSOS99U = aaaaaaaa
An 8-character string defining the UNIT specification used for intermediate data sets that are dynamically allocated by GDDM in TSO Batch or MVS Batch. In foreground TSO or if the option is set to blanks (by specifying it as TSOS99U = ()), GDDM allows the UNIT specification to be defaulted from the TSO user attribute data set (UADS), where available.

TSOTRCE = aaaaaaaa
An 8-character string that is the ddname used by GDDM for trace output.

VSECOLM = aaaaaaaa
An 8-character string defining the default file name used by GDDM for multicolored output resulting from files containing graphics or images suitable for use by composed-page printers. Such printers are defined by means of the DSOPEN GDDM function described in the *GDDM Base Programming Reference*, Volume 1.

The character string must contain a "+" substitution character.

VSEDFTS = aaaaaaaa
An 8-character string, which is the file name of the external defaults file.

VSEMONO = aaaaaaaa
An 8-character string defining the default file name used by GDDM for monochrome output resulting from files containing graphics or images suitable for use by composed-page printers.

VSETRCE = aaaaaaaa
An 8-character string, which is the file name used by GDDM for trace output.

Appendix B. Processing option groups and name-lists

Processing options (procopts) allow the user to specify precisely how the input or output of a device is to be processed, with regard to the devices available, the devices' capabilities, and the subsystem under which they run.

Name-lists are a means of grouping devices according to the device family, and the subsystem under which the application is running. For information on these, see "Name-lists" on page 160.

Processing option groups: summary

Processing option groups can be specified in DSOPEN calls, see the *GDDM Base Programming Reference, Volume 1*, and in nicknames, see "Using nicknames to define device characteristics" on page 3.

The processing option groups are summarized in numeric order of option group code in Table 21.

Detailed descriptions, in numeric order of option group code, are given on pages 150 through 159.

Table 21 (Page 1 of 2). Summary of processing options and nickname keywords

Procopt group code	Nickname keyword	Arguments	Examples
1	BMSCOORD	{NO|YES}	(BMSCOORD,NO)
2	OUTONLY	{NO|YES}	(OUTONLY,NO)
3	AUNLOCK	{NO|YES}	(AUNLOCK,NO)
4	PRINTCTL	n,n,n,n,.....	(PRINTCTL,0,1,66,0,0,0,80,0)
5	CDPFTYPE	{PRIM|SEC}	(CDPFTYPE,PRIM)
6	HRISPILL	{YES|NO}	(HRISPILL,YES)
7	HRISWATH	n	(HRISWATH,10)
8	HRIPSIZE	w,d,{TENTHS|MILLS}	(HRIPSIZE,50,30,TENTHS)
9	HRIFORMT	{BITMAP|CDPF}	(HRIFORMT,BITMAP)
10	PLTFORMF	{NO|YES}	(PLTFORMF,NO)
11	PLTPENV	n	(PLTPENV,30)
12	PLTPENW	n	(PLTPENW,10)
13	PLTPENP	n	(PLTPENP,10)
14	PLTAREA	xmin,xmax,ymin,ymax	(PLTAREA,0,70,0,70)
15	PLTPAPSZ	{★|A4|A3|...|A|B|...}	(PLTPAPSZ,A4)
16	PLTROTAT	{NO|YES}	(PLTROTAT,NO)
17	SEGSTORE	{YES|NO}	(SEGSTORE,NO)
18	STAGE2ID	xxxxxxxx,xxxxxxxx,...	(STAGE2ID,★,AUX2)
19	LOADDSYM	{NO|YES}	(LOADDSYM,YES)
20	ORIGINID	{NO|YES}	(ORIGINID,YES)
21	LCLMODE	{NO|YES}	(LCLMODE,NO)
22	HRIDOCNM	xxxxxxxx	(HRIDOCNM,FIGURE9)
23	SPECDEV	{aaaaaaa|★},ddname	(SPECDEV,IBM5080)
24	WINDOW	{NO|YES}	(WINDOW,YES)
25	PSCNVCTL	{NO|START|CONTINUE}	(PSCNVCTL,START)
26	FASTUPD	{N}	(FASTUPD,0)
27	CTLFAST	{NO|YES}	(CTLFAST,YES)
28	CTLMODE	{★|YES|NO}	(CTLMODE,NO)
29	CTLKEY	{TYPE,VALUE}	(CTLKEY,1,1)
30	CTLPRINT	{YES|NO}	(CTLPRINT,NO)
31	CTLSAVE	{YES|NO}	(CTLSAVE,YES)
32	INRESRCE	{YES|NO}	(INRESRCE,YES)
33	PCLK	{YES|NO}	(PCLK,YES)
34	DEVCPG	n	(DEVCPG,00273)
35	IPDSQUAL	{*|DP|DPQ|DPT|DPTQ|NLQ}	(IPDSQUAL,NLQ)
36	PCLKEVIS	{YES|NO}	(PCLKEVIS,YES)
1000	CMSINTRP	{PA1PA2|PA2|PA1|NONE}	(CMSINTRP,PA1PA2)
1001	CMSATTN	{ASIC|EXTENDED},n,addr	(CMSATTN,BASIC,0,0)
1002	CPSPOOL	xxxxxxxx,xxxxxxxx,...	(CPSPOOL,TO,RSCS)
1003	CPTAG	xxxxxxxx,xxxxxxxx,...	(CPTAG,OUR3287,PRT,=,GRAPH)

Table 21 (Page 2 of 2). Summary of processing options and nickname keywords

Procopt group code	Nickname keyword	Arguments	Examples
1004	INVKOPUV	{NO|YES}	(INVKOPUV,YES)
2000	TSOINTRP	{PA1|NONE}	(TSOINTRP,NONE)
2001	TSORESHW	n	(TSORESHW,12)
2002	PRINTDST	{CLASS|★}, {DESTNAME|★|DDNAME}	(PRINTDST,★,★)
3000	COLORMAS	n	(COLORMAS,1)

Processing option groups: full descriptions

The processing option groups are listed here in numeric order of option group code. A full description is given of each processing option group, in this format:

- The processing option group code and nickname keyword
- A definition of the nickname syntax
- A brief description of the function of the processing option group
- The applicable subsystems
- The applicable device families
- The length of the processing option group, expressed in fullwords
- A breakdown of the function of each full-word.

The processing option groups are summarized in Table 21 on page 149.

0 Dummy
Nickname syntax: not applicable

A dummy processing option group, which is ignored. It can be used to pad processing option-lists to any desired length.

Subsystems:	All
Devices:	All
Length:	1 full-word.

1 The option group code: 0

1 Coordination mode
Nickname syntax: (BMSCOORD,{NO|YES})

Coordination mode allows a GDDM CICS/VS application program to use Basic Mapping Support (BMS) for the alphanumeric portion of the screen, and lets GDDM build and display the graphics portion. The GDDM output functions are modified so that they alter only that part of the screen covered by the graphics field and do not corrupt any data established by BMS. Coordination mode is more fully described in "Using GDDM with Basic Mapping Support" on page 13.

Subsystems:	CICS/VS
Devices:	Family 1
Length:	2 full-words.

1 The option group code: 1
2 The type of coordination:
  0 Not in coordination mode (default)
  1 In coordination mode.

2 Output-only mode
Nickname syntax: (OUTONLY,{NO|YES})

Output-only mode means that functions such as ASREAD and FSSHOW, which normally imply a wait for the operator to enter data, should instead return immediately to the application without unlocking the keyboard (unless this has been imposed by the always-unlock-keyboard mode, see option group 3). One use of this option is to allow a device to be opened so that it can display a continuous series of pictures using FSSHOW, without any operator intervention.

Subsystems:	All
Devices:	Family 1
Length:	2 full-words.

1 The option group code: 2
2 Normal or output-only mode:
  0 Not output-only mode (default)
  1 Output-only mode.

3 Always-unlock-keyboard mode
Nickname syntax: (AUNLOCK,{NO|YES})

Always-unlock-keyboard mode means that functions such as FSFRCE, which normally cause output without unlocking the keyboard, should instead unlock the keyboard, while still returning immediately to the application. This could be useful in the IMS/VS environment, to avoid the need for the operator to press RESET before being able to enter the next transaction.

| It is also useful in CICS pseudoconversational applications to cause keyboards to be unlocked on FSFRCE instead of DSCLS, which improves performance.

The default value is defined in the AUNLOCK parameter in GDDM's external defaults (see Appendix A, "GDDM's default values" on page 127), and is subsystem-dependent.

This procopt is set to the value current at DSOPEN time. It is valid from the issue of DSOPEN to the issue of DSCLS. The value cannot be altered dynamically, if a change is required, the device must be reinitialized.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	All
Devices:	Family 1
Length:	2 full-words.

1 The option group code: 3
2 The type of keyboard mode:
  0 Normal mode (default for CICS/VS, TSO, VM/CMS)
  1 Always-unlock-keyboard mode (default for IMS/VS).

4 Print control options
Nickname syntax: (PRINTCTL,n,n,n,n,....)

(where n,n,n,n,... represents the values of Fullword 3 onwards, as defined below).

This option group controls printing and copy functions. The group has this format:

Fullword 1	Option code = 4
2	No. of full-words following
3	Heading indicator
4	Number of copies
5	Page depth
6	Top margin
7	Left margin
8	Bottom margin
9	Max FSLOG characters/line
10	Alphanumeric device type

Note: This option group is of variable length and is regarded as being "mergeable" (that is, if some of the options are omitted, the current values of these options are not changed).

Subsystems:	All
Devices:	Families 1, 2, and 3
Length:	2 + N full-words.

1 The option group code: 4.
2 Number (N) of full-word values that follow (can be 0 through 8).
3 The heading indicator:
- **0** Do not print a heading page
- **1** Print a heading page (default).

4 The number of copies (applicable to family 2 only): The default is 1. If 0 is specified, 1 is assumed.
5 The page depth in rows (FSLOG and FSLOGC only):

The default is 66 or the maximum page depth for the device.

The page depth specifies the vertical size of a page of paper, fold-to-fold, in rows. If zero is specified for this parameter, a value of 66 (or the device maximum) is assumed.
6 The depth of the top margin: The default is 0.

The top and left margins (full-words 6 and 7) specify the top left-hand corner, within each page of the paper, of the printed data. Also, for FSLOG and FSLOGC purposes, a bottom margin may be specified. The total number of printed lines for each page for FSLOG and FSLOGC data is:

```
(page depth)-(top margin)-(bottom margin)
```

Note: The maximum page size for the device is taken from the device definition, as defined by the device-token parameter.
7 The width of the left margin: The default is 0.

See the description for the top margin, under Fullword 6.
8 The depth of the bottom margin (FSLOG and FSLOGC only): The default is 0.
9 Maximum number of characters per line (FSLOG and FSLOGC only): The default is 80.

Left margin + maximum number of characters per line must not exceed the maximum page width for the device.
10 Alphanumeric device type for translation: The default is 0.

For details of the values that can be specified, see the description of ASTYPE in the *GDDM Base Programming Reference, Volume 1.*

5 Output file data-stream type
Nickname syntax: (CDPFTYPE,{PRIM|SEC})

Determines whether the formatted output file is to be constructed as a primary or a secondary data stream.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	2 full-words.

1 The option group code: 5.
2 Data-stream type:
- **0** Produce a primary data stream, or document (the default)
- **1** Produce a secondary data stream, or page segment.

A primary data stream is a complete document that can be printed as it stands. A secondary data stream is one that must be imbedded in another document before it can be printed. Primary data streams can be processed by:

- IBM Print Services Facility (PSF) for printing on the 3800-3, and 3820 printers
- IBM Composed Document Print Facility (CDPF) for printing on the 4250 printer.

Note: If a 4250 output file is to contain text that refers to the 4250-printer fonts in addition to graphics picture data, it is recommended that the file be formatted as a page segment and included as part of another document.

6 Spill file usage
Nickname syntax: (HRISPILL,{YES|NO})

Determines whether a spill file is to be used while processing a high-resolution image file.

The use of a spill file reduces storage requirements at the cost of processing time. If a spill file is not used and segments are used, primitives outside segments (temporary data) do not form part of the final image, except where they occur between the last GSSCLS and ASREAD or FSFRCE calls.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	2 full-words.

1 The option group code: 6.
2 Spill file usage:
  0 Store internal picture description on disk in a spill file (the default)
  1 Store internal picture description in main storage.

7 Number of swathes
Nickname syntax: (HRISWATH,n)

Determines whether a high-resolution image is to be processed as one horizontal "swath" or many. ("Swathes" are also called slices.)

The use of swathing reduces storage requirements but at the cost of processing time.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	2 full-words.

1 The option group code: 7
2 The number of swathes to be used: The default is 1, which means generate the output image with just one pass through the internal picture description.

8 Output paper size
Nickname syntax: (HRIPSIZE,w,d,{TENTHS|MILLS})

Determines the size of the paper, as width by depth. The default size of the paper is given by the device characteristics, which are defined by the device token being used.

Note: The term "paper size" is used, although the output medium need not be paper.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	4 full-words.

1 The option group code: 8.
2 The paper width: The width, in the units defined in Fullword 4.
3 The paper depth: The depth, in the units defined in Fullword 4.
4 Units: The units used in Fullword 2 and Fullword 3.
  0 Units are tenths of an inch
  1 Units are millimeters.

9 Output file format
Nickname syntax: (HRIFORMT,{BITMAP|CDPF})

Unformatted output is a representation of the picture as one bit for each pixel. Formatted output is in a form suitable for processing either by the Print Services Facility (PSF) for 3800-3 and 3820 printers, or by the Composed Document Printing Facility (CDPF) for the 4250.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	2 full-words.

1 The option group code: 9.
2 Formatted or unformatted output:
  0 Produce unformatted output
  1 Produce formatted output (the default).

10 Plotter page feed
Nickname syntax: (PLTFORMF,{YES|NO})

Specifies whether a page feed is required after each GDDM page transmitted to the plotter by an output call such as FSFRCE. A warning message (ADM0094) is issued when the device is opened if it does not support page feed. The GDDM default action is to cause a page feed for those devices that support it.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	6182, 6186 plotters
Length:	2 full-words.

1 The option group code: 10.
2 The plotter form feed option:
  0 Page feed (default for those devices that support page feed).
  1 (NO) No page feed.
  2 (YES) Page feed.

11 Plotter pen velocity
Nickname syntax: (PLTPENV,n)

Specifies the pen velocity to be used by a plotter. The value applies to **all** the pens in the plotter. The default (0) uses the velocity set up on the plotter. It may be necessary to specify a lower value for pens used on material such as transparencies.

The recommended values are:

- On paper:

50 centimeters/second:	Fiber-tipped pens
60 centimeters/second:	Roller
15 centimeters/second:	Drafting.

- On transparencies:

10 centimeters/second:	Fiber-tipped pens.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 7371, 7372, 7374, and 7375 plotters
Length:	2 full-words.

1 The option group code: 11.
2 The pen velocity:
  0 The velocity set up by the plotter operator (the default).
  1 through 255 The velocity in centimeters per second, related to the actual velocity values available for each plotter.

  If a value greater than the maximum for the plotter is specified, the maximum velocity is set. This is:

38 centimeters/second:	For a 7371 and 7372
60 centimeters/second:	For a 7374 and 7375.

  Note: Refer to the details on the velocity select (VS) instruction in the appropriate color plotter programming manual.

12 Plotter pen width
Nickname syntax: (PLTPENW,n)

Specifies the width of the pens to be used in a plotter. Applies to **all** the pens in the plotter.

GDDM uses the pen width to determine how far apart to space lines when the plotter fills areas. If the plotter uses pens of different widths in the same picture, the pen-width value must be set to the size of the pens used for filling areas.

The pen width is used for:

- Image pixel size
- Shading line separation
- Double-width line separation
- Background line width where clipped from underlying areas.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices	Family-1 7371, 7372, 7374, 7375, and 6180 plotters
Length:	2 full-words.

1 The option group code: 12.
2 The pen width, in tenths of a millimeter:

0	Pen width of 0.3 millimeters (the default)
1 through 10	Pen width of 0.1 through 1.0 millimeters.

---

13 Plotter pen pressure
Nickname syntax: (PLTPENP,n)

Specifies how hard the plotter pen is to be pressed onto the plot bed.

The recommended values are:

- On paper:

10 grams:	Fiber-tipped pens
18 grams:	Roller
50 grams:	Drafting.

- On transparencies:

18 grams:	Fiber-tipped pens.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 7374 and 7375 plotters
Length:	2 full-words.

1 The option group code: 13.
2 The pen pressure:

0	The pressure, as set by the user on the plotter control buttons (see below).
1 through 255	The pressure, in grams, related to the actual pressure that can be set on the plotter with the control buttons.

If a value greater than the maximum for the plotter is specified, the maximum pressure is set.

If a value less than the minimum for the plotter is specified, the minimum pressure is set.

The range of values that can be set on the 7374 and 7375 plotters using the plotter control buttons is:

Button	***Pressure***
1	10 grams
2	18 grams
3	26 grams
4	34 grams
5	42 grams
6	50 grams
7	58 grams
8	66 grams.

Note: Refer to the details on the pressure select instruction in the appropriate color plotter programming manual.

---

14 Plotting area
Nickname syntax: (PLTAREA,xmin,xmax,ymin,ymax)

Specifies the area of the paper into which GDDM is to draw the picture on a plotter. If all values are specified as zero, the user defines the plotting area (before the DSOPEN call is issued) by pressing the appropriate buttons (P1, P2, and ROTATE) on the plotter, when these buttons are supported; otherwise, the maximum plotting area is used.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 7371, 7372, 7374, 7375, and 6180 plotters
Length:	5 full-words.

1 The option group code: 14.
2 The minimum x value as a percentage of the maximum paper width. The default is 0.
3 The maximum x value as a percentage of the maximum paper width. The default is 100.
4 The minimum y value as a percentage of the maximum paper height. The default is 0.
5 The maximum y value as a percentage of the maximum paper height. The default is 100.

---

15 Plotter paper size
Nickname syntax: (PLTPAPSZ,{★|A4|A3|...|A|B|...})

Specifies the size of the paper that is loaded in a plotter. Plotters that have paper-size switches must be set correctly to indicate the size of the paper loaded; otherwise, the aspect ratio might be distorted, the picture might not be placed centrally, or only part of the picture might be drawn.

If this option group is not specified, GDDM uses whatever paper size is already loaded in the plotter.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 7371, 7372, 7374, 7375, and 6180 plotters
Length:	3 full-words.

1 The option group code: 15.
2 The paper-size code:

0 or ★	The default (whatever paper size is loaded)
1	A or A4 size
2	B or A3 size
3	C or A2 size
4	D or A1 size
5	E or A0 size.

3 The dimension-type code:

0 or ★	ISO dimensions (the default)
1	ISO dimensions (A4, A3, A2, A1, or A0)
2	ANSI dimensions (A, B, C, D, or E).

---

16 Plotter picture orientation
Nickname syntax: (PLTROTAT,{NO|YES})

By default, GDDM draws the plotted picture with the x (horizontal) axis along the longest side of the paper ("landscape" format). This option group allows the picture to be drawn to be rotated by 90 degrees, so that the x axis is along the shorter side of the paper ("portrait" format). This does not affect the way in which the paper is placed in the plotter; instead, it specifies the orientation of the picture relative to the paper on the plotter bed.

GDDM ignores option group 16 when the drawing area is set by pressing buttons on the plotter (see option group 14) because this action controls the orientation of the picture.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 7371, 7372, 7374, 7375, 6180 plotters
Length:	2 full-words.

1 The option group code: 16.
2 The orientation value:
- **0** No rotation (the default)
- **1** No rotation
- **2** Rotate the picture by 90 degrees.

---

17 Retained or unretained mode
Nickname syntax: (SEGSTORE,{YES|NO})

Indicates whether a 3270-PC/G or 3270-PC/GX work station is to operate in retained or unretained mode.

Retained mode means that graphics segments are held in the display's segment buffers and are not re-sent from the host when a picture is redisplayed.

Unretained mode means that graphics segments are not held in the display's segment buffers. Segments have to be retransmitted from the host to the display whenever a picture is updated.

Even if retained mode is specified, the device may be run in unretained mode if it is customized as being in output-only mode, or if there is not enough storage available in the device, or multiple graphics fields are being displayed.

Retained mode should be the preferred mode of operation because retained segments are required to perform functions locally.

However, if an application needs more segment storage than is available in the device, this can lead to continual switching between retained and unretained modes (with undesirable performance overhead). In such cases, it may be preferable to request unretained mode, and avoid the switching between modes.

Subsystems:	All
Devices:	Family-1 3270-PC/G and /GX work stations
Length:	2 full-words.

1 The option group code: 17.
2 Retained or unretained mode:
- **0** Retained mode (the default)
- **1** Unretained mode.

---

18 Deferred device name-list for print utility
Nickname syntax: (STAGE2ID,xxxxxxxx,xxxxxxxx,...)

Specifies the name-list for the device on which the print utility is to produce the output from a print file. The list of 8-byte name-parts defined in this group is passed (in the print file) to the print utility for use as its DSOPEN name-list parameter value.

For example, if a name-list of (★,aux-id) is specified, the print utility uses this in its DSOPEN call to access the auxiliary device attached to the session device.

The default is a zero value in full-word 2. If this processing option group is not specified or if full-word 2 is zero, the file is printed on the device specified in the original DSOPEN **name-list** parameter.

Under VM/CMS, this list is ignored if the ON parameter in the ADMOPUV command is specified (ON overrides the values specified in the list).

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family 2
Length:	2+2xN full-words.

1 The option group code: 18.
2 The number (N, in the range 0 through 2) of pairs of full-words that follow.
3 through 2+2xN: "N" pairs of full-words. Each pair forms an 8-byte name-part.

---

19 Load default symbol sets
Nickname syntax: (LOADDSYM,{NO|YES})

Indicates whether the 3270-PC/G or 3270-PC/GX work station is to use the device's default symbol sets or the GDDM default symbol sets. If the application program requires any alternative characters in the symbol set (for example, national use characters), GDDM's default symbol sets must be used. For details on changing GDDM's default symbol sets, see the information in the *GDDM Installation and System Management* manual that applies to the subsystem in use.

Note: Using GDDM's symbol sets reduces the amount of storage in the work station that is available for segment storage and for symbol sets loaded by the application program.

Subsystems:	All
Devices:	Family-1 3270-PC/G and 3270-PC/GX work stations, and 3179-G and 3192-G color display stations
Length:	2 full-words.

1 The option group code: 19.
2 The default symbol sets option:
- **0** Use the work station's default mode-2 and mode-3 symbol sets (the default)
- **1** Load GDDM's mode-2 and mode-3 symbol sets, replacing the device's default symbol sets.

---

20 Origin identification
Nickname syntax: (ORIGINID,{NO|YES})

Indicates whether GDDM is to draw an origin identification string (consisting of a userid, the date, and the time) in the bottom left-hand corner of the graphics field.

For plotters, the identification appears inside a background-shaded box, so that no part of the picture can obscure it. However, if the plotting area is small, the origin identification string might be clipped and the right-hand side might be lost.

For family-1 printers, the identification is similar to an alphanumeric field. The identification is truncated, if necessary, by the page width.

When specified for a family-2 device, the processing option is passed (in the print file) to the print utility, which specifies the processing option when opening the output device.

Note: This option group is of variable length and is regarded as being "mergeable" (that is, if Fullword 3 is omitted, the current value of the option is not changed).

Subsystems:	All
Devices:	All, but used by family-1 plotters and printers and family-2 printers only
Length:	2 + N full-words.

1 The option group code: 20.
2 The number (N, in the range 0 through 1) of full-word values that follow.
3 The identification value:
 0 No origin identification (the default)
 1 Origin identification required.

21 Local interactive graphics mode
Nickname syntax: (LCLMODE,{NO|YES})

Indicates whether panning and zooming or scaling of graphics on 3270-PC/G or 3270-PC/GX work stations is to be performed using local data streams or by rebuilding the picture in the host.

Full details of how to use local interactive graphics mode are given in the *GDDM Guide for Users* manual.

Subsystems:	All
Devices:	Family-1 3270-PC/G and /GX work stations
Length:	2 full-words.

1 The option group code: 21.
2 The local interactive graphics mode option:
 0 Local interactive graphics mode not allowed (the default)
 1 Local interactive graphics mode allowed.

22 Document name
Nickname syntax: (HRIDOCNM,xxxxxxxx)

Provides a name for the document or primary data stream that is passed to CDPF. This name is printed in the picture separator-line, above each picture. This can be used to help identify the owner of the printed output.

Subsystems:	TSO, VM/CMS
Devices:	Family-4, 4250 printers only
Length:	3 full-words.

1 The option group code: 22.
2 and 3 One pair of full-words, forming an 8-byte name part.

23 Special device
Nickname syntax: (SPECDEV,{special device name|★,{{ddname},}})

Provides a token defining the type of special device and a namelist providing information specific to a specific type of special device.

Subsystems:	TSO, VM/CMS
Devices:	Family-1
Length:	2+2xN full-words.

1 The option group code: 23.
2 The number (N, in the range 0 through 2) of pairs of words that follow.
3 and 4 Special device name.
 IBM5080 To use the 5080 Graphics System for graphics
 ★ To turn off the use of the 5080.
5 and 6 Information specific to this device.

For this SPECDEV name, there are only two full-words of device-specific information, which are ddname or blank when full-words 3 and 4 contain "IBM5080".

Note: The use of a blank indicates DUM5080; that is, no actual 5080 need be attached.

24 Window mode
Nickname syntax: (WINDOW,{NO|YES})

Indicates whether the device is to be used for windowing. It allows the use of the WSCRT call to define a window on the device. Subsequent calls of DSOPEN for the same device (same device name-list) open virtual devices, which appear in the window.

The use of the WINDOW processing option inhibits the use of real partitions.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	CICS/VS, TSO, VM/CMS
Devices:	Family-1 displays, except 5080 Graphics System
Length:	2 full-words.

1 The option group code: 24.
2 The type of window mode:
 0 Not in window mode (the default)
 1 In window mode.

25 CICS pseudoconversational control
Nickname syntax:
(PSCNVCTL,{NO|START|CONTINUE})

Specifies whether GDDM is to run in conversational mode or pseudoconversational mode.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	CICS (both MVS and VSE)
Devices:	Default family-1 display device only
Length:	2 full-words.

1 The option group code: 25.
2 The use of pseudoconversational mode.
 0 Do not use pseudo- conversational mode (the default)
 1 Start use of pseudo- conversational mode
 2 Continue use of pseudo- conversational mode.

26 Fast update mode
Nickname syntax: (FASTUPD,n)

Selects the level of picture degradation that is acceptable to enable a fast update of the graphic data on the device. The option selected can subsequently be queried and changed by the application using the FSUPDM call; see the *GDDM Base Programming Reference, Volume 1*.

The main use of this processing option is to control fast update mode by means of a nickname.

It only has an effect on 3270-PC/G and 3270-PC/GX work stations, 3179-G and 3192-G color display stations, 5550-family work stations, and devices supported by GDDM-PCLK. On these devices, the color mixing can be degraded to use exclusive-OR mode to enable segments to be changed or deleted without causing a redraw of the picture.

Subsystems:	All
Devices	Family-1 3270-PC/G and 3270-PC/GX work stations, 3179-G and 3192-G displays, 5550-family work stations, and devices supported by GDDM-PCLK
Length:	2 full-words.

1 The option group code: 26.
2 The type of window mode:
 0 No degradation of picture fidelity (default)
 1 Picture degradation acceptable using GDDM's chosen method for the picture.

27 User Control fast path mode
Nickname syntax: (CTLFAST,{NO|YES})

Allows the application to select fast path mode for User Control functions that require pointings. When (CTLFAST,YES) is specified and a User Control function that requires pointing (MOVE, SIZE, POINT, CENTER, ZOOM-IN, ZOOM-OUT) is selected by a PF key, it is assumed that the user has already positioned the cursor at the first pointing.

The GDDM default is (CTLFAST,NO).

Subsystems:	Not IMS/VS
Devices:	All family-1 displays
Length:	2 full-words.

1 The option-group code: 27.
2 The availability of fast-path mode for User Control functions that require pointings:
 0 Fast path mode is not selected (the default)
 1 Fast path mode is selected.

28 User Control
Nickname syntax: (CTLMODE,{★|YES|NO})

Allows the application the overall control of the User Control environment. The GDDM default is (CTLMODE,★).

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	Not IMS/VS
Devices:	All family-1 displays
Length:	2 full-words.

1 The option-group code: 28.
2 The availability of control mode:
 0 User Control is available for devices not capable of supporting real partitions (the default).
 1 User Control is always available, forcing emulated partitions.
 2 User Control is not allowed.

29 User Control key
Nickname syntax: (CTLKEY,type,value)

Allows the application to select a User Control key that is suitable to its environment. The default is (CTLKEY,4,3), which is PA3.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	Not IMS/VS
Devices:	All family-1 displays
Length:	3 full-words.

1 The option-group code: 29.
2 The type of key selected for entering User Control:
 0 None. User Control cannot be entered by key action.
 1 A PF key (see value below) is used to enter User Control.
 4 A PA key (see value below) is used to enter User Control.
3 Value. The number of the PA or PF key used:
 0 None. User Control cannot be entered by key action.
 n The number of the PA or PF key defined for User Control.

30 User Control print
Nickname syntax: (CTLPRINT,(YES|NO))

Allows the application to control the print or plot facilities offered in User Control. The default is (CTLPRINT,YES).

Subsystems:	Not IMS/VS
Devices:	All family-1 displays
Length:	2 full-words.

1 The option-group code: 30.
2 The ability to print from the screen:
 0 (YES) Printing is allowed in User Control
 1 (NO) Printing is not allowed in User Control.

31 User Control save
Nickname syntax: (CTLSAVE,(NO|YES))

Allows the application to control the picture-saving facilities offered in the User Control environment.

The default value is defined in the CTLSAVE parameter in GDDM's external defaults (see Appendix A, "GDDM's default values" on page 127), and is subsystem-dependent.

Subsystems:	Not IMS/VS
Devices:	All family-1 displays
Length:	2 full-words.

1 The option-group code: 31.
2 the ability to save the picture:
 0 (NO) Saving is not allowed from User Control
 1 (YES) Saving is allowed from User Control.

32 Inline resources
Nickname syntax: (INRESRCE,(NO|YES))

Indicates whether the output file contains inline resources. (See "Inline resources for AFPDS printers" on page 62.)

Subsystems:	All
Devices:	All AFPDS printers
Length:	2 full-words.

1 The option-group code: 32.
2 Inline resources supported:
 0 (NO) Inline resources are not supported (the default)
 1 (YES) Inline resources supported.

33 PCLK
Nickname syntax: (PCLK,(NO|YES))

Indicates whether GDDM-PCLK is to be made available. If set to YES, users of GDDM applications on non-graphics displays, such as 3278s, will be prompted to indicate whether they want to use GDDM-PCLK.

Subsystems:	Not IMS/VS
Devices:	PCLK
Length:	2 full-words.

1 The option-group code: 33.
2 GDDM-PCLK availability:
- **0 (NO)** GDDM-PCLK not available (the default)
- **1 (YES)** GDDM-PCLK available.

34 Device code-page
Nickname syntax: (DEVCPG,n)

Specifies the code page that GDDM is to use for a device. This code-page overrides that returned by a CECP device when GDDM opens it.

Subsystems:	All
Devices:	All
Length:	2 full-words.

1 The option-group code: 34.
2 Device code-page:
- **n** The global code-page identifier (see Figure 9 on page 124).

35 IPDS printer quality
Nickname syntax:
(IPDSQUAL,{*|DP|DPQ|DPT|DPTQ|NLQ})

Indicates the print quality.

Subsystems:	Not IMS/VS
Devices:	IPDS printers
Length:	2 full-words.

1 The option-group code: 35.
2 Print quality:

0 (*)	Printer hardware setting (the default)
1 (DP or DPQ)	Data processing quality
2 (DPT or DPTQ)	Data processing text quality
3 (NLQ)	Near letter quality.

36 Encoded data fields on personal computers
Nickname syntax: (PCLKEVIS,{NO|YES})

Indicates whether the fields are to be displayed or are to be made nondisplayable.

Subsystems:	Not IMS/VS
Devices:	PCLK
Length:	2 full-words.

1 The option-group code: 36.
2 Encoded data fields to be displayed:
- **0 (NO)** Encoded data fields to be nondisplayable (the default)
- **1 (YES)** Encoded data fields to be displayed.

(PCLKEVIS,YES) must be used with GDDM-PCLK if your terminal emulator normally discards nondisplayable characters.

1000 CMS PA1/PA2 protocol
Nickname syntax:
(CMSINTRP,{PA1PA2|PA2|PA1|NONE})

Under VM/CMS, a user can usually interrupt an executing program to contact the underlying supervisors. A GDDM application can choose, by this option, whether it requires this capability. The default is to retain the capability.

Notes:

1. PA2 can only cause entry to CMS subset mode when GDDM has a read outstanding at the terminal, but not if a partition other than partition zero is active.
2. For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	VM/CMS
Devices:	Family-1 device from which the program is being run, or auxiliary device attached to that device
Length:	2 full-words.

1 The option group code: 1000.
2 The type of PA1/PA2 protocol:
- **0** PA1 causes entry to CP mode; PA2 causes entry to CMS subset mode (default)
- **1** PA1 is returned to the application; PA2 causes entry to CMS subset mode
- **2** PA1 causes entry to CP mode; PA2 is returned to the application
- **3** PA1 and PA2 are returned to the application.

1001 CMS attention handling
Nickname syntax:
(CMSATTN,{BASIC|EXTENDED},n,addr)

Determines how asynchronous interrupts (attentions) are handled in a GDDM application.

For a more detailed discussion of VM/CMS attention handling, together with a full description of the contents of the attention feedback block (see Fullword 2), see Chapter 6, "Using GDDM under VM/CMS" on page 41.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	VM/CMS
Devices:	Family-1 device from which the program is being run, or auxiliary device attached to that device
Length:	4 full-words.
	This option group always contains four full-words. (If basic attention handling is requested, the third and fourth full-words must still be present even though they are not inspected.)

1 The option group code: 1001.
2 The type of attention handling:
- **0** Basic attention handling (the default); only an unsolicited ENTER causes an attention to be raised.

GDDM passes the attention to the next higher layer in the stack of attention handlers, and takes no action on its own behalf. All other interrupts received by GDDM are ignored.

1 Extended attention handling; all unsolicited interrupts received by GDDM cause an attention to be raised.

GDDM partially decodes the inbound data stream causing the attention, and builds an **attention feedback block**. This contains the identifier of the attention in a similar format to that returned on ASREAD. After this information is filled in, control is passed to the next higher attention handler in the stack. The feedback block is not owned by GDDM, but is supplied by the user by this option group. If, however, either the length or the address of the block is zero, the feedback block is not filled in.

3 The length of the attention feedback block. See the description of extended attention handling, above (Fullword 2).

4 The address of the attention feedback block. See the description of extended attention handling, above (Fullword 2).

1002 CMS CP SPOOL parameters
Nickname syntax: (CPSPOOL,xxxxxxxx,xxxxxxxx,.....)

Causes a CP SPOOL command to be issued for punch files that result from opening a family-1 device with a name-list of "PUNCH" under VM/CMS. If specified, this option group causes a CP SPOOL command of this form:

```
CP SPOOL PUNCH xxxxxxxx xxxxxxxx ........ ........
```

to be issued at the time of the DSOPEN call.

A specification of the form (CPSPOOL,TO,RSCS) can be used to direct such punch files to a product capable of processing them (such as RSCS Networking Version 2).

GDDM does not restore any previous spooling control options when the device is closed. The default is a zero value in full-word 2. If this processing option group is not specified or if full-word 2 is zero, no CP SPOOL command is issued.

Subsystems:	VM/CMS
Devices:	Family-1 device 'PUNCH'
Length:	2+2xN full-words.

1 The option group code: 1002.

2 The number (N, in the range 0 through 16) of pairs of full-words that follow.

3 through 2+2xN: "N" pairs of full-words, giving the appropriate spooling information as 8-character tokens.

1003 CMS CP TAG parameters
Nickname syntax: (CPTAG,xxxxxxxx,xxxxxxxx,.....)

Causes a CP TAG command to be issued for punch files that result from opening a family-1 device with a name-list of "PUNCH" under VM/CMS. If specified, this option group causes a CP TAG command of this form:

```
CP TAG DEV PUNCH xxxxxxxx xxxxxxxx ........ ........
```

to be issued at the time of the DSOPEN call.

GDDM inserts one blank character between each specified token, except that GDDM removes any excessive blank characters and any blank characters surrounding the character "=". Thus, a specification of the form:

```
(CPTAG,PRINTER1,PRT,=,GRAPH)
```

causes the following CP TAG command to be issued:

```
CP TAG DEV PUNCH PRINTER1 PRT=GRAPH
```

A specification like the one above can be used to notify products capable of processing punch files (such as RSCS Networking Version 2) about the graphic nature of the punch file.

GDDM does not restore any previous tag information when the device is closed. The default is a zero value in full-word 2. If this processing option group is not specified or if full-word 2 is zero, no CP TAG command is issued.

Subsystems:	VM/CMS
Devices:	Family-1 device 'PUNCH'
Length:	2+2xN full-words.

1 The option group code: 1003.

2 The number (N, in the range 0 through 16) of pairs of full-words that follow.

3 through 2+2xN: "N" pairs of full-words, giving the appropriate routing (tag) information as 8-character tokens.

1004 Automatic invocation of VM/CMS print utility
Nickname syntax: (INVKOPUV,{NO|YES})

Indicates whether GDDM is to invoke the GDDM print utility automatically after a print file has been created.

If this function is requested, a temporary print file is created, and the print utility is requested to print this file on the device specified by the **name-list** parameter. After printing, the temporary file is erased.

Subsystems:	VM/CMS
Devices:	Family 2
Length:	2 full-words.

1 The option group code: 1004.

2 Print utility control:
- **0** Do not invoke print utility
- **1** Invoke print utility automatically.

2000 TSO CLEAR/PA1 protocol
Nickname syntax: (TSOINTRP,{PA1|NONE})

Usually, in TSO, an end user can interrupt an executing program to contact the underlying supervisor. A GDDM application can choose, by this option, whether it requires this capability.

For a more detailed discussion of the use of the PA1 and CLEAR keys in an TSO environment, see Chapter 5, "Using GDDM under TSO" on page 33.

Subsystems:	TSO
Devices:	Family 1
Length:	2 full-words.

1 The option group code: 2000.

2 The type of attention handling:
- **0** PA1 causes attention, CLEAR is ignored (TSO default action)
- **1** PA1 and CLEAR are returned to the GDDM application (PA1 does not cause an attention).

2001 TSO reshow protocol
Nickname syntax: (TSORESHW,n)

This option controls which PF and PA key functions are passed to the GDDM application program on input. The key functions specified in this option are **not** to be passed. They are treated as messages from TSO, informing GDDM that the display was corrupted.

Any key functions specified in this option are not available to the application program. When pressed by the terminal user, the specified keys cause the current picture to be rebuilt and reshown.

This option group allows an application, executing in a TSO/VTAM environment, to alter the Attention Identifier (AID) that signals that the display was corrupted (typically, by line-by-line output). It can be set to be either the default PA key or a PF key. Changing it to a PF key releases the default PA key for other use.

Note: For a GDDM program running under the control of a task manager, if this processing option is specified for a virtual device, it is ignored, and the processing option for the real device is used instead.

Subsystems:	TSO
Devices:	Family 1
Length:	2 full-words.

1 The option group code: 2001.
2 The keys treated as "reshow" AIDs:

0	PA2 is treated as the "reshow" AID (the default)
1 through 24	The number of the PF key to be treated as the "reshow" AID.

2002 TSO family-2 print-file destination
Nickname syntax:
(PRINTDST,{class|★}{,destname|★|ddname})

This option controls the destination of the family-2 print output.

The default destination is the ADMPRNT queue.

Subsystems:	TSO (including TSO/BATCH and MVS/BATCH)
Devices:	Family 2
Length:	2+2xN full-words.

1 The option group code: 2002.
2 The number (N, in the range 1 through 2) of pairs of full-words that follow.
3 and 4 An 8-character token containing one of:

class	Appropriate output class for the JES spool system.
★	Output is to go to ADMPRINT queue or a ddname.

5 and 6 An 8-character token containing one of:

destname	The JES Remote Work Station name, associated through JES/328X, with the required target printer.
★	Output is to go to the ADMPRINT queue.
ddname	The ddname of a DD statement describing the output data set to be used.

3000 Color-master table identifier
Nickname syntax: (COLORMAS,n)

Identifies the color-master table to be used.

A color-master table defines how each input color is to be analyzed into one or more color masters. If this option group is not specified, a single monochrome master is generated.

Subsystems:	TSO, VM/CMS
Devices:	Family 4
Length:	2 full-words.

1 The option group code: 3000.
2 The identifier of the color master table: A number that is placed after the letters "ADM" to create a color table name. For example, the number 1 results in color table ADM00001 being used. Specifying 0 (the default) means that a monochrome master is generated.

For more information on color separations, see the *GDDM Application Programming Guide, Volume 1*.

For information on the ADMMCOLT macro, see also Chapter 10, "Setting up color-master tables" on page 79.

Name-lists

The following section describes the **name-list** values that can be specified for each subsystem and for each GDDM device family.

A **name-list** is a means of identifying which physical device is to be opened for use by a GDDM application program. It can be a parameter of the DSOPEN call (see the *GDDM Base Programming Reference, Volume 1*), or it can be specified as a nickname. The naming convention of the **name-list** varies according to the subsystem and device family in use.

Reserved names "★" and blanks

In all environments, for all families, there is a convention for two reserved values of the name-list(1) field.

- When this field is specified but is "★", the terminal used is as described under the options below for a name-count of 0, where this is valid. In other words, this is an explicit way to specify the default device name.
- When the field contains blanks, the device is a **dummy** one, that is, no real device is associated with this GDDM device. GDDM generates the data streams required but does not send them to any real device, nor does it try to receive data from a device.

  This option can be used to check a GDDM application when a real device with the necessary features is unavailable, or it can be used with the FSSAVE mechanism to generate SAVE files for a device that is unavailable when the application is to be run.

  When this option is selected, the application program must provide a device token parameter to supply the device characteristics that are to be used by GDDM.

Family-1 name-list

In all subsystems, the device name can specify the user console:

- By omitting the name list (by giving a length of 0 in DSOPEN)
- By setting all name-parts to "★".

Also, (under CICS/VS, TSO, or VM/CMS), the name-list parameter can identify an auxiliary device, such as a plotter that is attached to a 3270-PC/G or 3270-PC/GX work station, or a printer or plotter that is attached to a GDDM-PCLK work station. In such a case, **name-list(1)** identifies the 3270-PC/G or 3270-PC/GX, or GDDM-PCLK work station, and **name-list(2)** (other than "★") identifies the auxiliary device (the plotter or printer). GDDM uses this name to identify the appropriate port on the attaching work station.

Notes:

1. The name given in ***name-list(2)*** must be the same as the name given in the IEEE customization panel when the 3270-PC/G or 3270-PC/GX work station was set up. (This is not the same as the device type which must be of the form "IBMnnnn".)
2. A ***name-list(2)*** value of "ADMPLOT" has a special meaning. In this case, GDDM uses the first plotter defined in the IEEE customization panel when the 3270-PC/G or 3270-PC/GX work station was set up, regardless of the configured name.
3. In the case of GDDM-PCLK 1.1 only one plotter can be configured, so ADMPLOT should always be used. The special value ADMPCPRT should be used to open a PCLK-attached printer, see *GDDM-PCLK Guide*.

CICS/VS name-list

Family 1 – 3270 terminals
The name-count value must be 0, 1, or 2:

0	The device used is that identified by the Terminal Control Table (TCT) for the transaction.
1	Name-list(1) must contain either "★" or blanks. If it contains "★", the terminal is used as described for a name-count of 0.
2	Name-list(1) must contain either "★" or blanks.

If name-list(2) contains "★", the terminal is used as described for a name-count of 1. Otherwise, the name-list(2) value is the name of an auxiliary device (a plotter).

Family 2 – queued printer
The name-count value must be 1.

The name-list(1) value is the terminal identifier of the printer in the TCT.

Family 3 – system printer
The name-count value must be either 0 or 1:

0	A name is taken from the GDDM defaults. The supplied default is ADMS.
1	A name is taken from name-list(1).

The name is assumed to be the name of a transient data destination that can route the output to a subsystem printer. The transient data destination should be one defined in the CICS/VS Destination Control Table.

When name-list(1) contains "★", the printer is used as described for a name-count of 0.

Family 4 – composed-page printer files
Not applicable under CICS/VS.

IMS/VS name-list

Family 1 – 3270 terminals
The name-count value must be either 0 or 1:

0	An LTERM name is taken from the LTERM field of the I/O PCB.
1	An LTERM name is taken from name-list(1).

There must be at least one TP PCB whose destination is set to the LTERM name.

If name-list(1) contains "★", the terminal is used as described for a name-count of 0.

Family 2 – queued printers
The name-count value must be 1.

The name-list(1) value is an LTERM name. This LTERM must be for a 3270-family printer. There must be at least one TP PCB whose destination is set to the name of the GDDM-supplied print utility transaction.

Family 3 – system printers
The name-count value must be either 0 or 1:

0 An LTERM name is taken from the GDDM defaults. The supplied default is ADMLIST.
1 An LTERM name is taken from name-list(1).

There must be at least one TP PCB whose destination is set to the LTERM name. This LTERM must be for a SPOOL printer.

If name-list contains "★", the printer is used as described for a name-count of 0.

Family 4 – high-resolution image files
Not applicable under IMS/VS.

TSO name-list

Family 1 – 3270 terminals
The name-count value must be 0, 1, or 2:

0 The device is the terminal from which the application is being run.
1 Name-list(1) must contain either "★" or blanks. If it contains "★", the terminal is used as described for a name-count of 0.
2 Name-list(1) must contain either "★" or blanks.

If name-list(2) contains "★", the terminal is used as described for a name-count of 1. Otherwise, the name-list(2) value is the name of an auxiliary device (a plotter).

Family 2 – queued printers
The name-count value must be 1.

The name-list(1) value is the device identifier of the printer. This device identifier must be one of the names in the Master Print Queue data set of the GDDM print utility. Under VTAM, the device identifier must be included in SYS1.VTAMLIST.

Family 3 – system printers
The name-count value must be either 0 or 1:

0 A ddname for a SYSOUT file is taken from the GDDM defaults. The supplied default is ADMLIST.
1 A ddname for a SYSOUT file is taken from name-list(1).

If name-list(1) contains "★", the printer is used as described for a name-count of 0.

Family 4 – high-resolution image files
The name-count value must be 1 through 6.

The name-list value defines the DDNAME(s) or DSNAME(s) of the data set(s) that will be generated. More than one data set is generated if a color master table is being used (as specified by processing option group 3000).

Monochrome master

The name-list value must be of one of these:

★ A name is taken from the GDDM defaults. The supplied default is ADMIMAGE.

The inferred name is searched for as a DDNAME. If it cannot be found as a DDNAME, it is formed into a DSNAME of the form "qualifier(s).name" (where qualifier(s) is the active dsn-prefix, or userid, or both of these).

name1.name2[.name3....] or 'name1[.name2.name3....]'

The specified name is taken as a DSNAME, according to TSO naming conventions. Unless contained in quotes, the specified name must contain one (and only one) component of "★". Whether contained in quotes or not, if any one component of the name is "★", that component is replaced with a value taken from the GDDM defaults. The supplied default is ADMIMAGE.

If contained in quotes, the name is taken as a complete DSNAME. If not contained in quotes, it is formed into a complete DSNAME of the form:

```
'qualifier(s).namel.name2...'
```

where qualifier(s) is the active dsn-prefix, or userid, or both of these.

If the specified name exceeds 8 characters in length, it must be placed in consecutive members of the array, and, if necessary, padded with blanks.

For example, if the DSNAME is contained in quotes and is

```
aaaa.bbbb.ccc
```

then it would look like this:

namelist(1) =	'	a	a	a	a	.	b	b
namelist(2) =	b	b	.	c	c	c	'	

In PL/I, a string can be overlaid on the array to simplify this (but the name-count must still specify the number of 8-byte tokens).

Color masters

The name-list value must be of one of these:

★ A value is taken from the GDDM defaults. The supplied default is ADMCOL+. The "+" is replaced by 1, 2, 3, and so on (up to a maximum of 9) for each color master data set.

The first derived name (for example, ADMCOL1), is searched for as a ddname. If it is found as a ddname, all the other derived names must also exist as ddnames. If it cannot be found as a ddname, all the derived names are formed into DSNAMEs of the form:

```
'qualifier(s).name'
```

where qualifier(s) is the active dsn-prefix, or userid, or both of these.

name1.name2[.name3....] or 'name1[.name2.name3....]'

The specified name is taken to identify DSNAMEs, according to TSO naming conventions. The specified name must contain one (and only one) component of "★". That component is replaced with a value taken from the GDDM defaults. The supplied default is ADMCOL+. The "+" is replaced by 1, 2, 3, and so on, (up to a maximum of 9) for each color master data set.

If contained in quotes, the derived names are taken as complete DSNAMEs. If not contained in quotes, they are formed into complete DSNAMEs of the form "qualifier(s).name1.name2..." (where qualifier(s) is the active dsn-prefix, or userid, or both of these).

If the specified name exceeds 8 characters in length, it must be placed in consecutive members of the array, and, if necessary, padded with blanks.

For example, if the DSNAME is contained in quotes and is

`aaaa.*.ccc`

where "★" is replaced by ADMCOL1, ADMCOL2, and so on, then it would look like this:

namelist(1) =	'	a	a	a	a	.	*	.
namelist(2) =	c	c	c	.				

In PL/I, a string can be overlaid on the array to simplify this (but the name-count must still specify the number of 8-byte tokens).

In this example, the derived DSNAMEs when using a color table specifying four-color masters would be:

```
'aaaa.ADMCOL1.ccc'
'aaaa.ADMCOL2.ccc'
'aaaa.ADMCOL3.ccc'
'aaaa.ADMCOL4.ccc'
```

VM/CMS name-list

Family 1 – 3270 terminals
The name-count value must be 0, 1, or 2:

0 The device is the terminal from which the application is being run.

1 Name-list(1) must contain one of these:
- "★"
- Blanks
- "PUNCH" A character form of device address (for example "061").

If name-list(1) contains "★", the terminal is used as described for a name-count of 0.

If name-list(1) = "PUNCH", GDDM writes the 3270 device output to the CMS virtual punch, in the form described in "Native CMS file processing" on page 42. In this case, the application must provide a device token parameter to supply the device characteristics that are to be used by GDDM.

2 Name-list(1) must contain one of these:
- "★"
- Blanks
- "PUNCH"
- A character form of device address (for example "061").

If name-list(2) contains "★", the terminal is used as described for a name-count of 1. Otherwise, the name-list(2) value is the name of an auxiliary device (a plotter).

Family 2 – queued printers
The name-count value must be 1 through 3.

Unless processing option group 1004 (INVKOPUV) is specified, the name-list(1), name-list(2), and name-list(3) values define the filename, filetype, and filemode (respectively) of the print file that is to be generated. The supplied default for filetype is ADMPRINT. Filemode defaults to A1.

If automatic invocation of the VM/CMS Print Utility is requested (as specified by INVKOPUV), name-count and name-list identify a family-1 device, and must therefore be as defined for family-1 (above).

Family 3 – system printers
The name-count value must be 0, 1, 2, or 3:

0	The device is the currently-defined printer; that is, device 00E.
1 through 3	Name-list(1), name-list(2), and name-list(3) define the filename, filetype, and filemode (respectively) of the print file that is to be generated. The supplied default for filetype is ADMLIST. Filemode defaults to A1.

When name-list(1) contains "★", the printer is used as described for a name-count of 0.

Family 4 – high-resolution image files
The name-count value must be 1 through 3.

The name-list(1), name-list(2), name-list(3) values define the filename, filetype, and filemode, respectively, of the CMS file(s) that is generated. More than one file is generated if a color master table is being used (as specified by processing option group 3000).

For both monochrome and multicolor masters, "A1" is assumed if the filemode is omitted.

Monochrome master
When the filetype is omitted or is specified as "★", the filetype is taken from the GDDM defaults. The supplied default is ADMIMAGE.

Color masters
When the filetype is specified, it must be "★". The filetype is taken from the GDDM defaults. The supplied default is ADMCOL+. The "+" is replaced by 1, 2, 3, and so on (up to a maximum of 9) for each color master file.

For example, if:

namelist(1) =	a	a	a	a				
namelist(2) =	*							

the derived file identifiers when using a color table specifying four-color masters would be:

```
aaaa ADMCOL1 A1
aaaa ADMCOL2 A1
aaaa ADMCOL3 A1
aaaa ADMCOL4 A1
```

Appendix C. GDDM object file formats

Version 2 Release 2 of GDDM supports the following object names and types:

Table 22. GDDM object names and types

Object name	Object type
ADMSYMBL	Symbol set
ADMGGMAP	Generated GDDM mapgroup
ADMSAVE	FSSAVE file
ADMCFORM	Chart format file
ADMCDATA	Chart data file
ADMGDF	GDF file
ADMCDEF	Chart definition file
ADMPROJ	Projection definition file
ADMIMG	Image data file

Record structure

Every record in a stored GDDM object is 400 bytes long. The first 20 bytes of each record comprise a record identification field, which is used as a source key when the object is stored on a keyed database (as in CICS/VS or IMS/VS). The remaining 380 bytes of the first record provides more information on the object, and the remaining 380 bytes of subsequent records comprise the object data.

Table 23. GDDM stored object file format

Length (bytes)	Content	Record type
20	Identification field	Header record
380	Information field	
20	Identification field	Data record 1
380	Data field	
⋮		
20	Identification field	Data record n
380	Data field	

The header record

The first record in a GDDM stored object is a header record containing miscellaneous control and comment information. It consists of two fields; the record identification field, and the record information field.

The record identification field

The first eight bytes of this field contain the name of the object. This name is the same in all the records of the object.

The second eight bytes of this field contain the object type (see Table 22), and is also the same for all the records in the object.

The last four bytes of the record identification field contain the record sequence number, starting at 1, in fixed binary form.

Table 24. GDDM stored object — record identification field format

Offset	Length	Type	Content
0	8	CHAR(8)	Object name
8	8	CHAR(8)	Object type
16	4	FIXED(31)	Record sequence number

The information field

The remaining 380 bytes of the header record provide extra information about the record, such as the GDDM version and release number, and the date and time the record was encoded. The format is:

Table 25 (Page 1 of 2). GDDM stored object — record information field format

Offset	Length	Type	Content
20	4	FIXED(31)	GDDM object V1R1 – length of object V1R2 – X'00000010' V1R3 and later – X'00000000'
24	4	CHAR(4)	Reserved
28	4	CHAR(4)	GDDM Version and release (for example: '1030' for Version 1, Release 3.0)
32	4	FIXED(31)	Object major type (same as type in record identification field)
36	4	FIXED(31)	Object minor type (for Image symbol sets – 1 for Vector symbol sets – 2 otherwise – 0)
40	4	FIXED(31)	Length of supplied user comments
44	8	CHAR(8)	Date and time stored (encoded) – date (00YYDDD+ format) – time (0HHMMSS+ format)

Table 25 (Page 2 of 2). GDDM stored object — record information field format

Offset	Length	Type	Content
52	20	CHAR(20)	Date and time stored (EBCDIC)
72	8	CHAR(8)	Reserved (must be all X'00')
80	255	CHAR(255)	Up to 255 bytes of user comments
335	63	CHAR(63)	Reserved
398	2	FIXED(16)	Code page identifier

The data record

The second and subsequent records in a GDDM stored object contain the object data. The first 20 bytes of these records constitute record identification fields, as defined in Table 22 on page 163, leaving 380 bytes for the data proper.

For objects generated by GDDM Version 1 Release 2 or later, the remaining 380 bytes of each record contain one or more data blocks.

Each data block contains a two-byte length field followed by up to 32000 data bytes as defined for the particular type of object.

For example, a symbol-set object contains just one data block starting with a two-byte length field, followed by data bytes as defined in Appendix F, "Symbol-set formats" on page 199. These data bytes themselves start with two-byte length fields.

Each record may contain one or more data blocks concatenated together, and if necessary, data blocks are spanned across records — although the two-byte length field at the start of a data block is never split across records. Unused space at the end of any record consists of X'00'.

Appendix D. GDF order descriptions

Graphics data format (GDF) is a means of storing pictures. GDDM uses it internally, and also makes it available to application programs. It consists of a set of orders with similar meanings to the GDDM graphics call statements. In many cases there is a one-for-one mapping between GDF orders and GDDM call statements.

GDDM supports a picture prolog that contains information about the size of the picture and the symbol sets used in the picture. A detailed description of the orders that relate to the picture prolog is given under "Picture prolog" on page 183. The information the picture prolog provides is:

- The coordinate type
- The picture boundary
- The picture scale and aspect ratio
- The symbol sets that are referenced
- The drawing defaults information.

The initial Comment order in the generated GDF is retained for compatibility with previous releases of GDDM.

Compatibility

GDDM ensures upward compatibility of GDF orders from previous releases to the current release. The orders are **not** downward-compatible from the current release to previous releases.

Saving GDF orders

Applications can save GDF orders for later use as follows:

- As application-written GDF files (GDDM Version 1 Release 2 onwards)

  Use GSGET to move GDF orders from GDDM into application-program storage. The application program can then write these to auxiliary storage.

- As GDDM-written ADMGDF objects produced from Version 1 Release 4 onwards.

  Use GSSAVE to save GDF orders as a specially formatted ADMGDF object on auxiliary storage. This object contains the name of the file "ADMGDF" in columns 9 through 14 of each record. The ADMGDF objects can be processed by a GDDM application using GSLOAD.

GDF can be retrieved in two formats, fixed or floating point. Floating-point GDF corresponds as closely as possible to the GDDM calls used to generate the picture. The data primitives will have been clipped, only if the application requested clipping using a GSCLP call statement. Fixed-point GDF does not necessarily match the original commands (the data is always clipped).

The GDF data that results may not necessarily resemble the original commands used to generate the picture because these have been processed to suit the primary device in use. For example, coordinates will have been converted to an internal coordinate system with some loss of precision. Complex primitives (such as curved fillets) may have been simplified and approximated. Clipping may have resulted in alterations to the primitives supplied. The data is thus not a substitute for the original. It can, however, be useful in producing an approximate copy of the stored data on another device.

The GDF file conversion utility can also be used to convert the file from the first format to the second.

Figure 10 shows the flow of events:

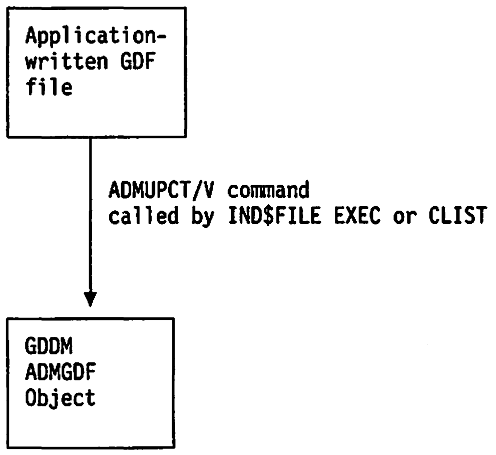

Application-written GDF file
ADMUPCT/V command called by IND$FILE EXEC or CLIST
GDDM ADMGDF Object

Figure 10. GDF file conversion – format 1 to format 2

To convert an application-written GDF file into an ADMGDF object the command is:

Under TSO:

```
ALLOC F(ADMPIF) DA('pif-dataset-name')SHR
ALLOC F(ADMGDF) DA('admgdf-dataset-name')SHR
CALL 'GDDM.OSPID.GDDMLOAD(ADMUPCT)' 'pif-member /
      (PUT admgdf-member options'
```

Where admgdf-dataset-name must exist, and must be partitioned. The data set usually has the attributes LRECL(400) and RECFM(F) but these can be altered.

If pif-dataset-name is sequential, pifmember should be omitted.

Notes:

1. The user must allocate two ddnames:
   - ADMPIF for the data set containing the application-written GDF file (partitioned or sequential)
   - ADMGDF for the partitioned data set to contain the ADMGDF object.
2. The program expects GDDM Version 1 Release 3 or Version 1 Release 2 data sets by default to be LRECL=400 and RECFM=F, but these defaults can be changed.

Under CMS:

```
ADMUPCV gdf-file-id (PUT admgdf-name options
```

Note: The `gdf-file-id` is a standard CMS file identifier.

The options are:

- {*NEWFile*|REPlace} – creates a new ADMGDF object or replaces an existing object of the same name
- {*FIXed*|FLOAT} – creates the ADMGDF file in fixed-point or floating-point format.

Format of GDF objects

The format of the data returned by GSGET is:

```
Comment order, with coordinate information
Begin Symbol-Set Mapping PSC
    Map Symbol-Set Identifier PSC
    :
    :
End Symbol-Set Mapping PSC
Begin Picture Prolog PSC
    Set Drawing Default PSC
    :
    :
End Picture Prolog PSC
Picture GDF (contains GDF orders)
```

The information in this appendix will help to interpret GDDM-created GDF orders that are to be used outside GDDM, or to create new GDF orders that can subsequently be used within GDDM.

For an example program that shows how to handle GDF data, see the *GDDM Application Programming Guide, Volume 1.*

Coordinates and aspect ratio

The coordinate values in the Picture Boundary PSC order and the initial Comment order are the upper and lower bounds of the picture space. In fixed-point GDF, these values are the values suitable for the device. In floating-point GDF, they are the continuation of the current window bounds to the picture space boundary. Note that unclipped floating-point GDF can contain orders with coordinates that are outside these limits.

To reshow a GDF picture, the window coordinates should be reset to the picture boundary values. The GDF picture can be reshown at any size. To preserve the aspect ratio of the picture, a GSPS call is required that is based on the coordinate values in the Picture Scale PSC order. This order defines the aspect ratio of the coordinates; the default aspect ratio is 1.

GDF orders: summary

Alphabetic list

Table 26 shows the GDF orders in alphabetic order as they are described in this appendix. It provides useful information for those who need to write the orders.

Table 27 on page 168 shows the GDF orders in the order of their code values and it provides useful information for those who need to interpret the orders.

Table 26 (Page 1 of 2). Alphabetic summary of GDF orders

	Order code and usage				
Order name	**Primitives**	**Primitives at current position**	**Set**	**Push and set**	**Others**
Arc	X'C6'	X'86'			
Arc Parameters			X'22'	X'62'	
Area					X'68'
Background Color Mix			X'0D'	X'4D'	
Call Segment					X'07'
Character Angle			X'34'	X'74'	
Character Box			X'33'	X'03'	
Character-Box Spacing			X'36'	X'76'	
Character Direction			X'3A'	X'7A'	
Character Precision			X'39'	X'79'	
Character Set			X'38'	X'78'	
Character Shear			X'35'	X'75'	
Character String	X'C3'	X'83'			
Color			X'0A'	X'4A'	
Comment					X'01'
Current Position			X'21'	X'61'	
End Area					X'60'
Extended Color			X'26'	X'66'	
Fillet	X'C5'	X'85'			
Foreground Color Mix			X'0C'	X'4C'	
Fractional Line Width			X'11'	X'51'	
Full Arc	X'C7'	X'87'			
Image Begin	X'D1'	X'91'			
Image Data					X'92'
Image End					X'93'
Line	X'C1'	X'81'			
Line Type			X'18'	X'58'	
Line Width			X'19'	X'59'	
Marker	X'C2'	X'82'			
Marker Box			X'37'	X'77'	
Marker Scale			X'41'		
Marker Type			X'29'	X'69'	
Model Transform			X'24'	X'64'	
Pattern			X'28'	X'09'	
Pick (Tag) Identifier			X'43'	X'23'	
Pop					X'3F'
Process Specific Control					X'02'
Relative Line	X'E1'	X'A1'			
Segment Attribute					X'72'
Segment Attribute Modify					X'73'
Segment Characteristics					X'04'
Segment End					X'71'
Segment End Prolog					X'3E'

Table 26 (Page 2 of 2). Alphabetic summary of GDF orders

	Order code and usage				
Order name	**Primitives**	**Primitives at current position**	**Set**	**Push and set**	**Others**
Segment Position					X'53'
Segment Start					X'70'
Set Viewing Window			X'27'		
Text Alignment			X'10'	X'50'	

Code value list

Table 27 shows the GDF orders in the order of their code values and it provides useful information for those who need to interpret the orders.

Table 26 on page 167 shows the GDF orders in alphabetic order as they are described in this appendix. It provides useful information for those who need to write the orders.

Table 27 (Page 1 of 2). Summary of GDF orders in order of code values

Code	Name of GDF order	Mnemonic
X'01'	Comment	GCOMT
X'02'	Process Specific Control	GPSC
X'03'	Push And Set Character Box	GPSCC
X'04'	Segment Characteristics	GSGCH
X'07'	Call Segment	GSCALL
X'09'	Push And Set Pattern	GPSPT
X'0A'	Set Color	GSCOL
X'0C'	Set Foreground Color Mix	GSMX
X'0D'	Set Background Color Mix	GSBMX
X'10'	Set Text Alignment	GSTA
X'11'	Fractional Line Width	GSFLW
X'18'	Set Line Type	GSLT
X'19'	Set Line Width	GSLW
X'21'	Set Current Position	GSCP
X'22'	Set Arc Parameters	GSAP
X'23'	Push And Set Pick (Tag) Identifier	GPSPIK
X'24'	Set Model Transform	GSTM
X'26'	Set Extended Color	GSECOL
X'27'	Set Viewing Window	GSVIEW
X'28'	Set Pattern	GSPT
X'29'	Set Marker Type	GSMT
X'33'	Set Character Box	GSCC
X'34'	Set Character Angle	GSCA
X'35'	Set Character Shear	GSCH
X'36'	Set Character-Box Spacing	GSCBS
X'37'	Set Marker Box	GSMC
X'38'	Set Character Set	GSCS
X'39'	Set Character Precision	GSCR
X'3A'	Set Character Direction	GSCD
X'3E'	Segment End Prolog	GEPROL
X'3F'	Pop Attribute	GPOP
X'41'	Marker Scale	GSMSC
X'43'	Set Pick (Tag) Identifier	GSPIK
X'4A'	Push And Set Color	GPSCOL
X'4C'	Push And Set Foreground Color Mix	GPSMX
X'4D'	Push And Set Background Color Mix	GPSBMX
X'50'	Push And Set Text Alignment	GPSTA
X'51'	Push And Set Fractional Line Width	GPSFLW
X'53'	Segment Position	GSSPOS
X'58'	Push And Set Line Type	GPSLT
X'59'	Push And Set Line Width	GPSLW
X'60'	End Area	GEAR
X'61'	Push And Set Current Position	GPSCP
X'62'	Push And Set Arc Parameters	GPSAP
X'64'	Push And Set Model Transform	GPSTM
X'66'	Push And Set Extended Color	GPSECOL
X'67'	Push And Set Viewing Window	GPVIEW
X'68'	Area	GBAR
X'69'	Push And Set Marker Type	GPSMT
X'70'	Segment Start	GBSEG
X'71'	Segment End	GESEG
X'72'	Segment Attribute	GISAT
X'73'	Segment Attribute Modify	GMSAT
X'74'	Push And Set Character Angle	GPSCA
X'75'	Push And Set Character Shear	GPSCH
X'76'	Push And Set Character-Box Spacing	GPSCBS
X'77'	Push And Set Marker Box	GPSMC
X'78'	Push And Set Character Set	GPSCS
X'79'	Push And Set Character Precision	GPSCR
X'7A'	Push And Set Character Direction	GPSCD
X'81'	Line (at current position)	GCLINE
X'82'	Marker (at current position)	GCMRK
X'83'	Character String (at current position)	GCCHST
X'85'	Fillet (at current position)	GCFLT

Table 27 (Page 2 of 2). Summary of GDF orders in order of code values

Code	Name of GDF order	Mnemonic
X'86'	Arc (at current position)	GCARC
X'87'	Full Arc (at current position)	GCFARC
X'91'	Image Begin (at current position)	GCBIMG
X'92'	Image Data	GIMD
X'93'	Image End	GEIMG
X'A1'	Relative Line (at current position)	GCRLINE
X'C1'	Line	GLINE
X'C2'	Marker	GMRK
X'C3'	Character String	GCHST
X'C5'	Fillet	GFLT
X'C6'	Arc	GARC
X'C7'	Full Arc	GFARC
X'D1'	Image Begin	GBIMG
X'E1'	Relative Line	GRLINE

Process specific orders (PSC)

Default process specific orders — numeric list

These orders are only valid within the picture prolog. They are listed in alphabetic order of order name on pages 184 through 189.

Table 28. Numeric list of default process specific orders

Order name	Order code
Set Default Foreground Color Mix	X'0C'
Set Default Background Color Mix	X'0D'
Set Default Coordinate Type	X'0E'
Set Default Text Alignment	X'10'
Set Default Fractional Line Width	X'11'
Set Default Line Type	X'18'
Set Default Picture Scale	X'20'
Set Default Arc Parameters	X'22'
Set Default Extended Color	X'26'
Set Default Pattern Symbol	X'28'
Set Default Marker Symbol	X'29'
Set Default Character Box	X'33'
Set Default Character Angle	X'34'
Set Default Character Shear	X'35'
Set Default Character-Box Spacing	X'36'
Set Default Marker Box	X'37'
Set Default Character Set	X'38'
Set Default Character Precision	X'39'
Set Default Character Direction	X'3A'
Set Default Pick Identifier	X'43'

General structure

A GDF stream consists of a sequence of orders.

Each order is identified by a one-byte order code and contains one or more bytes of operand data.

Order formats

The order is represented in one of two formats depending on the length of the operand data.

The first format applies to orders with up to 255 bytes of operand data. The second applies only to orders that have a single byte of operand data.

Normal format

In the normal format, there is a one-byte order code and a one-byte length field, followed by "length" bytes of operand data:

order code (1 byte)	length (can be zero) (1 byte)

```
  ...
operand data
(up to 255 bytes)
  ...
```

Therefore, the maximum possible length of a GDF order is:

```
    1              1           up to 255
order code  +  data length  +    data
   byte           byte           bytes
                    =
                   257
                  total
                  bytes
```

Short format

If the first hexadecimal digit of an order code is less than 8, and the second hexadecimal digit is 8 or greater, the GDF is a short-format order. This consists of two bytes; the first one is the order code (as just defined), and the second one contains the operand data.

order code (1 byte)	operand data (1 byte)

Padding

Orders can be followed by padding bytes X'00' so that the next order aligns on a convenient boundary.

Coordinate data

Many of the orders contain coordinate data or coordinate-related data. Coordinates may use different representations, either fixed or floating point.

When integer coordinates are used, the integers can be:

- Halfword length
- One-byte length.

These coordinate values are normal 7-bit or 15-bit numbers with sign. When negative, they are in twos-complement notation.

When floating-point coordinates are used, they are in standard short floating-point format.

The type and length of coordinates must be specified on the GSPUT call. This is constant for the string.

Primitives

The following graphics primitives can be represented:

- Line (relative or absolute)
- Marker
- Character string
- Curved "fillet"
- Arc (circular, elliptical, or full)
- Image.

The orders have a close correspondence with many of the GDDM functions.

Current position

The GDF order formats given below contain all relevant coordinates. However, for brevity the start position of the graphics primitive can be omitted. When omitted, current position is used in its place.

Current position is set by each of the orders. It is set to the end point of a line or arc and, except for character strings, the rule is the same as for the corresponding GDDM function.

The presence or absence of the initial coordinate is shown by bit 1 of the order code. When it is 0, the first coordinate in the order is omitted and current position is to be assumed in its place.

Coordinate lengths

In the following primitives, coordinates are assumed to be pairs of two-byte signed integers, assuming that this is the coordinate length in use. When one-byte or four-byte coordinates are used, the length of the order changes accordingly. Coordinate and coordinate-related fields are marked by "★"

Attributes

GDDM provides two forms of attribute order; these are:

- Push And Set
- Set.

GDDM maintains a stack of attributes, which can be removed from the stack by using the Pop order.

A Push And Set attribute order puts the current value of the attribute being set onto the attribute stack and sets the value of the attribute to the value in the order. The Pop order unstacks the most recently pushed attribute on the stack and sets the popped attribute to the value restored from the stack.

The difference between a Set attribute and a Push And Set attribute order is generally shown by the state of bit 1 of the order code, thus:

0	The order is a Set attribute
1	The order is a Push And Set attribute.

There are three exceptions to this rule; they are:

Pattern

- The Set form is X'28'
- The Push And Set form is X'09'.

Character Box

- The Set form is X'33'
- The Push And Set form is X'03'.

Pick (Tag) Identifier

- The Set form is X'43'
- The Push And Set form is X'23'

Both the Set and the Push And Set orders correspond to GDDM attribute setting functions, according to the current attribute mode; see the description of the GSAM call in the *GDDM Base Programming Reference, Volume 1*. For example, the GPSLT order corresponds to the GSLT function when the attribute mode is 0 (preserve attributes).

As with the equivalent call statements, attribute setting orders change the current values of the attributes. An attribute setting applies to all subsequent primitives (to which it is relevant) until a new setting is made.

Attribute-setting orders appearing in a GDF string argument to GSPUT affect the current attribute settings after the call. The effects are not purely local to primitives within the string, but may affect subsequent primitives.

The details of the effects of the orders are not given in full. For more explanation, see the corresponding call statement descriptions in the *GDDM Base Programming Reference, Volume 1*.

GDF orders: full descriptions

This section describes the content and format of the GDF orders, which are presented in alphabetic order.

Format of tables

Where applicable, the Set form of the order is described in an abbreviated way to reduce duplication. Only the hexadecimal order-code value details are repeated; the contents of the remaining fields are the same as for the described Push And Set form.

In these definitions, the Set form of the order is separated from the Push And Set form by a horizontal double line; the information under the Push And Set form then applies to both forms.

Where the Content of an order is given as "LEN", it means that the field length is variable; this is indicated by a "2★" in the Field length column.

Some orders are available as "order", or "order at current position"; only some of the following data applies to each one, and it is annotated accordingly.

Format of examples

The examples are given in hexadecimal, usually assuming halfword coordinates. Blanks in the hexadecimal strings are to aid readability; they have no other significance.

For detailed information about the GDDM calls mentioned, see the *GDDM Base Programming Reference, Volume 1.*

Arc

This order constructs an arc starting at (x0,y0), passing through (x1,y1), and ending at point (x2,y2).

The intermediate point (x1,y1) should, for greatest accuracy, lie midway along the arc. (If it coincides with either end point the arc becomes undefined.) The initial point and the final point must not coincide.

The arc may be part of a circle or part of the ellipse defined by the previous "arc parameters" order. a length proportional to "a"; the axis parallel to the y axis has a length proportional to "b".

The initial coordinate pair (x0,y0) may be omitted. Current position is then used as the starting point of the arc and the order code becomes X'86'.

The current position is set to point (x2,y2).

Fld len	Content	Meaning
1	X'C6' or X'86'	Arc order code (GARC) or Arc (at current position) (GCARC)
1	LEN	Length of following data
2★	x0 (omitted for order X'86')	x coordinate of start of arc
2★	y0 (omitted for order X'86')	y coordinate of start of arc
2★	x1	x coordinate of intermediate point
2★	y1	y coordinate of intermediate point
2★	x2	x coordinate of end of arc
2★	y2	y coordinate of end of arc

Arc parameters

This order determines the shape of subsequent arcs. The full parameters give a transformation that maps the unit circle to an ellipse of the required shape:

```
x' = Px + Ry
y' = Sx + Qy
```

A circle results if P=Q and R=S=0.

If P=a, Q=b, an ellipse results. The axis parallel to the x axis has a length proportional to "a"; the axis parallel to the y axis has a length proportional to "b."

If R and S are nonzero, the ellipse is tilted. Usually, for an ellipse with major and minor axes proportional to "a" and "b", tilted at angle "theta" to the x axis:

```
P = a.cos(theta)
Q = b.cos(theta)
R = -b.sin(theta)
S = a.sin(theta)
```

Fld len	Content	Meaning
1	*Set* X'22' *Push & set* X'62'	Arc Parameters order code
1	LEN	Length of following data
2★	P	x coordinate of major axis end
2★	Q	y coordinate of minor axis end
2★	R	x coordinate of minor axis end
2★	S	y coordinate of major axis end

Area

The Area order approximates to the GSAREA and GSENDA functions.

Fld len	Content	Meaning
1	X'68'	Area order code
1	FLAGS	Bit 0 = on if this marks the start of an area = off if this order marks the end of an area. Bit 1 = on if the boundary lines are to be drawn

Examples:

```
68 80
C1 0A 00 00  05 00  05 05  00 05  00 00
68 00
```

Draws a rectangular area 5 units square. Boundary lines are not drawn.

```
68 C0
C1 0A 00 00  05 00  05 05  00 05  00 00
68 00
```

Draws the same area, but includes boundary lines.

Background color mix order

The Background Color Mix order corresponds to the GSBMX function.

Fld len	Content	Meaning
1	*Set* X'0D' *Push & set* X'4D'	Background Color Mix order code
1	MODE	X'00' Default X'01' OR X'02' Overpaint X'03' Underpaint X'04' Exclusive-OR (implemented as overpaint) X'05' leave alone

Call segment order

The Call Segment order corresponds to the GSCALL function.

Fld len	Content	Meaning
1	X'07'	Call Segment order code
1	X'06'	Length of data
2	X'0000'	Reserved
4	SEGID	Identifier of segment to be called

Character angle

The Character Angle order corresponds to the GSCA function. It controls the angle of subsequent character strings.

Fld len	Content	Meaning
1	*Set* X'34' *Push & set* X'74'	Character Angle order code
1	LEN	Length of following data
2★	Ax	x coordinate of a point that defines the angle of the text
2★	Ay	y coordinate of the point

Note: Ax and Ay specify a relative vector that defines the angle of the baseline of the string. When the coordinate (x,y) is on the baseline, (x + Ax, y + Ay) is also on the baseline.

When both Ax and Ay are zero, the current character angle attribute is set to the drawing default value.

Character box

The Character Box order corresponds to the GSCB function. The order specifies the size of characters in following character strings.

For 1-byte or 2-byte integer coordinates, the order can optionally be extended to provide a fractional portion of the character box; see the FRACTWIDTH and FRACTDEPTH fields.

Fld len	Content	Meaning
1	*Set* X'33' *Push & set* X'03'	Character Box order code
1	LEN	Length of following data
2★	CHARWIDTH	Width of character box
2★	CHARHEIGHT	Height of character box
2★	FRACTWIDTH	Fractional portion of character box width, specified as multiples of 1/65536 for 2-byte format or 1/256 for 1-byte format
2★	FRACTDEPTH	Fractional portion of character box depth, specified as multiples of 1/65536 for 2-byte format or 1/256 for 1-byte format

Note: When either the fractional width or depth is to be specified, both must be included. The integer and fractional character widths (depths) together form a character width (depth) that defines the character box required.

Character-box spacing

The Character-Box Spacing order corresponds to the GSCBS function.

The order specifies the spacing of characters in following character strings.

Fld len	Content	Meaning
1	*Set* X'36' *Push & set* X'76'	Character-Box Spacing order code
1	LEN	Length of following data
1	FLAGS	Bit 0 value 0 Set character box spacing Bit 0 value 1 Set default character box spacing Bits 1 through 7 Reserved — must be set to X'0000000'
1	X'00'	Reserved
2★	HSPACE	Horizontal character-box spacing
2★	VSPACE	Vertical character-box spacing

Character direction

The Character Direction order corresponds to the GSCD function.

Fld len	Content	Meaning
1	*Set* X'3A' *Push & set* X'7A'	Character Direction order code
1	DIRECTION	Value for character direction This is interpreted as follows: X'00' Default X'01' Left to right X'02' Top to bottom X'03' Right to left X'04' Bottom to top
Note: The character direction gives the placement of each character relative to the previous one, either along or perpendicular to the baseline.		

Character precision

The Character Precision order corresponds to the GSCM function.

Fld len	Content	Meaning
1	*Set* X'39' *Push & set* X'79'	Character Precision order code
1	DIRECTION	Value for character mode attribute These are interpreted as follows: X'00' Default X'01' String precision X'02' Character precision X'03' Stroke precision Other Not defined

Character set

The Character Set order corresponds to the GSCS function.

Fld len	Content	Meaning
1	*Set* X'38' *Push & set* X'78'	Character Set order code
1	LCID	Local identifier for the character set: X'00' Default X'01' APL X'41' through X'DF' User-defined set Other Not defined

Character shear

The Character Shear order corresponds to the GSCH function. It controls the shear of subsequent characters.

Fld len	Content	Meaning
1	*Set* X'35' *Push & set* X'75'	Character Shear order code
1	LEN	Length of following data
2★	Hx	Hx and Hy specify a relative vector that defines the angle at which characters are to be sheared.
2★	Hy	y increment: see above

Notes:

1. Hx and Hy specify a vector that defines the angle of the upright strokes of a character relative to the baseline. If the lower left-hand corner of a character is placed at (0,0) and the character baseline lies along the x axis, the line from (0,0) to (Hx,Hy) gives the direction of upright strokes.
2. If both Ax and Ay are zero, the current shear attribute is set to the drawing default value.

Character string

The Character String order corresponds to the GSCHAR and GSCHAP functions.

Fld len	Content	Meaning
1	X'C3' or X'83'	Character String order code or Character String (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'83')	x coordinate at which character string is to be placed
2★	y0 (omitted for order X'83')	y coordinate of character string
V	STRING	EBCDIC character code of each character in the string. All characters above and including X'40' are valid.

Note: The character string is placed at the indicated coordinate. The attributes of the string (for example, mode, size, angle) are taken from the current values.

If the character string has a length that is odd, the length field in the order contains an odd number. The order must be padded with padding characters to an even number of bytes.

The position (x0,y0) may be omitted, in which case the order code becomes X'83' and the string is placed at the current position.

Current position is not changed. (This is different from GSCHAR.)

Examples:

```
C3 08 0002 0003 C1C2C3C4
```

Draws the character string "ABCD" at coordinate (2,3).

```
83 03 C5C6C7
```

Draws the character string "EFG" at the current position.

Color

The Color orders approximate to the GSCOL function.

Fld len	Content	Meaning
1	*Set* X'0A' *Push & set* X'4A'	Color order code
1	COLOR	Value for color attribute. The value is an index to a notional color table as follows: X'00' Default X'01' Blue X'02' Red X'03' Magenta (pink) X'04' Green X'05' Turquoise (cyan) X'06' Yellow X'07' Neutral: white on displays black on hardcopy X'08' Background: black on displays white on hardcopy Other Not defined
Note: The GSCOL call and the X'0A' Set Color order may be mapped to the X'26' Set Extended Color order as follows: • Colors 0 through 8 are mapped to X'FF00' through X'FF08' in the Extended Color order on page 175. • All other values map directly to a two-byte hexadecimal value.		

Fld len	Content	Meaning
1	*Set* X'26' *Push & set* X'66'	Extended Color order code
1	LEN	Length of following data
2	COLOR	Value for color attribute The value is an index into a notional color table, as follows: X'0000' Default X'FF00' Default X'0001' or X'FF01' Blue X'0002' or X'FF02' Red X'0003' or X'FF03' Magenta (pink) X'0004' or X'FF04' Green X'0005' or X'FF05' Turquoise (cyan) X'0006' or X'FF06' Yellow X'0007' White X'0008' Black X'FF07' Neutral/ multicolor (white on displays, black on hardcopy) Other values are not defined. If a color value is outside the range of color values supported by a device, the color displayed is device-dependent (see GSCOL). For color separation on family-4 devices, the color values depend on the loaded color table.
Note: All subsequent primitives have the color given until this is reset.		

Comment

The Comment order holds GDDM or application-program data within a GDF stream. Comments are stored in floating-point GDF.

The first GDF order returned by GSGET (and, by convention, in GDF files) contains the coordinate range and coordinate type. This convention is maintained in GDDM Version 2 Release 1 but has been superseded by Process Specific Control (PSC) orders.

Fld len	Content	Meaning
1	X'01'	Comment order code
1	LEN	Length of following data
2	COORD-TYPE	2 2-byte integer 4 4-byte integer
2★	xL	x lower boundary of picture space
2★	xU	x upper boundary of picture space
2★	yL	y lower boundary of picture space
2★	yU	y upper boundary of picture space
Note: It is recommended that Comment orders, created by an application program to contain application-specific information, should take the following form. (GDDM suggests the following convention but does not enforce it.)		

Fld len	Content	Meaning
1	X'01'	Comment order code
1	LEN	Length of following data
2	COORD-TYPE	Reserved as X'0000'
8	IDENT	Application identifier
n	DATA	User data

Current position

The Current Position orders approximate to the GSCP call.

Fld len	Content	Meaning
1	*Set* X'21' *Push & set* X'61'	Current Position order code
1	LEN	Length of following data
2★	x	x coordinate of new current position
2★	y	y coordinate of new current position

End area

The End Area order has the same meaning as the Area order (see page 172), with the "end area" bit set. GDDM accepts both forms of order, but only generates the X'60' End Area order.

Fld len	Content	Meaning
1	X'60'	End Area order code
1	LEN	Length of following data
N	X'00'	Reserved (must be all nulls)

Fillet

The Fillet order approximates to the GSPFLT function.

Fld len	Content	Meaning
1	X'C5' or X'85'	Fillet order code or Fillet (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'85')	x coordinate of line start
2★	y0 (omitted for order X'85')	y coordinate of line start
2★	x1	x coordinate of first line end
2★	y1	y coordinate of first line end
2★	x2	x coordinate of second line end
2★	y2	y coordinate of second line end
	...	...

Note: The order shown generates a single fillet. More coordinate pairs may be added to form a polyfillet. The points are joined in order by imaginary straight lines. A curve is then fitted to the lines as follows. The curve is tangential to the first line at its starting point and to the last line at its end point. If there are intermediate lines, the curve is tangential to these lines at their center points. In the special case when only two points are supplied, a straight line results.

The initial coordinate pair (x0,y0) may be omitted. The current position is then used as the starting point of the arc, and the order code becomes X'85'. The current position is set to the last point specified.

Examples:

```
C5 08 0002 0003  0004 0006
```

Draws a line from coordinate (2,3) to coordinate (4,6).

```
C5 0C 0000 0000  0004 0000  0004 0004
```

Draws a curve, beginning at coordinate (0,0) and tangential to the line from (0,0) to (4,0). Initially the curve is horizontal. The curve then takes an approximately circular path to meet the line from (4,0) to (4,4) at (4,4).

```
C5 08 00 00  04 00  04 08  00 08
```

assuming byte coordinates, draws two curves. The first is that in the previous example and the second completes an approximation to a semicircular arc.

Foreground color mix

The Foreground Color Mix order corresponds to the GSMIX function.

Fld len	Content	Meaning
1	*Set* X'0C' *Push & set* X'4C'	Foreground Color Mix order code
1	MODE	Value for mix-mode attribute. Mix mode controls how an inserted primitive affects the existing picture. In all modes, generated 0 bits leave the underlying features untouched; new 1 bits become whatever the current color is if that bit was previously of background color. The effect of a new 1 bit over an existing 1 bit depends on the particular mode: the old color (underpaint), the new color (over-paint), or a mixture (mix) may result. X'00' Default X'01' Mix X'02' Overpaint X'03' Underpaint X'04' Exclusive-OR X'05' Leave alone Other Not defined

Fractional line width

The Fractional Line Width order corresponds to the GSFLW function.

Fld len	Content	Meaning
1	*Set* X'11' *Push & set* X'51'	Fractional Line Width order code
1	LEN	Length of following data
1	INTEGRAL LINE WIDTH	The integer portion of the line-width multiplier
1	FRACTIONAL LINE WIDTH	The fractional portion of the line-width multiplier, specified as multiples of 1/256
Note: The integral and fractional line widths together form a line-width multiplier that defines the line width required. For an explanation of the interpretation of the multiplier, see the *GDDM Base Programming Reference, Volume 1.*		

Full arc

The Full Arc order allows a complete circle or ellipse to be specified in one order. The size and shape of the circle or ellipse are determined by the Set Arc Parameters order; see page 171. Note that the Set Arc Parameters order sets the relative lengths of the major and minor axes for three-point arcs, but for the Full Arc order it sets the absolute size, in world coordinates, of a full circle or ellipse.

The coordinate pair may be omitted. The order code then becomes X'87', and the arc is drawn with its center at the current position. The current position is unchanged. (The X'87' version of the order draws the arc at the current position.)

The major and minor axes of the arc are defined as

`M.a  M.b`

where

M is the two-byte unsigned fractional fixed-point multiplier; the first eight bits are the integral part and the second eight bits are the fractional part.

a and b are the lengths of the major and minor axes obtained from the Arc Parameters order; see page 171.

A Full Arc order is allowed in an area definition and causes the area to be closed.

Fld len	Content	Meaning
1	X'C7' or X'87'	Full Arc (at given position) order code or Full Arc (at current position) order code
1	LEN	Length of following data
2★	x (omitted for order X'87')	x Coordinate value
2★	y (omitted for order X'87')	y Coordinate value
2	M	Multiplier
Note: The arc is drawn with its center at point (x,y), which becomes the current position.		

Image – begin

The Begin Image order, together with the Image Data and End Image orders approximate to the GSIMG and GSIMGS functions.

An image consists of a rectangular array of display points.

It is represented by a sequence of orders. The first is a Begin Image order and the last is an End Image order (see below). Between these delimiters, several Image Data orders may occur, giving the array of display points in the image.

The initial coordinate pair (x0,y0) can be omitted. The current position is then used to place the image data, and the order code becomes X'91'. The current position is not changed by a series of Image orders.

The size of the display point array and its representation are given by the Begin Image order. The fields IMAGEWIDTH and IMAGEDEPTH are optional and may be either both specified, or both omitted. When specified, the image is scaled to fill the area identified by the fields, using the rules defined in the GSIMGS call. When omitted, each display point is represented by one bit in the display point array.

Fld len	Content	Meaning
1	X'D1' or X'91'	Begin Image (at given position) order code or Begin Image (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'91')	The x position at which the image is to be placed
2★	y0 (omitted for order X'91')	The y position at which the image is to be placed
2	FORMAT	The format of the image data. This field must have the value 0.
2	WIDTH	The width of the image in display points
2	DEPTH	The depth of the image in display points
2★	IMAGEWIDTH	The desired width of the image in coordinate units
2★	IMAGEDEPTH	The desired depth of the image in coordinate units

Image – data

For image FORMAT 0, each Image Data order contains the display points for one row of the display point array. Thus for an image with a DEPTH of N, there are N Image Data orders between the Begin and End Image orders. Each Image Data order contains data for WIDTH display points. Each display point is represented by a single bit. When the bit is one, the display point is "on"; when the bit is zero, the display point is "off".

Fld len	Content	Meaning
1	X'92'	Image Data order code
1	LEN	Length of following data
V	PIXELDATA	The display points of the image

Example:

```
91 06 0000 0009 0004
92 02 FF80
92 02 8080
92 02 8080
92 02 FF80
93 02 0000
```

Draws an image, whose size is nine display points wide by four deep, at the current position. The image consists of a small square of "on" display points (which appear in the current color) surrounding an "off" center.

Image – end

This order ends the construction of an image.

Fld len	Content	Meaning
1	X'93'	End Image order code
1	LEN	Length of following data
2	X'0000'	Reserved

Line

The Line order approximates to the GSPLNE function.

Fld len	Content	Meaning
1	X'C1' or X'81'	Line order code or Line (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'81')	x coordinate of line start order code
2★	y0 (omitted for order X'81')	y coordinate of line start order code
2★	x1	x coordinate of first line end
2★	y1	y coordinate of first line end
	. . .	. . .

Note:

A line is drawn from the first coordinate given (x0,y0) to the second (x1,y1).

The order shown is for a single line, but usually any number of coordinates can be present. Consecutive coordinates in the order are joined by straight lines. The data length must be an even multiple of the coordinate length.

The initial coordinate pair (x0,y0) may be omitted. Current position is then used as the starting point of the first line and the order code becomes X'81'.

Current position is set to the last point specified.

Note that a line order with only an initial position is permitted. This serves only to move current position.

Examples

```
C1 08 0002 0003  0004 0006
```

Draws a line from coordinate (2,3) to coordinate (4,6).

```
C1 0C 0002 0003  0004 0006  0009 0009
```

Draws a line from coordinate (2,3) to coordinate (4,6) and a line from (4,6) to (9,9).

```
C1 06 02 03  04 06  09 09
```

Draws the same lines, but 1-byte coordinates are used.

```
C1 04 0002 0003
```

Draws no lines. However, current position is changed to the last coordinate (2,3).

```
81 04 0004 0006
```

Draws a line from current position to point (4,6). Thus, the pair of orders:

```
C1 04 0002 0003
81 04 0004 0006
```

has the same effect as the first:

```
C1 08 0002 0003  0004 0006.
```

Line type

The Line Type order corresponds to the GSLT function.

Fld len	Content	Meaning
1	*Set* X'18' *Push & set* X'58'	Set Line Type order code
1	LINETYPE	Value for line type attribute. The value is an index into a notional line type table as follows: X'00' Default X'01' Dotted line X'02' Short dashed line X'03' Dash-dot line X'04' Double-dotted line X'05' Long-dashed line X'06' Dash-double-dot line X'07' Solid line X'08' Invisible line Other Not defined

Line width

The Line Width order corresponds to the GSLW function.

Fld len	Content	Meaning
1	*Set* X'19' *Push & set* X'59'	Set Line Width order code
1	LINEWIDTH	Value for line-width attribute

Marker

The Marker order approximates to the GSMRKS function.

Fld len	Content	Meaning
1	X'C2' or X'82'	Marker order code or Marker (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'82')	x coordinate of marker
2★	y0 (omitted for order X'82')	y coordinate of marker
	...	...

Note:

The order is shown for a single marker. More coordinate pairs may be added. The current marker is placed at each point specified.

The first (or only) coordinate pair may be omitted. The order code then becomes X'82', and a marker is placed at the current position in addition to any points specified.

The current position is set to the last coordinate specified, or, if none, is unchanged.

Examples:

```
C2 04 0002 0003
```

Draws the current marker at coordinate (2,3).

```
C2 0C 0002 0003  0004 0006  0009 0009
```

Draws markers at (2,3) (4,6) and (9,9).

```
82 00
```

Draws the current marker at current position.

Marker box

The Marker Box order specifies the size of the cell used for scaling vector markers.

Fld len	Content	Meaning
1	*Set* X'37' *Push & set* X'77'	Marker Box order code
1	LEN	Length of following data
2★	MARKER-WIDTH	Width of marker cell
2★	MARKER-HEIGHT	Height of marker cell

Marker scale

The Marker Scale order approximates to the GSMSC function. It sets the scale of the marker box with respect to the default marker box.

Fld len	Content	Meaning
1	X'41'	Marker Scale order code
1	LEN	Length of following data
4	SCALE	Scale of marker box with respect to the default marker box in floating-point format

Marker type

The Marker Type order corresponds to the GSMS function.

Fld len	Content	Meaning
1	*Set* X'29' *Push & set* X'69'	Marker Type order code
1	N	Marker number. The attribute determines which symbol is displayed by the marker primitive. The interpretation of the values is: X'00' Default X'01' Cross X'02' Plus X'03' Diamond X'04' Square X'05' 6-point star X'06' 8-point star X'07' Filled diamond X'08' Filled square X'09' Dot X'0A' Small Circle X'0B' through X'40' Not defined X'41' through X'EF' User-defined

Model transform

The Model Transform order is a transformation matrix.

Fld len	Content	Meaning
1	*Set* X'24' *Push & set* X'64'	Model Transform order code
1	LEN	Length of following data
1	X'00'	Reserved
1	FLAGS	Bits 0 — 5: Reserved (all 0) Bits 6 — 7: B'00' Replace B'01' Postmultiply B'10' Premultiply (equivalent to GSSTFM preemptive). Other values are not supported
2	MASK	Load Mask Values
N	MATRIX	Transformation Matrix

Note:

A segment transformation is defined by a matrix

$$M = \begin{bmatrix} M11 & M12 & M13 & M14 \\ M21 & M22 & M23 & M24 \\ M31 & M32 & M33 & M34 \\ M41 & M42 & M43 & M44 \end{bmatrix}$$

which is applied to primitives p to give p′ as follows:

```
(x',y',z',1) = (x,y,z,1).M
```

The GDF order defines the matrix elements in the order

```
M11,M12,...,M14,M21,...,M44
```

This differs from the GSSTFM call, which specifies the matrix elements in the order

```
M11,M21,M31,M12,...,M33
```

The MASK field identifies those elements of the transformation defined by the MATRIX field. The bits within MASK correspond to, in order, elements

```
M11,M12,...,M14,M21,...M44
```

of the transformation. The values provided in MATRIX correspond, in order, to those elements of the transformation identified by bits set to 1 within MASK. All uninitialized values within the transformation matrix are set from the identity transformation.

Only elements M11, M12, M21, M22, M41, and M42 are processed by GDDM. All other elements must be zero or one (as in the identity matrix).

The transformation elements may be specified in one-byte, two-byte, or four-byte form, corresponding to the data type GDF coordinates.

The fixed-point representation of the matrix elements is: M41, M42 are twos complement numbers (8-bit or 16-bit). Elements M11, M12, M21, and M22 are twos complement numbers in the following form:

```
SB.bb bbbb (bbbb bbbb)
```

where S is the sign bit (1 = negative), B is the integer bit, and b the fractional bits (6 or 14 of them).

Pattern

The Pattern order corresponds to the GSPAT function.

Fld len	Content	Meaning
1	*Set* X'28' *Push & set* X'09'	Pattern order code
1	PATTERN	Value for pattern attribute. This attribute determines which pattern (either built-in or defined through the GSLSS call) is to be used to shade the interior of subsequent areas. The interpretation of the values is: X'00' Default X'01' through X'08' Density 1 to density 8 (decreasing) X'09' Vertical lines X'0A' Horizontal lines X'0B' Diagonal lines 1 (bottom left to top right) X'0C' Diagonal lines 2 (bottom left to top right) X'0D' Diagonal lines 1 (top left to bottom right) X'0E' Diagonal lines 2 (top left to bottom right) X'0F' No shading X'10' Solid shading X'11' through X'40' Not defined X'41' through X'EF' User-defined (See the description of the GSPAT call in the *GDDM Base Programming Reference, Volume 1*)

Pick (tag) identifier

The Pick order corresponds to the GSTAG function.

A value of X'00000000'is considered a "null" value. Any output primitive that is given a null tag does not take part in correlation.

Fld len	Content	Meaning
1	*Set* X'43' *Push & set* X'23'	Pick (Tag) Identifier order code
1	4	Length of following data
4	TAG	Tag value

Pop

The Pop order pops the top attribute of the current attribute stack and sets the popped attribute to the restored value.

The order is valid outside segments but the results can be unpredictable because GDDM limits the size of the stack that can be created by temporary primitive attributes.

Fld len	Content	Meaning
1	X'3F'	Pop order code
1	X'00'	Reserved

Process specific control

GDDM uses the Process Specific Control (PSC) order to store picture prolog and symbol-set information in a GDF object. Because it does not add any graphics data to the picture, the order is ignored by the GSPUT call.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LENGTH	Length of following data
1	X'xx'	Process Identifier
1	X'xx'	Function Identifier
N	DATA	Process-specific Controls
Note: GSGET returns two types of PSCs: • X'01' – for symbol-set names • X'02' – for the picture prolog.		

Symbol-set names

The symbol-set PSCs contain the names and types of the symbol sets that are currently loaded. When an IBM 4250 printer is the primary device, the code page is returned, as well as the symbol-set name and type.

There are three PSCs for defining symbol-set names. These are:

Begin Symbol-Set Mapping – X'017E'
Map Symbol-Set Identifier – X'0140'
End Symbol-Set Mapping – X'017F'

The symbol-set types and names are recorded in the Map Symbol-Set Identifier PSC. There is one control for each symbol set. All the controls are bracketed between a Begin Symbol-Set Mapping and End Symbol-Set Mapping PSC, as defined below.

GSGET returns one group of symbol-set mapping information.

Begin Symbol-set mapping

The Begin Symbol-Set Mapping PSC precedes the picture prolog and the segments.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'02'	Length of following data
1	X'01'	GDDM Symbol-Set Process Identifier
1	X'7E'	Begin Symbol-Set Mapping

Map Symbol-set identifier

Several Map Symbol-Set Identifier PSC orders follow the Begin Symbol-Set Mapping PSC order.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'01'	GDDM Symbol-Set Process Identifier
1	X'40'	Map symbol-set ID to name and type
1	TYPE	Type of symbol set (see below)
1	NUMBER	Number of symbol-set definition (see below)
1	ID	Symbol-set identifier (LCID)
8	SS-NAME	Symbol-Set Name
8	CP-NAME	Code Page Name (optional)

Note: The **TYPE** parameter can have the following values:

X'01'	Image symbol set
X'02'	Vector symbol set
X'03'	Shading pattern
X'04'	Marker (image) symbol set
X'05'	4250 printer image symbol set
X'06'	Marker (vector) symbol set.

The **NUMBER** parameter is always returned as zero by GSGET.

End Symbol-set mapping

The End Symbol-Set Mapping PSC order appears at the end of the Map Symbol-Set Identifier PSC orders. Within the symbol-set mapping structure, any orders other than PSCs, comments, and no operations cause an implicit end to symbol-set mapping. A warning message is issued.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'02'	Length of following data
1	X'01'	GDDM Symbol-Set Process Identifier
1	X'7F'	End Symbol-Set Mapping

Picture prolog

GDDM uses PSCs to define the coordinate type, the extent of GDF, and the default attribute values. The PSCs are known as drawing defaults. They are returned in the GDF after the map symbol-set PSC orders (if these are present).

All the Picture Prolog PSC orders are bracketed between a Begin Picture Prolog PSC order and an End Picture Prolog PSC order.

Begin picture prolog

The Begin Picture Prolog PSC order precedes the Picture Prolog PSC order. Only one picture prolog is returned by the GSGET call.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'02'	Length of following data
1	X'02'	GDDM Picture Prolog Identifier
1	X'7E'	Begin Picture Prolog

Set picture boundary

The Set Picture Boundary PSC defines the picture space. For fixed-point GDF, this is in device coordinates. For floating-point GDF, it is in world coordinates. This PSC contains the same data as the Set Default Viewing Window PSC and should be used for setting window coordinates when redisplaying a GDF picture.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Common Picture Prolog
1	X'32'	Set Picture Boundary
1	LEN-3	Length of following data
1	X'00'	Reserved
1	MASK	Each bit has a value shown below
	Bit 0 Reserved 1 Reserved 2 xL x left limit B'0' Not included in list of WW values B'1' Included in list of WW values 3 xR x right limit B'0' Not included in list of WW values B'1' Included in list of WW values 4 yB y bottom limit B'0' Not included in list of WW values B'1' Included in list of WW values 5 yT y top limit B'0' Not included in list of WW values B'1' Included in list of WW values 6 zN z near limit B'0' Not included in list of WW values B'1' Included in list of WW values 7 zF z far limit B'0' Not included in list of WW values B'1' Included in list of WW values	
0-16	WW	Window values
Note: The coordinates can be 2-byte fixed-point data or 4-byte floating-point data.		

Set Picture Origin

The Set Picture Origin PSC defines the lower left-hand corner of the graphics field.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	GDDM Picture Prolog Identifier
1	X'01'	Set Picture Origin order code
1	LEN-3	Length of following data
2★	x0	x coordinate of picture origin
2★	y0	y coordinate of picture origin
Note: The values returned in x0,y0 are the coordinates of the lower left-hand corner of the graphics field.		

Default process specific orders

All of these orders are only valid within the picture prolog. They are listed in numeric order of order code in Table 28 on page 169.

Set default arc parameters

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'22'	Set Default Arc Parameters
1	LEN	Length of following data
2★	P	P parameter of the transform
2★	Q	Q parameter of the transform
2★	R	R parameter of the transform
2★	S	S parameter of the transform

Set default background mix

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'0D'	Set Default Background Mix
1	MODE	Background Mix Value. This value is interpreted as follows: X'00' Standard default X'02' Opaque X'05' Transparent

Set default character angle

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'34'	Set Default Character Angle
1	LEN	Length of following data
2★	AX	x coordinate of a point that defines the angle of the string
2★	AY	y coordinate of a point that defines the angle of the string
Note: AX and AY specify a relative vector that defines the angle of the baseline of the string.		

Set default character box

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'33'	Set Default Character Box
1	LEN	Length of following data
2★	CHARWIDTH	Width of character box
2★	CHARHEIGHT	Height of character box

Set default character-box spacing

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'36'	Set Default Character-Box Spacing
1	X'06'	Length of following data
1	FLAGS	Bit 0 0 – set character-box spacing 1 – set default character-box spacing Bits 1 through 7 Reserved – must be set to X'0000000'
1	X'00'	Reserved
2	HSPACE	Horizontal character-box spacing
2	VSPACE	Vertical character-box spacing

Set default character direction

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'3A'	Set Default Character Direction
1	DIRECTN	Direction interpreted as follows: X'00' Standard default X'01' Left to right X'02' Top to bottom X'03' Right to left X'04' Bottom to top Other Reserved

Set default character precision

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'39'	Set Default Character Precision
1	MODE	Precision interpreted as follows: X'00' Standard default X'01' String precision X'02' Character precision X'03' Stroke precision Other Reserved

Set default character set

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'38'	Set Default Character Set
1	LCID	Local identifier for the character set X'00' Standard default X'01' through X'FF' Specified symbol set

Set default character shear

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'07'	Picture Prolog Identifier
1	X'35'	Set Default Character Shear
1	LEN	Length of following data
2★	HX	HX and HY specify a relative vector that defines the angle at which characters are to sheared
2★	HY	y increment: see above

Set Default Coordinate Type

The Set Default Coordinate Type PSC defines the coordinate type of the primitive coordinates in the segments that follow. The default coordinate type is 2-byte fixed point.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	GDDM Picture Prolog Identifier
1	X'0E'	Set Default Coordinate Type
1	bit 0 bit 1 - 7	0 = 2-byte fixed-point number 1 = 4-byte floating-point number Reserved as 0

Set default extended color

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'05'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'26'	Set Default Extended Color
1	X'02'	Length of following data
2	COLOR	Color interpreted as follows: X'0000' Standard default X'0001' through X'0006' As for X'FF01' through X'FF06' X'FF00' Standard default X'FF01' Blue X'FF02' Red X'FF03' Magenta (pink) X'FF04' Green X'FF05' Turquoise (cyan) X'FF06' Yellow X'FF07' Neutral X'FF08' Background X'0007' White X'0008' Black X'0009' Dark blue X'000A' Orange X'000B' Purple X'000C' Dark green X'000D' Dark turquoise (cyan) X'000E' Mustard X'000F' Gray X'0010' Brown Other Reserved

Set default foreground mix

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'0C'	Set Default Foreground Mix
1	MODE	Foreground Mix Value. This value is interpreted as follows: X'00' Standard default X'01' OR (Mix) X'02' Opaque X'03' Underpaint X'04' Exclusive-OR X'05' Transparent

Set default fractional line width

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'04'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'11'	Set Default Fractional Line Width
1	INTEGRAL LINE WIDTH	The integer portion of the line-width multiplier
1	FRACTIONAL LINE WIDTH	The fractional portion of the line-width multiplier, specified as multiples of 1/256

Set default line type

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'18'	Set Default Line Type
1	TYPE	Line Type interpreted as follows: X'00' Standard default X'01' Dotted X'02' Short dashed X'03' Dash-dot X'04' Double dotted X'05' Long dashed X'06' Dash-double-dot X'07' Solid X'08' Invisible Other Reserved

Set default marker box

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'37'	Set Default Marker Box
1	LEN	Length of following data
2★	MARKER-WIDTH	Width of marker box
2★	MARKER-HEIGHT	Height of marker box

Set default marker type

This order is only valid within the picture prolog.

Fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'29'	Set Default Marker Symbol
1	SYMBOL	Marker symbol interpreted as follows X'00' Standard default X'01' Cross X'02' Plus sign X'03' Diamond X'04' Square X'05' 6-point star X'06' 8-point star X'07' Filled diamond X'08' Filled square X'09' Dot X'0A' Small circle X'41'through X'EF' User-defined

Set default pattern symbol

This order is only valid within the picture prolog.

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'03'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'28'	Set Default Pattern Symbol
1	SYMBOL	Symbol interpreted as follows: X'00' Standard default X'01' through X'08' Decreasing density X'09' Vertical lines X'0A' Horizontal lines X'0B' Diagonal lines 1 (bottom-left to top-right) X'0C' Diagonal lines 2 (bottom-left to top-right) X'0D' Diagonal lines 1 (top-left to bottom-right) X'0E' Diagonal lines 2 (top-left to bottom-right) X'0F' No shading X'10' Solid shading X'41'through X'EF' User-defined

Set Default Pick Identifier

This order is only valid within the picture prolog.

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'07'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'43'	Set Default Pick Identifier
1	X'04'	Length of following data
4	PICKID	Pick Identifier

Set Default Picture Scale

The format of the Set Default Picture Scale PSC is as follows:

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Common Picture Prolog
1	X'20'	Set Default Picture Scale order code
1	LEN-3	Length of following data
4	x	x scaling factor (see text)
4	y	y scaling factor

Notes:

1. When the coordinate type is two-byte fixed, the first halfword encodes the integer part of the scaling factor and the second halfword encodes the fractional part.
2. The scaling factors specify the number of coordinate units per millimeter. The default scaling factor is 20 per millimeter. Zero and negative scaling factors are not valid.

Set Default Text Alignment

The format of the Set Default Text Alignment PSC is as follows:

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'04'	Length of following data
1	X'02'	Picture Prolog Identifier
1	X'10'	Set Default Text Alignment
1	Hor TA	Horizontal Text Alignment. Interpreted as follows: X'00' Default X'01' Normal X'02' Left X'03' Center X'04' Right X'FF' Standard
1	Ver TA	Vertical Text Alignment. Interpreted as follows: X'00' Default X'01' Normal X'02' Top X'03' Cap X'04' Half X'05' Base X'06' Bottom X'FF' Standard

Set default viewing window

The Set Default Viewing Window PSC defines the picture space. For fixed-point GDF, this is in device coordinates. For floating-point GDF, it is in world coordinates. This PSC contains the same data as the Set Picture Boundary PSC and should be considered as defining a view of a picture.

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	LEN	Length of following data
1	X'02'	Common Picture Prolog
1	X'32'	Set Picture Boundary
1	LEN-3	Length of following data
1	X'00'	Reserved
1	MASK	Each bit has a value shown below
	Bit 0 Reserved 1 Reserved 2 xL x left limit B'0' Not included in list of WW values B'1' Included in list of WW values 3 xR x right limit B'0' Not included in list of WW values B'1' Included in list of WW values 4 yB y bottom limit B'0' Not included in list of WW values B'1' Included in list of WW values 5 yT y top limit B'0' Not included in list of WW values B'1' Included in list of WW values 6 zN z near limit B'0' Not included in list of WW values B'1' Included in list of WW values 7 zF z far limit B'0' Not included in list of WW values B'1' Included in list of WW values	
0-16	WW	Window values
Note: The coordinates can be 2-byte fixed-point data or 4-byte floating-point data.		

End Picture Prolog

The End Picture Prolog PSC follows the Picture Prolog PSC orders.

fld len	Content	Meaning
1	X'02'	Process Specific Control order code
1	X'02'	Length of following data
1	X'02'	GDDM Picture Prolog Identifier
1	X'7F'	End Picture Prolog

Relative line

The Relative Line order defines one or more straight lines. The end point of each line is given as a one-byte signed offset from the start of the line. Note that the offsets are always one-byte fixed, even in the floating-point form.

Order code X'A1' omits the current position, and draws lines from the current position.

fld len	Content	Meaning
1	X'E1' or X'A1'	Relative Line order code or Relative Line (at current position) order code
1	LEN	Length of following data
2★	x0 (omitted for order X'A1')	x coordinate of line start
2★	y0 (omitted for order X'A1')	y coordinate of line start
1	x1	x1 coordinate of first line end point
1	y1	y1 coordinate of first line end point
	. . .	

Notes:

1. The current position is updated to the last point specified.
2. The data length must be an even multiple of the coordinate length.

Segment attribute

The Segment Attribute order approximates to the GSSATI function. It sets the attributes to be assigned to subsequently generated segments.

fld len	Content	Meaning
1	X'72'	Segment Attribute order code
1	LEN	Length of following data
1	ATTRIBUTE	The attribute to be set for subsequent segments. The attributes that can be set are: X'01' Detectability X'02' Visibility X'03' Highlighting X'04' Transformability X'05' Reserved X'06' Chained
1	VALUE	The value to be assigned to the specified attribute. The values that can be set are: X'00' Not detectable, invisible, not highlighted, or unchained X'01' Detectable, visible, highlighted, nontransformable, or chained X'02' Transformable

Segment attribute modify

The Segment Attribute Modify order approximates to the GSSATS function. It modifies the attributes that are currently assigned to a segment.

fld len	Content	Meaning
1	X'73'	Segment Attribute Modify order code
1	LEN	Length of following data
1	ATTRIBUTE	The attribute to be modified. The attributes that can be modified are: X'01' Detectability X'02' Visibility X'03' Highlighting X'04' Transformability X'05' Reserved X'06' Chained
1	VALUE	The value to be assigned to the specified attribute. The values that can be set are: X'00' Not detectable, invisible, not highlighted, or unchained X'01' Detectable, visible, highlighted, nontransformable, or chained X'02' Transformable
4	IDENTIFIER	The identifier of the segment for which the attributes are to be modified.

Segment characteristics

The Segment Characteristics order adds more attributes to a segment. It is valid only within the prolog of a segment.

General format:

fld len	Content	Meaning
1	X'04'	Segment Characteristics order code
1	LEN	Length of following data
1	CHID	Identifier code for characteristics
1	DATA	Data

Notes:

1. GDDM sets X'80' in the CHID field. All other values are reserved. Values above X'80' are allocated to applications other than GDDM.
2. GDDM preserves all Segment Characteristics orders with values of other than X'80' in the CHID field. The use of the Segment Characteristics order with a CHID value of X'80' is defined below.

Uses of the segment characteristics order

The order can be used to provide information that corresponds to the GSSORG function.

fld len	Content	Meaning
1	X'04'	Segment Characteristics order code
1	LEN	Length
1	X'80'	Identifier for GDDM
1	X'00'	Reserved
1	X'04'	Segment origin
1	LEN	Length of coordinates
2★	x0	x coordinate origin
2★	y0	y coordinate origin

Segment end

This order corresponds to the GSSCLS function.

fld len	Content	Meaning
1	X'71'	End Segment order code
1	LEN	0, no data

Segment end prolog

The Segment End Prolog order shows the end of the prolog section of each segment. See also page 192.

fld len	Content	Meaning
1	X'3E'	Segment End Prolog order code
1	X'00'	Reserved

Segment position

The Segment Position order approximates to the GSSPOS function. It sets the position of a transformable segment.

fld len	Content	Meaning
1	X'53'	Segment Position order code
1	LEN	Length of following data
2★	X0	x coordinate of the segment position
2★	Y0	y coordinate of the segment position
4	IDENTIFIER	The identifier of the transformable segment that is to be positioned.

Segment start

This order corresponds to the GSSEG function. The order may be truncated immediately after the SEGMENT-ID field. In this case, all segment attributes are taken from the current initial segment attributes as set by the Segment Attribute order (see page 190), or by the GSSATI call.

Segment attribute information can be extended by using a segment prolog. The presence of a segment prolog is shown by a bit in the Segment Start order.

fld len	Content	Meaning
1	X'70'	Segment Start order code
1	X'0C' or X'04'	Length of following data
4	SEGMENT-ID	The identifier to be given to the following segment, or 0 if unnamed. A full-word positive or 0 integer (as in GSSEG).
2	FLAGS	Bit 0 0 = visible 1 = invisible Bit 1 Reserved – must be 1. Bit 2 0 = nondetectable 1 = detectable Bit 3 Reserved – must be 1. Bit 4 0 = no highlighting 1 = highlighting Bit 5 Reserved – must be 1 Bit 6 Reserved – must be 0 Bit 7 Reserved – must be 0 Bit 8 0 = chained 1 = nonchained Bit 9 Reserved – must be 0 Bit 10 Reserved – must be 0 Bit 11 0= no prolog 1 = prolog Bit 12 0= nontransformable 1= transformable Bit 13 Reserved – must be 0 Bit 14 Reserved – must be 0 Bit 15 Reserved – must be 0
2	L2	Length of segment (see Note)
4	X'00000000'	Reserved – must be 0

Notes:

1. GDDM returns the length of a fixed-point GDF segment in the Segment Start order retrieved using GSGET. The length of segment is ignored on GSPUT; segments must be closed by an explicit Segment End order. When the length of a segment cannot be represented as a 2-byte unsigned number, a length of zero is set.
2. The segment attributes in the Segment Start order override the initial segment attributes that are in effect at the time the segment is created. The segment attributes can be altered by GSSATS in the usual way.
3. Within the prolog of a segment, only the following orders are valid:
   - A no-operation (X'00')
   - Comment (X'01')
   - Process Specific Control (X'02')
   - Segment Characteristics (X'04')
   - Pop (X'3F')
   - Marker Scale (X'41')
   - The attribute orders shown below:

Arc Parameters	(X'22' or X'62')
Character Angle	(X'34' or X'74')
Character Box	(X'03' or X'33')
Character-Box Spacing	(X'36' or X'76')
Character Direction	(X'3A' or X'7A')
Character Precision	(X'39' or X'79')
Character Set	(X'38' or X'78')
Character Shear	(X'35' or X'75')
Color and Extended Color	(X'0A', X'4A', X'26', or X'66')
Fractional Line Width	(X'11' or X'51')
Line Type	(X'18' or X'58')
Line Width	(X'19' or X'59')
Marker Box	(X'37' or X'77')
Marker Type	(X'29' or X'69')
Foreground Color Mix	(X'0C' or X'4C')
Model Transform	(X'24' or X'64')
Pattern	(X'28' or X'09')
Pick (Tag) Identifier	(X'43' or X'23').
Segment Viewing Window	(X'27')
Text Alignment	(X'10' or X'50').

Primitive attributes in the segment prolog are treated as being ordinary primitive attributes. GDDM does not create any primitive attributes apart from the transform in the segment prolog. For upward compatibility of GDF, it is advisable not to place primitive attribute orders (other than the Model Transform order) within the segment prolog.

Segment viewing window

This order corresponds to the GSSVL function.

fld len	Content	Meaning
1	*Set* X'27' *Push & set* X'67'	Set Segment Viewing Window
1	LEN	Length of following data
1	X'00'	Reserved
1	MASK	Each bit has a value shown below
	Bit 0 Reserved 1 Reserved 2 xL x left limit B'0' Not included in list of WW values B'1' Included in list of WW values 3 xR x right limit B'0' Not included in list of WW values B'1' Included in list of WW values 4 yB y bottom limit B'0' Not included in list of WW values B'1' Included in list of WW values 5 yT y top limit B'0' Not included in list of WW values B'1' Included in list of WW values 6 zN z near limit B'0' Not included in list of WW values B'1' Included in list of WW values 7 zF z far limit B'0' Not included in list of WW values B'1' Included in list of WW values	
0-16	WW	Window values

Text alignment

This order corresponds to the GSTA function.

fld len	Content	Meaning
1	*Set* X'10' *Push & set* X'50'	Set Text Alignment
1	X'02'	Length of following data
1	Hor TA	Horizontal Text Alignment. Interpreted as follows: X'00' Default X'01' Normal X'02' Left X'03' Center X'04' Right X'FF' Standard
1	Ver TA	Vertical Text Alignment. Interpreted as follows: X'00' Default X'01' Normal X'02' Top X'03' Cap X'04' Half X'05' Base X'06' Bottom X'FF' Standard

Appendix E. Image object definitions

Image data is entered (or "put") into images using the IMAPTS, IMAPT, and IMAPTE calls. The reverse process of retrieving image data from an image (or "get") is done by using the IMAGTS, IMAGT, and IMAGTE calls. If the image is self-defining, the "put" process is a transfer operation and so a projection can be used.

The "get" process is always a transfer operation.

Images can also be entered into and retrieved from the application program using the IMASAV and IMARST calls. These store complete images into, and retrieve complete images from, a database or GDDM object library.

For detailed information on all image calls, see the *GDDM Base Programming Reference, Volume 1*.

Formats and compression types

Image data must have a valid combination of format and compression type. The following image data formats are allowed:

1 Unformatted
2 3193 data stream format
3 Composed-page printer format.

The following image data compression types are allowed:

1 Uncompressed
2 MMR (IBM 8815)
3 IBM 4250
4 IBM 3800.

Only specific combinations of format and compression are allowed; these are indicated by an "X" in the following table:

Table 29. Valid combinations of format and compression

	Unformatted	3193 DSF	CPPF
Uncompressed	X	X	
MMR (IBM 8815)	X	X	X
IBM 4250			X
IBM 3800			X

Notes:

1. MMR = modified-modified read format (8815 compatible)
2. 3193DSF = 3193 data stream format
3. CPPF = composed-page printer format.

3193 data stream and composed-page printer formats

Self-defining data comprises a data stream containing a list of image objects. These are indicated in Figure 11 on page 196 by suitable mnemonics.

For entry of formatted data (IMAPT) into either of these formats, GDDM processes only the first image object in the data stream (from BIC to EIC for 3193 data stream, and from BIM to EIM for composed-page printer format) and ignores all others as shown in Figure 11 on page 196.

For retrieval of formatted data (IMAGT), GDDM constructs an image object either in 3193DSF or in CPPF.

For 3193DSF data, GDDM constructs BIC, ISP, IEP, ID, and EIC structured fields, but it does not return ILP.

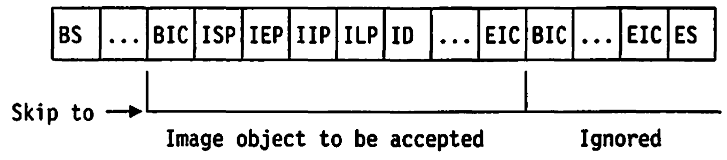
For CPPF data, GDDM constructs BIM, IID, IRD, and EIM structured fields, but it does not return IOC – except for 120 ppi 3800 image data, which is returned as 240 ppi with a scale factor of 2 in the IOC.

For more information on the 3193 data stream format, see the *IBM 3193 Display Station Description* manual.

For more information on the composed-page printer format, see the

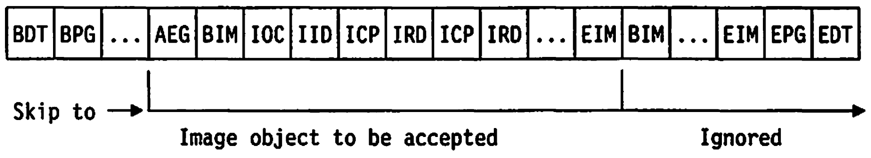
- *Print Management Facility: User Guide and Reference* manual
- *Composed Document printing Facility: General Information* manual.

3193DSF data stream



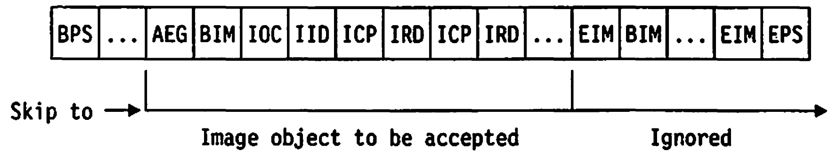
BS	:	Begin Segment
BIC	:	Begin Image Content
ISP	:	Image Size Parameter
IEP	:	Image Encoding Parameter
IIP	:	Image IDE Size Parameter
ILP	:	Image LUT-ID Parameter
ID	:	Image Data
EIC	:	End Image Content
ES	:	End Segment

CPPF document



BDT	:	Begin Document
BPG	:	Begin Page
AEG	:	Active Environment Group (optional)
BIM	:	Begin Image
IOC	:	Image Output Control (optional)
IID	:	Image Input Descriptor
ICP	:	Image Cell Position
IRD	:	Image Raster Data
EIM	:	End Image
EPG	:	End Page
EDT	:	End Document

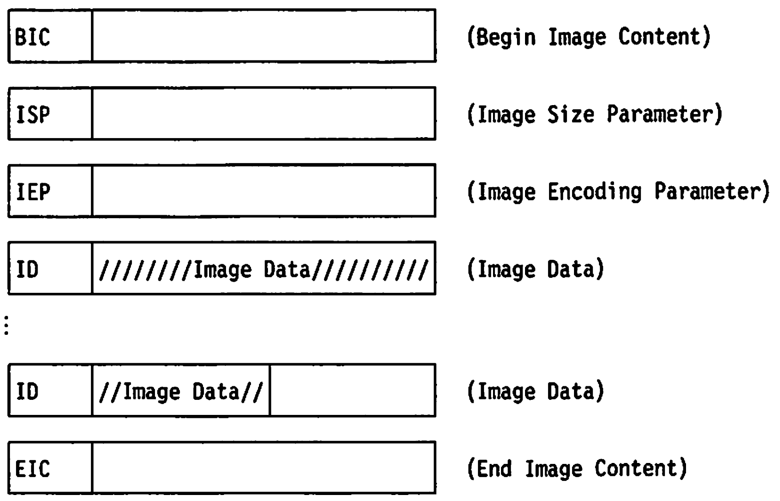
CPPF page segment



BPS	:	Begin Page Segment
EPS	:	End Page Segment

Figure 11. Accepted data streams (3193DSF and CPPF)

3193DSF output data stream



CPPF output data stream

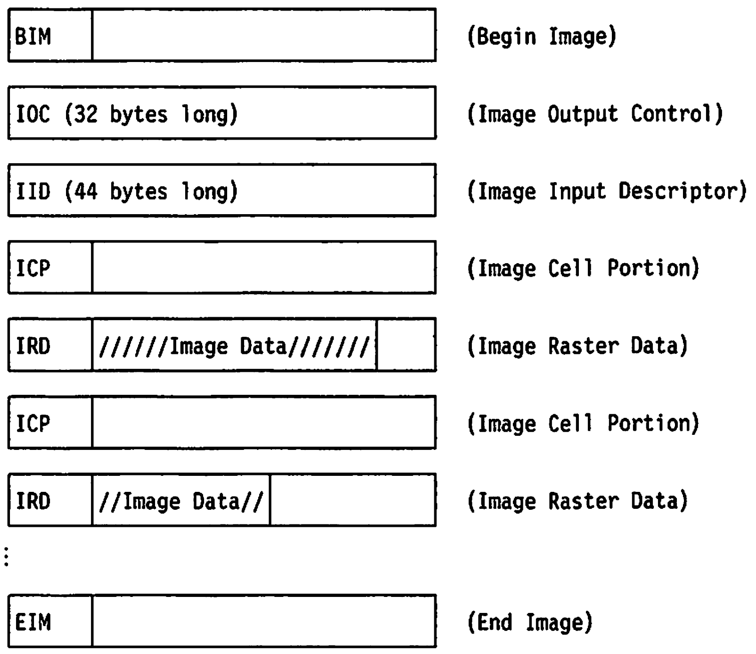


Figure 12. IMAGT data streams from GDDM

Unformatted data

Unformatted binary image data is defined as follows:

Compression	Uncompressed data; 1 bit per pixel, 8 bits per byte.
	Compressed data; as defined by the compression algorithm.
Padding	Uncompressed data; the end of each row is padded to the byte boundary, if it does not fall on one.
	Compressed data; padding is defined by the compression algorithm.
Structure	No headers, trailers, or imbedded control fields, other than those defined by the compression algorithm The pixels (and trailing pad values) occupy contiguous storage.
	Row 0 (that is, the top of the picture) comes first, followed by the other rows in order.
	Within a row, the pixels with the lower index (that is, the left of the picture) come first.

Objects in the GDDM object library

GDDM image introduces some new objects types to the GDDM object library. The default names for these, and the relevant GDDM calls are:

Table 30. Default names for image object types

Object type	Name	GDDM calls
Projection	ADMPROJ	IMPSAV, IMPRST
Image	ADMIMG	IMASAV, IMARST

The relationship of these names to the file descriptors of the various operating systems that GDDM supports is the same as for existing GDDM objects and is defined in these chapters:

Chapter 2, "Using GDDM under CICS/VS" on page 7
Chapter 3, "Using GDDM under IMS/VS" on page 23
Chapter 4, "Using GDDM under MVS/XA" on page 31
Chapter 5, "Using GDDM under TSO" on page 33
Chapter 6, "Using GDDM under VM/CMS" on page 41.

Appendix F. Symbol-set formats

This appendix describes the formats for image symbol sets (ISS) and vector symbol sets (VSS) held on files (or passed as parameters in symbol-set manipulation calls).

In either case, definitions start with a two-byte field that gives the total length of the definitions (including the length field). Then follows one or more definition components, each of which is in one of the formats described below (depending upon whether the definitions relate to a dot-image symbol set (ISS) or to a vector symbol set (VSS)).

The following rules must be observed. For the purpose of these rules, pattern and marker definitions are treated as MODE=2.

1. ISS and VSS components must not be mixed within a definition. ISS definitions cannot be loaded as MODE=3; VSS definitions can be loaded only as MODE=3.
2. For ISS definitions, the following considerations apply to the width (P) and depth (Q) of the cell matrix in display points:

   When either is specified as zero or is not specified, it is assumed to be equal to the cell width or depth of the actual device (except for format type '00001'B, where P is assumed to be 9).

   When the format type is '00001'B, P (specified or assumed) must be 9.

   When P is not a multiple of eight for row-loading format (type '00011'B), the storage occupied by each row must be padded on the right with zero bits to the next byte boundary.

   Similarly, when Q is not a multiple of eight for column-loading format (type '00101'B), the storage occupied by each column must be padded at the bottom with zero bits to the next byte boundary.
3. For MODE=1 definitions, the data format must be one that is supported by the actual device to which the definitions are to be loaded. One or more components may be specified, either to define different color planes for a multicolored definition, or to reduce the total length in cases where only widely-scattered character codes are to be loaded. Although checks are made, it is possible for a symbol set definition to pass these checks and still be rejected by the device or controller when the definitions are actually transmitted.
4. For MODE=2 definitions, only one component may be specified for a monochrome definition, or exactly three components (one for each color) for a multicolored definition. In the latter case, the starting character code and the number of codes defined must be the same for all three color planes.
5. For MODE=3 definitions (VSS), only one component may be specified.
6. The CLEAR bit is not supported by GDDM, although its setting is not altered by GDDM before transmitting MODE=1 definitions.

Image symbol set component format

Table 31. Image symbol set component format

Byte	Field length	Content	Meaning
0	2	LENGTH	Total length of structure (including LENGTH field).
2	1	TYPE	Type of symbol set X'06' = Image Symbol Set.
3	1	FLAGS	Bit 0 (EXTENDED) is on if extended format of definition. Bit 1 (CLEAR) is on if all definitions in the specified symbol set (plane) are to be cleared before processing the definitions. Bit 2 (SKIPSUPP) is on if skip is to be suppressed after printing a row that contains any symbol from this ISS. Bits 3 through 7 (TYPE) define the data format for the definitions (see Note 1): '00001'B 18-byte form: the first two bytes contain a 16-bit vertical slice; the following 16 bytes contain 8-bit horizontal slices; for a 9 by 12 cell, the last 4 bytes contain binary zero. '00011'B Row loading: bits within each row go from left to right, padded to a byte boundary; successive rows are from top to bottom. '00101'B Column loading: bits within each column go from top to bottom, padded to a byte boundary; successive columns are from left to right.
4	1		Reserved.
5	1	CP0	Starting character code within this symbol set (in range X'41' through X'FE').
6	1		Reserved.
7	1	LEXT (see Note 2)	Length of extended parameters; gives the length of fields from and including LEXT to the end, but excluding CDEF. Must be specified as 6, if present.
8	1	EXTFLAGS (see Note 2)	Bit 0 (APA) is on if all points in the cell are not addressable. Bit 1 (CB) is on for no LCID compare. Bit 2 (OB) is on for no operator selectability. Bits 3 through 7 '00000'B – Reserved.
9	1	P (see Note 2)	Number of x units in dot matrix.
10	1	Q (see Note 2)	Number of y units in dot matrix.
11	1	SUBSN (see Note 2)	Subsection identifiers, as follows: X'00' one-byte codes X'42' — X'7F' subsection identifiers for two-byte coded data Other reserved.
12	1	COLOR (see Note 2)	Bits 0 through 4: reserved Bits 5 through 7: color planes to be loaded as follows: '000'B all planes '001'B blue plane '010'B red plane '100'B green plane Other reserved.
7 or 13	V	CDEF (CP0-CPn)	Symbol definitions, starting at character code CP0, in ascending order, and in the format defined by Byte 3.

Notes:

1. Data format definitions:
   TYPE '00001'B is equivalent to cell format 1 for displays,
   TYPE '00011'B is equivalent to cell format 3 for graphics,
   TYPE '00101'B is equivalent to cell format 2 for printers.

   "Cell formats" are specified in the Image Symbol Editor. For more information, see the *GDDM Image Symbol Editor* manual.
2. Present only if Bit 0 (EXTENDED) of FLAGS is on.

Vector symbol set component format

Table 32. Vector symbol set component format

Byte	Field length	Content	Meaning
0	2	LENGTH	Total length of structure (including LENGTH field).
2	1	TYPE	type of symbol set X'01' = Vector Symbol Set.
3	1	FLAGS	Bit 0 (EXTENDED) is on if definition is in extended format. Bit 1 is ignored. Bit 2 (SHADED) is on if all symbols defined are to be shaded using the default shading pattern. This has the effect of surrounding each symbol definition implicitly by GSAREA and GSENDA. Bits 3 through 7 (TYPE) define the data format for the definitions. For VSS, each symbol is formed by lines and (for type 3) curves. Three types are defined: '00001'B type 1 '00010'B type 2 '00011'B type 3.
4	1		Reserved (must be zero).
5	1	CP0	Starting character code within this symbol set (in range X'00' through X'FF').
6	1	FLAGS	Bit 0 (PROPORTIONAL SPACING) is on if each index entry is extended by a halfword value specifying the width of each symbol. If this flag is off, each symbol has the width P. Valid only for type-3 definitions. Bit 1 (LINES ONLY) is on if only the following GDF orders are contained within the symbol definitions: {extended order} line; {extended order} line at current position; end of data. Valid only for type-3 definitions. Bits 2 through 7 reserved (must be zero).
7	1	LEXT (see Note)	Length of extended parameters; gives the length of fields from and including LEXT to the end, but excluding CDEF. Minimum value is 1. Maximum value is 9.
8	1	(see Note)	Reserved (must be zero).
9	2	P (see Note)	Range of x (0 through P). If this operand is not present, or it is specified as 0, then the value 15 is assumed.
11	2	Q (see Note)	Range of y (0 through Q). If this operand is not present, or it is specified as 0, then the value 15 is assumed.
13	2	(see Note)	Reserved (must be zero).
15	1	CPn *	Last character code within this symbol set. If this operand is not present, X'FE' is assumed. CPn must not be less than CP0.
7 to 16	V	CDEF(CP0-CPn)	Symbol definitions, starting a character code point CP0, ascending order. See below for format types 1, 2, and 3.

Note: Present only if bit 0 (EXTENDED) of FLAGS is on.

Format of symbol definitions

P and Q together define the character box within which a normal symbol fits. The bottom left-hand corner of the box is (0,0) and the top right-hand corner is (P,Q).

Undefined character codes are generally displayed as a blank, but see the description of the GSCHAR call in the *GDDM Base Programming Reference, Volume 1*.

Characters are always drawn using the default line type and line width, and with the default area pattern.

Type 1
The definitions start with an index of (CPn + 1 - CP0) two-byte values. Each index value is the offset from the start of CDEF (that is, from the start of the index) of the start of the definition of the corresponding character code (CP0 through CPn). This index must always be present in its entirety, even if not all the characters in the code range are defined. The maximum length of the index, if CP0 is specified as X'00', and CPn as X'FF', is therefore 256 by 2 bytes. Undefined values should be represented by a zero in the index.

Each character is defined as a series of points that define the shape of the character. Each point defines either a line from the preceding point, or a move to be performed to that point. The endpoints of each line (or move) are given by an (x,y) coordinate pair of signed relative values (relative to the previous coordinate, or to the bottom left-hand of the character box for the first coordinate pair). Each coordinate pair occupies two bytes (one byte for the x coordinate, and one for the y). If the first stroke is a line rather than a move, the line is drawn from the bottom left-hand corner of the box. The top bit of the y-coordinate byte is on if the stroke to that point is visible (that is, line rather than move); after the last coordinate pair, two bytes of all 1 bits indicates the end of the definition for that symbol. This format of an individual code point symbol is:

0	DX1	0	DY1	2 bytes
0	DX2	VIS	DY2	2 bytes
.			.	
.			.	
0	DXN	VIS	DYN	2 bytes
X'FFFF..'				2 bytes

Type 2
For type-2 format, the endpoints of each line (or move) are given by a (dx,dy) coordinate pair of signed relative values. Each coordinate pair occupies four bytes (two bytes for the x coordinate, and two for the y), and is preceded by two bytes of flag bits, so that each point requires a total of six bytes. Each symbol consists of a series of these point definitions, defining the lines and moves needed to draw it. The start is from the bottom left-hand of the character box (0,0). The last point for a particular symbol is recognized by means of a flag, in the flag halfword.

One of the flags designates "branch." This means that, instead of the dx halfword, a point number is specified, to which to branch for the remaining definitions for that symbol. The actual offset within CDEF is given by:

```
offset=point-number*6
```

In the case of a branch, the dy value is ignored.

As with type 1, CDEF starts with an index, with (CPn+1-CP0) entries corresponding to symbol codes CP0 through CPn. Each entry is a branch, as defined above, which in effect defines the starting position of a symbol.

This format of an individual point is illustrated below.

	E	B	M	R	2 bytes
DX or point-number					2 bytes
DY					2 bytes

where:

E bit (bit 12)	is on if this is the last point for the current symbol
B bit (bit 13)	is on if this is a branch
M bit (bit 14)	is on if this is a move, not a line
R bit (bit 15)	is ignored.

Type 3
The definitions start with an index, just the same as the index for type 1. However, if the "PROPORTIONAL SPACING" flag in the header is set, each two-byte index offset entry is followed by a two-byte signed symbol width. This makes the entire index twice its normal size. Each symbol width value must be in the range

```
(-P < w < 0) or (0 < w <= P)
```

where the values 0 and -P are reserved. If the width is positive, the boundaries of the symbol (for spacing purposes only) are the left-hand side of the box, and a line "w" to the right of the left-hand side. If the width is negative, the boundaries are the right-hand side, and a line "−w" to the left of the right-hand side. Thus a width of "+P" is the default (full) width.

A type-3 symbol definition consists of a series of GDF (graphics data format) orders. These typically specify lines and curves that make up the symbol. The orders for a given symbol are terminated by an end-of-data marker, which is a single byte with the value X'FF'. All orders should be a complete number of half words, and, for performance reasons, should be aligned on a half word boundary.

See Appendix D, "GDF order descriptions" on page 165 for a description of GDF orders. A symbol generated by the Vector Symbol Editor typically uses the following orders:

- Line (X'C1')
- Line at Current Position (X'81')
- Fillet (X'C5')
- Fillet at Current Position (X'85')
- Area (X'68').

Whenever a type-3 symbol is processed, a particular type of coordinate data is assumed. This depends on the values of P and Q. If both P and Q are less than 128, the default is one-byte signed absolute coordinates. If either P or Q are greater than 127, the default is two-byte signed absolute coordinates.

If the SHADED flag in the header is set, each symbol is drawn, using the default shading pattern, as though that symbol were enclosed in "Begin Area" and "End Area" orders. These orders are **implicit**. If the SHADED flag in the header is **not** set, individual shaded symbols should include an **explicit** "Begin Area" order and an explicit "End Area" order (just before the X'FF' marker).

Appendix G. Device characteristics tokens

This appendix describes the device characteristic tokens (usually called device tokens) supplied by GDDM. Many Input/Output tasks in GDDM can be done without using device tokens at all; however, there are some instances where they should be used. For example:

- Under IMS/VS, to define the data base that links the characteristics of terminals with logical terminal names, and so determine the type of data stream that GDDM sends.
- To specify some special types of device, such as the IBM 4250 page printer, in a DSOPEN call.
- To override the information obtained by GDDM about a particular device so that device information is taken from the token rather than from the device itself, which is the usual source.

GDDM-supplied device tokens

The GDDM-supplied device tokens are shown in the tables in this appendix.

Note: Your installation may have changed the GDDM-supplied device tokens or created tokens of their own.

The meanings of the tokens are shown as the macro definitions used to create them. You may need to study the meanings of the macro operands to understand the tokens. The operands are explained in the *GDDM Installation and System Management* manual that applies to the subsystem in use.

Creating your own device tokens

The GDDM-supplied device tokens are designed to cater for most requirements. For information, see the *GDDM Installation and System Management* manual that applies to the subsystem in use.

Table 33 (Page 1 of 2). GDDM-supplied device tokens for queriable terminals and printers

This set of token definitions is part of ADMLSYS1. The "buffer code" corresponds to the code in the **dev** parameter of the ADMM3270 macro. A device token of ★ is sufficient for printers if they are directly-attached.

Locally-attached 3179 Models G1 and G2 displays and 3192-G displays	
L3179G	3179-G, 32 rows by 80 columns
L3179GM	3179-G, 32 rows by 80 columns with mouse
Locally-attached 3270-PC displays	
L3270PC	3270-PC displays
Locally-attached 3270-PC/G work stations	
L5279A1	3270-PC/G, 32 rows by 80 columns
L5279A1M	3270-PC/G, 32 rows by 80 columns, with mouse
L5279A1T	3270-PC/G, 32 rows by 80 columns, with tablet
L5279A2	3270-PC/G, 49 rows by 80 columns
L5279A2M	3270-PC/G, 49 rows by 80 columns, with mouse
L5279A2T	3270-PC/G, 49 rows by 80 columns, with tablet
ADMKPCA1	copy of L5279A1, generated for use by the ADMUPC utility for dummy devices
Locally-attached 3270-PC/GX work stations (32 rows, 80 columns)	
L5379CS	3270-PC/GX, color
L5379CSM	3270-PC/GX, color, with mouse
L5379CST	3270-PC/GX, color, with tablet
L5379MS	3270-PC/GX, monochrome
L5379MSM	3270-PC/GX, monochrome, with mouse
L5379MST	3270-PC/GX, monochrome, with tablet
L5379CD	3270-PC/GX, color, dual screen
L5379CDM	3270-PC/GX, color, dual screen, with mouse
L5379CDT	3270-PC/GX, color, dual screen, with tablet
L5379MD	3270-PC/GX, monochrome, dual screen
L5379MDM	3270-PC/GX, monochrome, dual screen, with mouse
L5379MDT	3270-PC/GX, monochrome, dual screen, with tablet
Locally-attached 3278 and 3279 displays	
No PS compression is specified on the assumption that this is most efficient for a channel-attached controller	
L78A2	3278, buffer code 2
L78A3	3278, buffer code 3
L78A4	3278, buffer code 4
L78A5	3278, buffer code 5
L79A2	3279, buffer code 2
L79A3	3279, buffer code 3
Plotters attached to 3179 Models G1 and G2 or 3192-G displays	
L3179G80	6180 plotter
L3179G82	6182 plotter

Table 33 (Page 2 of 2). GDDM-supplied device tokens for queriable terminals and printers

L3179G84	6184 plotter		
L3179G86	6186 plotter		
L3179G71	7371 plotter		
L3179G72	7372 plotter		
Plotters attached to 3270-PC/G and /GX work stations			
L6180	6180 plotter		
L6182	6182 plotter		
L6184	6184 plotter		
L6186	6186 plotter		
L7371	7371 plotter		
L7372	7372 plotter		
L7374	7374 plotter		
L7375	7375 plotter		
Plotters attached to 5550-family work stations			
L5550G71	7371 plotter		
L5550G72	7372 plotter		
Locally-attached 3268, 3287, and 3262 printers, with no compression			
L68	3268, LU type 3 protocols		
L87	3287, (four color only) LU type 3 protocols		
L87S	3287, (four color only) LU type 1 (SCS) protocols		
L3262	3262 belt printer		
Remotely-attached 3278 and 3279 displays			
PS compression is specified on the assumption that this is most efficient for a link-attached controller			
R78A2	Remote 3278, buffer code 2		
R78A3	Remote 3278, buffer code 3		
R78A4	Remote 3278, buffer code 4		
R79A2	Remote 3279, buffer code 2		
R79A3	Remote 3279, buffer code 3		
Remotely-attached 3287 printers, with compression			
R87	Remote 3287, (four color only) LU type 3 protocols		
R87S	Remote 3287, (four color only) LU type 1 (SCS) protocols		
3812 Model 2 with 3270 attachment feature			
This is an IPDS (Intelligent printer data stream) printer connected as a family 1 or a family 2 device			
X3812A4	LU type 0 protocols	A4 paper	93 rows by 82 columns
X3812Q	LU type 0 protocols	Quarto paper	88 rows by 85 columns
S3812A4	LU type 1 (SCS) protocols	A4 paper	93 rows by 82 columns
S3812Q	LU type 1 (SCS) protocols	Quarto paper	88 rows by 85 columns
4224 printers, LU type 0 protocols			
X4224SS	64K byte RAM, no loadable alphanumeric symbol sets, 68 rows by 132 columns		
X4224SE	512K byte RAM, up to 6 loadable alphanumeric symbol sets, 68 rows by 132 columns		
X4224QS	64K byte RAM, no loadable alphanumeric symbol sets, 88 rows by 85 columns		
X4224QE	512K byte RAM, up to 6 loadable alphanumeric symbol sets, 88 rows by 85 columns		
4224 printers, LU type 1 (SCS) protocols			
S4224SS	64K byte RAM, no loadable alphanumeric symbol sets, 68 rows by 132 columns		
S4224SE	512K byte RAM, up to 6 loadable alphanumeric symbol sets, 68 rows by 132 columns		
S4224QS	64K byte RAM, no loadable alphanumeric symbol sets, 88 rows by 85 columns		
S4224QE	512K byte RAM, up to 6 loadable alphanumeric symbol sets, 88 rows by 85 columns		
Note: These tokens require the 4224 printer to be set to 10 characters per inch and 8 lines per inch			

Table 34. GDDM-supplied device tokens for Kanji devices, and 8775 and 3290 displays.

This set of token definitions is generated as part of ADMLSYS1	
KANJI displays and printers	
KANJI	Kanji 3278 Model 2 display
KANJIP	Kanji 3283 printer, LU type 1 (SCS) protocols
K78A2	Kanji 3278 Model 2 display
K83S	Kanji 3283 printer, LU type 1 (SCS) protocols
K83	Kanji 3283 Model 2 printer, LU type 3 protocols
L5550A	Kanji 5550 display, non-graphics
L5553A	Kanji 5553 printer
5550-family work stations (non-graphics)	
L5550A	Japanese 3270 emulation display, monochrome
L5553A	Japanese 3270 emulation printer, LU type 3 protocols
L5553AI	Japanese 3270 emulation display, LU type I (SCS) protocols
L5550G4	Japanese 3270-PC Version 4.0 display, monochrome
L5550H4	Japanese 3270-PC Version 4.0 display, color
L5553B34	Japanese 3270-PC Version 4.0 printer, type 3 protocols
L5553BI4	Japanese 3270-PC Version 4.0 printer, type 1 protocols
5550-family work stations (graphics)	
L5550GC2	Japanese 3270-PC/G Version 2.0 display, 16-dot font
L5550GH2	Japanese 3270-PC/G Version 2.0 display, 24-dot font
L5550GC3	Japanese 3270-PC/G Version 3.0 display, 16-dot font
L5550GH3	Japanese 3270-PC/G Version 3.0 display, 24-dot font
L5550GC5	Japanese 3270-PC/G Version 5.0 display, 16-dot font
L5550GH5	Japanese 3270-PC/G Version 5.0 display, 24-dot font
The following tokens are produced automatically by the ADMM3270 macro and cannot be altered	
8775 displays with PS and partitions	
ADMK7510	8775, 12 rows by 80 columns
ADMK7520	8775, 24 rows by 80 columns
ADMK7530	8775, 32 rows by 80 columns
ADMK7540	8775, 43 rows by 80 columns
8775 displays with partitions and scrolling	
ADMK751S	8775, 12 rows by 80 columns
ADMK752S	8775, 24 rows by 80 columns
ADMK753S	8775, 32 rows by 80 columns
ADMK754S	8775, 43 rows by 80 columns
3290 displays	
ADMK9020	3290, 24 rows by 80 columns
ADMK9030	3290, 32 rows by 80 columns
ADMK9040	3290, 43 rows by 80 columns
ADMK9050	3290, 27 rows by 132 columns
ADMK9060	3290, 62 rows by 160 columns

Table 35. GDDM-supplied device tokens for nonqueriable terminals and printers (family 1)

Nonqueriable display terminals	
ADMK7810	3278 Model 1
ADMK781A	3278 Model 1 with APL
ADMK7820	3278 Model 2
ADMK782A	3278 Model 2 with APL
ADMK7830	3278 Model 3
ADMK783A	3278 Model 3 with APL
ADMK7840	3278 Model 4
ADMK784A	3278 Model 4 with APL
ADMK7850	3278 Model 5
ADMK785A	3278 Model 5 with APL
Nonqueriable printers	
ADMKQUEP	default token for family-2 (queued) printers
ADMK8710	3287 Model 1
ADMK871A	3287 Model 1 with APL

Table 36. GDDM-supplied device tokens for GDDM-PCLK displays, printers, and plotters

GDDM-PCLK work station configurations	
LPCM	PCLK display with CGA, monochrome, 24 rows by 80 columns, (640 by 200 pixels)
LPCC1	PCLK display with EGA, 16-color, 24 rows by 80 columns, (640 by 200 pixels)
LPCC2	PCLK display with EGA, 16-color, 24 rows by 80 columns, (640 by 350 pixels)
LPCC3	PCLK display with Personal System/2 display adapter, 16-color, 24 rows by 80 columns, (640 by 480 pixels)
LPCC4	PCLK display with Personal System/2 display adapter 8514/A, 16-color, 24 rows by 80 columns, (1024 by 768 pixels)
LPCC5	PCLK display with Personal System/2 display adapter MCGA, 2-color, 24 rows by 80 columns, (640 by 480 pixels)
GDDM-PCLK work-station configurations with a locally attached printer	
LPC3852	PCLK display with 3852 color ink-jet printer
LPC4201	PCLK display with 4201 proprinter
LPC42012	PCLK display with 4201 Model 2 proprinter
LPC4202	PCLK display with 4202 proprinter XL
LPC4207	PCLK display with 4207 proprinter X-24
LPC4208	PCLK display with 4208 proprinter XL-24
LPC5152	PCLK display with 5152 monochrome graphics printer
LPC5182	PCLK display with 5182 color impact printer
LPC5201	PCLK display with 5201 Quietwriter
LPC5202	PCLK display with 5202 Quietwriter-III
GDDM-PCLK work-station configurations with a locally attached plotter	
LPC7371	PCLK display with 7371 2-pen plotter
LPC7372	PCLK display with 7372 6-pen plotter
LPC7374	PCLK display with 7374 8-pen plotter
LPC7375	PCLK display with 7375 8-pen plotter
LPC6180	PCLK display with 6180 8-pen plotter
LPC6182	PCLK display with 6182 8-pen plotter
LPC6184	PCLK display with 6184 8-pen plotter
LPC6186	PCLK display with 6186 8-pen plotter

Table 37. GDDM-supplied device tokens for system printers (family 3)

System printers	
ADMKSYSP	default for non-3800 printers
Various non-3800 printers	
S1403N6	1403, 66 rows by 85 columns, 6 lines per inch
S1403N8	1403, 88 rows by 85 columns, 8 lines per inch
S1403W6	1403, 66 rows by 132 columns, 6 lines per inch
S1403W8	1403, 88 rows by 132 columns, 8 lines per inch
Various 3800 printers	
S3800N6	3800, 60 rows by 85 columns, 6 lines per inch
S3800N8	3800, 80 rows by 85 columns, 8 lines per inch
S3800N12	3800, 120 rows by 85 columns, 12 lines per inch
S3800W6	3800, 60 rows by 136 columns, 6 lines per inch
S3800W8	3800, 80 rows by 136 columns, 8 lines per inch
S3800W12	3800, 117 rows by 136 columns, 12 lines per inch
S3800N6S	3800, 45 rows by 110 columns, 6 lines per inch
S3800N8S	3800, 60 rows by 110 columns, 8 lines per inch
S3800W6S	3800, 45 rows by 136 columns, 6 lines per inch
S3800W8S	3800, 60 rows by 136 columns, 8 lines per inch

Note: Device tokens for 3800 printers can also be used for 3820 printers and 3812 printers except for the 136 columns width

Table 38. GDDM-supplied device tokens for page printers (family 4)

ADMKHRIG — Default token for family-4 (page) printers

This token is used when the DSOPEN device token is given as "★," and no nicknames are in force to alter it. It describes a 4250 printer at 600 pixels per inch, 6 pixels per unit line width, with an output image width of 8.5 inches, and a depth of 11.0 inches.

3800, 3812, or 3820 printer at 120 pixels per inch, 3 pixels per unit line width	
IMG120	3800, width 8.5, depth 11.0 inches
3800 printer at 240 pixels per inch, 3 pixels per unit line width	
IMG240X	3800, width 13.9, depth 12.5 inches
3800, 3812, or 3820 printer at 240 pixels per inch, 3 pixels per unit line width	
IMG240	3800, width 8.5, depth 11.0 inches
LETTER	3800, width 8.5, depth 11.0 inches
LEGAL	3800, width 8.5, depth 14.0 inches
A4	3800, width 8.3, depth 11.7 inches
EXECUTIV	3800, width 7.5, depth 10.5 inches
3800 printer at 240 pixels per inch, 1 pixel per unit line width	
FINE240	3800, width 13.9, depth 12.5 inches
3800 AFPDS printer at 240 pixels per inch	
P38PPN1	3800, width 8.5, depth 10.0 inches, 1 pixel per unit line width
P38PPN3	3800, width 8.5, depth 10.0 inches, 3 pixels per unit line width
IMG2401	3800, width 8.5, depth 11 inches, 1 pixel per unit line width
4250 printer at 600 pixels per inch, 6 pixels per unit line width	
IMG85	4250, width 8.5, depth 11.0 inches
IMG117	4250, width 11.7, depth 10.0 inches
IMG600X	4250, width 11.7, depth 14.0 inches
IMG600Y	4250, width 17.0, depth 11.0 inches
IMGA3X	4250, width 297.0, depth 420.0 millimeters
4250 printer at 600 pixels per inch, 1 pixel per unit line width	
FINE600	4250, width 11.7, depth 14.0 inches

Note that the values given in DSOPEN's processing option groups 5, 8, and 9 are overridden when the following tokens are used:

Canonical (unformatted) bit image output	
CAN512	Unformatted bit image output, 512 by 512 pixels
CAN1024	Unformatted bit image output, 1024 by 1024 pixels

Table 39. GDDM-supplied device tokens for image display and scanners

Image display	
L3193	
Image display with attached scanner	
L319317	3117 flat-bed scanner
L319318	3118 sheet-feed scanner

Appendix H. Call format descriptor module

A GDDM call format descriptor module, which is independent of the subsystem under which GDDM is running, is provided with GDDM. The module contains information for each GDDM Call statement, describing the number of parameters required on the call, and the type of each parameter.

The address of the call format descriptor module can be acquired by an application program by using the CALLINF external defaults option in a SPINIT call; see Chapter 1, "Customizing your program and its environment" on page 1, and "Initialization" on page 103.

The call format descriptor module is in three sections. The first section provides an address table locating the descriptors for call statements with a given first two characters. The second section provides descriptors for all GDDM calls that have the same first two letters in their name, and the third section provides descriptors for the parameters for a specific call statement.

The address table

The address table is located at offset 0 from the entry point of the module. The format of the address table is:

Table 40. Call format descriptor module – address table

Field name	Field offset	Field length
RCPPIDEN	0	8
RCPPVERS	8	4
RCPPTABP(1)	C	8
..RCPPFTWO(1)	C	2
—	E	2 (reserved)
..RCPPSPTR(1)	10	4
RCPPTABP(2)	14	8
..RCPPFTWO(2)	14	2
—	16	2 (reserved)
..RCPPSPTR(2)	18	4
.	1C+	.
.		.
.		.
RCPPTABP(n)	4+(8*n)	8
..RCPPFTWO(n)	4+(8*n)	2
—	4+(8*n+2)	2 (reserved)
..RCPPSPTR(n)	4+(8*n+4)	4

RCPPIDEN
Table identifier containing the character string "ADMADCP ". Note the mandatory terminating blank.

RCPPVERS
A full-word integer identifying the version number of the Call Format Descriptor Module. If this field is set to '1', the extended Call Descriptor Table is present. Applications that use the calls in the extended table should test the version code, and if this is set to '1', they should scan the table until they reach X'FFFE'. Applications that do not use the extension scan the Call Format Descriptor Table only until they reach X'FFFF'.

RCPPFTWO(n)
A two-byte character string containing the first two characters of the GDDM call statements described by the Call Descriptor Table addressed by RCPPSPTR(n).

A value of X'FFFF' indicates the end of the address table.

RCPPSPTR(n)
The address of the Call Descriptor Table, which defines all GDDM call statements that start with the two characters identified by field RCPPFTWO(n).

The call descriptor table

The Call Descriptor Table is addressed from the address table shown in Table 40. There is one entry in the Call Descriptor Table for each GDDM call statement for which the first two letters are the same as in the address table entry used to locate the descriptor table. The format of the Call Descriptor Table is shown in Table 41 on page 210.

RCPPLENG(n)
The length of this entry in the Call Descriptor Table. The next entry in the table is at offset RCPPLENG from this field.

A value of X'FFFF' indicates the end of the version 0 call descriptor table. If RCPPVERS is set to 1, the call descriptor table extension is present; a value of X'FFFE' indicates the end of the call descriptor table extension.

RCPPFLAG(n)
A set of flags to indicate features of the CALL statement.

RCPPIO

0	The call cannot cause a terminal I/O.
1	The call may cause I/O to the terminal. This flag is set if any of the flags RCPPGIO, RCPPDIO, or RCPPIIO are set to 1.

RCPPOGP

0	The call is available in the base function of GDDM.
1	The call is available only through another licensed program in the GDDM family of licensed programs.

RCPPGIO

0	No I/O is performed to the device (unless either flag RCPPDIO or flag RCPPIIO is set to 1).
1	The call causes I/O to the terminal. For example, FSFRCE outputs data to the device, ASREAD outputs data and awaits terminal operator input.

Table 41. Call format descriptor module – call descriptor table

Field name	Field offset	Field length
RCPPSTAB(1)	0	see RCPPLENG(1)
.RCPPLENG(1)	0	2
.RCPPFLAG(1)	2	2
..RCPPIO (1)	1xxx xxxx	bit within RCPPFLAG
..RCPPOGP(1)	x1xx xxxx	bit within RCPPFLAG
..RCPPGIO(1)	xx1x xxxx	bit within RCPPFLAG
..RCPPDIO(1)	xxx1 xxxx	bit within RCPPFLAG
..RCPPIIO(1)	xxxx 1xxx	bit within RCPPFLAG
..RCPPCPAG(1)	xxxx x1xx	bit within RCPPFLAG
..RCPPHCNG(1)	xxxx xx1x	bit within RCPPFLAG
..RCPPAPLS(1)	xxxx xxx1	bit within RCPPFLAG
.RCPPNAME(1)	4	4
.RCPPRCP(1)	8	4
.RCPPDESC(1)	C	see RCPPNARG
RCPPSTAB(2)	0+RCPPLENG(1)	see RCPPLENG(2)
.RCPPLENG(2)	0+RCPPLENG(1)	4
)		
) as above,	.	.
) with ...(2)	.	.
)	.	.
RCPPSTAB(n)	–	–

RCPPDIO

0 No data-set I/O that causes terminal activity can result from this call.

1 The call may cause I/O activity to a data set. It may result in a terminal I/O operation on some subsystems; for example, a password prompt in opening a data set.

RCPPIIO

0 Data is never sent to the terminal for the call (unless either flag RCPPGIO or flag RCPPDIO is set to one).

1 The call may cause data to be output to the terminal by GDDM if specific conditions are met. Currently, this can only occur if the device is a 3270-PC/G or 3270-PC/GX work station, and the application is drawing graphics primitives outside segments. Implicit I/O occurs whenever too much data stream is accumulated, or a change is made to primitives within segments when primitives outside segments have been drawn.

RCPPCPAG

0 The call applies to GDDM pages other than the current one.

1 The call applies to the current GDDM page only.

RCPPHCNG

0 The call does not cause any change to the hierarchical structure.

1 The call may cause a change to the hierarchical structure of GDDM. One or more of a page, a partition, a partition set, or a device are affected. The flag is set if any of the current elements in the hierarchy may be changed, or an entry may be added to or deleted from the set of hierarchical entities.

RCPPAPLS

0 The call does not require any special processing by APL.

1 The call may require special processing by APL.

RCPPNAME(n)
The last four characters of the GDDM call statement name.

RCPPRCP(n)
Fullword integer specifying the GDDM request control parameter (RCP) code associated with the call statement.

RCPPDESC(n)
This field contains the descriptors for the arguments that may be passed on to GDDM. The Parameter Descriptor Table shown below describes the contents of this field.

Table 42. Call format descriptor module — parameter descriptor table

Field name	Field offset	Field length
RCPPPDES(1)	0	
.RCPPNARG(1)	0	1
.RCPPDFLG(1)	1	1
..RCPPMATC	1xxx xxxx	bit within RCPPDFLG
(reserved)	x1xx xxxx	bit within RCPPDFLG
(reserved)	xx1x xxxx	bit within RCPPDFLG
(reserved)	xxx1 xxxx	bit within RCPPDFLG
(reserved)	xxxx 1xxx	bit within RCPPDFLG
(reserved)	xxxx x1xx	bit within RCPPDFLG
(reserved)	xxxx xx1x	bit within RCPPDFLG
(reserved)	xxxx xxx1	bit within RCPPDFLG
.RCPPMVAL(1)	2	2
.RCPPDARG(1,1)	4	4
..RCPPAFLG	4	1
RCPPCHAR	1xxx xxxx	bit within RCPPAFLG
RCPPFIX	x1xx xxxx	bit within RCPPAFLG
RCPPFLO	xx1x xxxx	bit within RCPPAFLG
RCPPUNDF	xxx1 xxxx	bit within RCPPAFLG
RCPPLEN	xxxx 1xxx	bit within RCPPAFLG
RCPPNLEN	xxxx x1xx	bit within RCPPAFLG
RCPPINP	xxxx xx1x	bit within RCPPAFLG
RCPPOUT	xxxx xxx1	bit within RCPPAFLG
..RCPPLACC(1,1)	5	1
..RCPPLVAL(1,1)	6	2
.RCPPDARG(1,2)	8	4
)		
) as above,		
) with ...(1,2)		
)		
.RCPPDARG(1,n)	4*RCPPNARG	4
)		
) as above,		
) with ...(1,n)		
)		
RCPPPDES(2)		
)		
) as above,		
) with ...(2)		
)		
RCPPPDES(n)		

The parameter descriptor table

The Parameter Descriptor Table (see Table 42) is imbedded within the Call descriptor table as described above. For each GDDM Call statement there is one or more sets of parameter descriptors. Multiple descriptors are provided when the contents of a parameter list may vary depending upon the contents of the first argument in the parameter list.

RCPPNARG(n)
A one-byte field containing the number of elements in the array RCPPDARG described below. This field contains a value of zero if no parameters are passed to or received from GDDM.

The value of this field may be greater than the number of parameters passed to or received from GDDM. In this case, the argument descriptors contain dummy entries used to copy length information between the accumulators used to determine the length of passed or returned data.

With the exception of the dummy entries, each successive element in the array RCPPDARG(n) describes successive arguments passed to or received from GDDM on the call statement.

RCPPDFLG(n)
A set of flags, one of which is currently used, to indicate features of the parameters passed to or received from GDDM.

RCPPMATC(n)

0 The parameter descriptors provided in the array RCPPDARG(n) are valid regardless of the contents of the first argument passed to GDDM.

1	The parameter descriptors provided in the array RCPPDARG described below are only valid if the contents of the first parameter, which are always a fixed-point number, are the same as the value specified in the field RCPPMVAL(n). If the contents of the passed parameter do not match those in RCPPMVAL, the next set of parameter descriptors, RCPPPDES(n+1), must be used to test for a matching argument value, or to describe the argument list, depending upon the value of flag RCPPMATC(n+1).

RCPPMVAL(n)
If flag RCPPMATC(n) = 1, this two-byte field contains the value that the first parameter passed to GDDM must match if the parameter descriptors in array RCPPDARG(n) are the correct descriptors for the instance of the call statement.

RCPPDARG(n,m)
This is an array of dimension RCPPNARG(n). Each element of the array is four bytes long, and is either a descriptor for an argument passed to or received from GDDM, or is a dummy entry used to prime the length accumulators.

RCPPAFLG(n,m)
A set of flags to indicate the type of the data passed to or received from GDDM.

The parameter data-type flags (in RCPPAFLG, bits 0..5) are set as combinations of these bits, with the meaning shown below

All unlisted combinations are reserved for future use.

000100	The parameter contains undefined format data. The structure of the argument is too complex to describe with a control block structure, probably because the length of the data item cannot be determined without knowledge of the values of the contents of one or more fields imbedded within a prior argument passed to GDDM.
001000	The parameter contains floating point data.
010000	The parameter contains full-word fixed point data.
010100	The parameter contains half-word fixed point data.
100000	The parameter contains character data.
000010	The data passed in this parameter is a full-word fixed-point number that is used as either a length or an array dimension. Field RCPPLACC(n,m) contains the number of an accumulator into which the length should be multiplied.
000001	The parameter being described contains a full-word length or dimension. Field RCPPLACC(n,m) contains an accumulator number into which the length should be multiplied. Parameter descriptor RCPPDARG(n,m+1) also describes the same passed parameter. This parameter descriptor is therefore used to prime two or more length accumulators from the same argument passed to GDDM.

RCPPINP

0	The data passed in the parameter is not input to GDDM.
1	The data passed in this parameter is input to GDDM.

RCPPOUT

0	The data passed in the parameter is not output from GDDM.
1	The data passed in this parameter is output from GDDM.

RCPPLACC(n,m)
This one-byte field contains an accumulator number. Accumulators are used to define the length of character strings or the number of elements in an array of numbers. Nine accumulators are provided, and all accumulators are assumed to start with an initial value of one.

If either of the flags RCPPLEN or RCPPNLEN is set to 1, this field contains the accumulator number that the argument passed to GDDM should be multiplied into.

If flags RCPPLEN and RCPPNLEN are both set to 0, the accumulator contains the number of characters in a character argument, or the number of full-words in a numeric array. If an accumulator number of zero is specified, the length or dimension is assumed to be 1. This length or dimension is subject to modification by the contents of field RCPPLVAL(n,m).

RCPPLVAL(n,m)
This two-byte field contains a modifier to be applied to the length of character strings or the dimension of numeric arrays. The total length of the character string, or dimension of a numeric array is obtained by multiplying the contents of the accumulator specified in field RCPPLACC(n,m) with the value of the field RCPPLVAL(n,m).

Appendix I. APL request codes module

An APL request codes module, which is independent of the subsystem under which GDDM is running, is provided with GDDM. The module defines for each GDDM call statement, the associated APL request code to be used by an APL function when requesting services of GDDM through the APL Auxiliary Processor AP126.

Although all GDDM call statements have an equivalent APL code assigned, not all codes are supported through AP126. The APL manuals listed below identify the supported codes for each of the subsystems for which APL is available.

- *VS APL for CICS/VS Terminal User's Guide,*
- *VS APL for CMS: Terminal User's Guide,*
- *VS APL for OS/TSO: Terminal User's Guide,*
- *APL2 Programming: System Services Reference* manual.

The address of the APL Request Codes Module can be acquired by an application program by using the CALLINF external default option in the SPINIT call; see "Initialization" on page 103, and Chapter 1, "Customizing your program and its environment" on page 1.

The APL Request Codes Module is in two sections. The first provides an address table locating the descriptors for a specific range of APL codes. The second section defines the equivalence between APL request codes and GDDM calls for all codes within a specific range.

The address table

The address table is located at offset 0 from the entry point of the module. The format of the address table is as follows:

Table 43. APL request codes module – address table

Field name	Field offset	Field length
RCPAIDEN	0	8
RCPAVERS	8	4
RCPATNUM	C	4
RCPAENUM	10	4
RCPATABP(1)	14	8
...RCPALOW(1)	...+0	2
...RCPAHIGH(1)	...+2	2
...RCPAPTR(1)	...+4	4
RCPATABP(2)	1C	8
....	....	....
....	....	....
RCPATABP(n)	14+(n-1)★8	8

RCPAIDEN
Module identifier containing the character string "ADMADAP."

RCPAVERS
A full-word integer identifying the version number of the APL Request Codes Module. The field is currently set to zero.

RCPATNUM
A full-word integer containing the number of assigned APL codes defined in the following tables.

RCPAENUM
A full-word integer containing the number of table indexes to follow. One table index exists for each block of APL codes in the range:

```
100*n : 100*(n+1)-1
```

where n is greater than or equal to 0. Thus, the maximum APL code is:

```
RCPAENUM*100-1
```

RCPALOW(n)
A two-byte integer identifying the lowest value in the APL index table pointed to by RCPAPTR(n). The value is currently always set to 100★(n-1).

RCPAHIGH(n)
A two-byte integer identifying the highest value in the APL index table pointed to by RCPAPTR(n). The value is always less than 100★n.

RCPAPTR(n)
The address of the request code table for those APL codes in the range RCPALOW(n) through RCPAHIGH(n). If the value of the pointer is zero, there are no codes assigned within the range.

The request code table

The Request Code Table is addressed from the address table described in "The address table." There is one entry in the table for each potential APL code in the range RCPALOW(n) through RCPAHIGH(n). The format of the request code table is:

Table 44. APL request codes module – request code table

Field name	Field offset	Field length
RCPAAPLC(1)	0	8
RCPAAPLN(1)	0	2
RCPAAPLG(1)	2	6
RCPAAPLC(2)	8	8
RCPAAPLN(2)	8	2
RCPAAPLG(2)	A	6
....	....	....
....	....	....
RCPAAPLC(m)	8★(m-1)	8

RCPAPLN(i)
A two-byte integer containing the APL request code. A code of zero indicates that there is no GDDM function assigned to that code.

RCPAAPLG(i)
A six-byte character string containing the name of the GDDM call (for example "ASREAD") corresponding to the APL function code in RCPAPLN(i).

GDDM Base calls and associated APL codes

GDDM Base APL codes, in alphabetic order

This table lists the APL codes for the GDDM Base calls in alphabetic order of call name.

The table on page 219 lists the APL codes for the GDDM Base calls in numeric order.

Call name	APL code	Description
APDEF	280	Define a field list
APDEL	281	Delete a field list
APMOD	282	Modify a field list
APQIDS	283	Query field list identifiers
APQNUM	284	Query field list numbers
APQRY	285	Query a field list
APQSIZ	286	Query a field list size
APQUID	287	Query unique field list identifier
ASCCOL	421	Specify character colors within a field
ASCGET	422	Get field contents
ASCHLT	423	Specify character highlights within a field
ASCPUT	424	Specify field contents
ASCSS	425	Specify character symbol sets within a field
ASDFLD	401	Define or delete a single field
ASDFLT	406	Set default field attributes
ASDFMT	402	Define alphanumeric fields, deleting all existing fields
ASDTRN	403	Define I/O translation tables
ASFBDY	436	Define field outline
ASFCLR	404	Clear fields
ASFCOL	407	Define field color
ASFCUR	430	Position the cursor
ASFEND	408	Define field end attribute
ASFHLT	409	Define field highlighting
ASFIN	410	Define input null-to-blank conversion
ASFINT	411	Define field intensity
ASFMOD	412	Change field status
ASFOUT	413	Define output blank-to-null conversion
ASFPSS	414	Define primary symbol set for a field
ASFSEN	437	Define field mixed-string attribute
ASFTRA	434	Define field transparency attribute
ASFTRN	415	Assign translation table set to a field
ASFTYP	416	Define field type
ASGGET	433	Get double-character field contents
ASGPUT	432	Specify double-character field contents
ASMODE	426	Define the operator reply mode
ASQCOL	427	Query character colors for a field
ASQCUR	431	Query cursor position
ASQFLD	418	Query field attributes
ASQHLT	428	Query character highlights for a field
ASQLEN	443	Query length of field contents
ASQMAX	419	Query the number of fields
ASQMOD	420	Query modified fields
ASQNMF	435	Query the number of modified fields
ASQSS	429	Query character symbol sets for a field
ASRATT	417	Define field attributes
ASREAD	101	Device output/input
ASRFMT	405	Define multiple fields without deleting existing fields
ASTYPE	111	Override alphanumeric character-code assignments
CDPU	1196	Control the printing of Composite Documents
DSCLS	902	Close a device
DSCMF	439	User Control function
DSDROP	904	Discontinue device usage
DSOPEN	901	Open a device
DSQCMF	440	Query user control function
DSQDEV	907	Query device characteristics
DSQUID	905	Query unique device identifier
DSQUSE	906	Query device usage

Call name	APL code	Description
DSRNIT	908	Reinitialize a device
DSUSE	903	Specify device usage
ESACRT	127	Create application group
ESADEL	128	Delete application group
ESAQRY	129	Query the current application group
ESASEL	130	Select an application group
ESEUDS	124	Specify encoded user default specification
ESLIB	112	Library management
ESPCB	113	Identify program communication block
ESQCPG	133	Query code page of a GDDM object
ESQEUD	135	Query encoded user default specification
ESSCPG	134	Set code page of a GDDM object
ESSUDS	123	Specify source-format user default specification
FSALRM	109	Sound the terminal alarm
FSCHEK	106	Check picture complexity before output
FSCLS	601	Close alternate device
FSCOPY	602	Send page to alternate device
FSENAB	313	Enable/disable device input
FSEXIT	114	Specify an error exit, or error threshold, or both
FSFRCE	102	Update the display
FSINIT	117	Initialize GDDM processing
FSLOG	603	Send character string to alternate device
FSLOGC	606	Send character string with carriage-control character to alternate device
FSOPEN	604	Open alternate device
FSPCLR	301	Clear the current page
FSPCRT	302	Create a page
FSPDEL	303	Delete a page
FSPQRY	304	Query specified page
FSPSEL	305	Select a page
FSPWIN	309	Set page window
FSQCPG	306	Query current page identifier
FSQDEV	110	Query device characteristics
FSQERR	107	Query last error
FSQSYS	122	Query systems environment
FSQUPD	663	Query update mode
FSQUPG	307	Query unique page identifier
FSQURY	121	Query device characteristics
FSQWIN	310	Query page window
FSREST	103	Retransmit data
FSRNIT	118	Reinitialize GDDM
FSSAVE	104	Save current page contents
FSSHOR	119	Extended FSSHOW
FSSHOW	105	Display a saved picture
FSTERM	116	Terminate GDDM processing
FSTRAN	132	Translate character string
FSTRCE	108	Control internal trace
FSUPDM	662	Set update mode
GSAM	647	Set attribute mode
GSARC	521	Draw a circular arc
GSARCC	598	Specify aspect-ratio control (for copy)
GSAREA	522	Start a shaded area
GSBMIX	664	Set current background color-mixing mode
GSBND	657	Define a data boundary
GSCA	510	Set current character angle
GSCALL	653	Call a segment
GSCB	511	Set character-box size
GSCBS	646	Set character-box spacing
GSCD	512	Set current character direction
GSCH	558	Set current character shear
GSCHAP	523	Draw a character string at current position
GSCHAR	524	Draw a character string at a specified point
GSCLP	501	Enable and disable clipping
GSCLR	506	Clear the graphics field
GSCM	513	Set current character mode
GSCOL	514	Set current color
GSCOPY	605	Send graphics to alternate device

Call name	APL code	Description
GSCORR	638	Explicit correlation of tag to primitive
GSCORS	655	Explicit correlation of structure
GSCP	668	Set current position
GSCPG	215	Set current code page
GSCS	515	Set current symbol set
GSDEFE	661	End drawing defaults definition
GSDEFS	660	Start the drawing defaults definition
GSDSS	201	Load a graphics symbol set from the application program
GSELPS	551	Draw an elliptic arc
GSENAB	572	Enable or disable a logical input device
GSENDA	525	End a shaded area
GSFLD	502	Define the graphics field
GSFLSH	573	Clear the graphics input queue
GSFLW	561	Set current fractional line width
GSGET	555	Retrieve graphics data
GSGETE	556	End retrieval of graphics data
GSGETS	554	Start retrieval of graphics data
GSIDVF	571	Initial data value, float
GSIDVI	570	Initial data value, integer
GSILOC	568	Initialize locator
GSIMG	552	Draw a graphics image
GSIMGS	565	Draw a scaled graphics image
GSIPIK	569	Initialize pick device
GSISTK	595	Initialize stroke device
GSISTR	594	Initialize string device
GSLINE	526	Draw a straight line
GSLOAD	593	Load segments
GSLSS	202	Load a graphics symbol set from auxiliary storage
GSLT	516	Set current line type
GSLW	517	Set current line width
GSMARK	527	Draw a marker symbol
GSMB	636	Set marker-box size
GSMIX	518	Set current foreground color-mixing mode
GSMOVE	528	Move without drawing
GSMRKS	529	Draw a series of marker symbols
GSMS	519	Set the current type of marker symbol
GSMSC	563	Set marker scale
GSPAT	520	Set current shading pattern
GSPFLT	557	Draw a curved fillet
GSPLNE	530	Draw a series of lines
GSPOP	649	Restore attributes
GSPS	503	Define the picture space
GSPUT	553	Restore graphics data
GSQAGA	589	Query all geometric attributes
GSQAM	648	Query the current attribute mode
GSQATI	579	Query initial segment attributes
GSQATS	581	Query segment attributes
GSQBMX	665	Query the current background color-mixing mode
GSQBND	656	Query the current data boundary definition
GSQCA	532	Query character angle
GSQCB	533	Query character-box size
GSQCBS	650	Query character-box spacing
GSQCD	534	Query character direction
GSQCEL	535	Query default graphics cell size
GSQCH	559	Query character shear
GSQCHO	575	Query choice device data
GSQCLP	536	Query the clipping state
GSQCM	537	Query the current character mode
GSQCOL	538	Query the current color
GSQCP	539	Query the current position
GSQCPG	216	Query code page
GSQCS	540	Query the current symbol-set identifier
GSQCUR	541	Query the cursor position
GSQFLD	585	Query the graphics field
GSQFLW	562	Query the current fractional line width
GSQLID	643	Query logical input device

Call name	APL code	Description
GSQLOC	576	Query graphics locator data
GSQLT	542	Query the current line type
GSQLW	543	Query the current line width
GSQMAX	544	Query the number of segments
GSQMB	637	Query marker box
GSQMIX	545	Query the current color mixing mode
GSQMS	546	Query the current marker symbol
GSQMSC	564	Query marker scale
GSQNSS	209	Query the number of loaded symbol sets
GSQORG	639	Query segment origin
GSQPAT	547	Query the current shading pattern
GSQPIK	577	Query pick data
GSQPKS	654	Query pick structure
GSQPOS	583	Query segment position
GSQPRI	635	Query segment priority
GSQPS	548	Query the picture-space definition
GSQSEN	667	Query mixed string attribute of graphics text
GSQSIM	574	Query existence of simultaneous queue entry
GSQSS	210	Query loaded symbol sets
GSQSSD	586	Query symbol set data
GSQSTK	597	Query stroke data
GSQSTR	596	Query string data
GSQSVL	659	Query the current segment viewing limits
GSQTA	645	Query the current text alignment
GSQTAG	567	Query current tag
GSQTB	560	Query the text box
GSQTFM	591	Query segment transform
GSQVIE	549	Query the current viewport definition
GSQWIN	550	Query the current window definition
GSREAD	120	Await graphics input
GSRSS	207	Release a graphics symbol set
GSSAGA	588	Set all geometric attributes
GSSATI	578	Set initial segment attributes
GSSATS	580	Modify segment attributes
GSSAVE	592	Save a segment
GSSCLS	507	Close the current segment
GSSCPY	633	Copy a segment
GSSCT	651	Set current transform
GSSDEL	508	Delete a segment
GSSEG	509	Create a segment
GSSEN	666	Set mixed string attribute of graphics text
GSSINC	632	Include a segment
GSSORG	587	Set segment origin
GSSPOS	582	Set segment position
GSSPRI	634	Set segment priority
GSSTFM	590	Set segment transform
GSSVL	658	Define segment viewing limits
GSTA	644	Set text alignment
GSTAG	566	Set current primitive tag
GSUWIN	584	Define a uniform graphics window
GSVECM	531	Vectors
GSVIEW	504	Define a viewport
GSWIN	505	Define a graphics window
IMACLR	1604	Clear a rectangle in an image
IMACRT	1601	Create an image
IMADEL	1603	Delete the image associated with the identifier
IMAGID	1600	Get and reserve a unique image identifier
IMAGT	1613	Retrieve image data from an image
IMAGTE	1614	End retrieval of data from an image
IMAGTS	1612	Start retrieval of data from an image
IMAPT	1610	Enter data into an image
IMAPTE	1611	End data entry into an image
IMAPTS	1609	Start data entry into an image
IMAQRY	1619	Query attributes of an image
IMARES	1602	Convert the resolution attributes of an image
IMARF	1620	Change resolution flag of an image

Call name	APL code	Description
IMARST	1608	Restore image from auxiliary storage
IMASAV	1607	Save image on auxiliary storage
IMATRM	1605	Trim an image down to the specified rectangle
IMPCRT	1650	Create an empty projection
IMPDEL	1652	Delete projection
IMPGID	1651	Get and reserve a unique projection identifier
IMPRST	1654	Restore projection from auxiliary storage
IMPSAV	1653	Save projection on auxiliary storage
IMRBRI	1665	Define brightness conversion algorithm
IMRCON	1666	Define contrast conversion algorithm
IMRCVB	1664	Define bi-level conversion algorithm
IMREX	1655	Define rectangular sub-image in pixel coordinates
IMREXR	1656	Define rectangular sub-image in real coordinates
IMRNEG	1663	Negate the pixels of an extracted image
IMRORN	1661	Turn an extracted image clockwise through a number of right angles
IMRPL	1657	Define place position in pixel coordinates
IMRPLR	1658	Define place position in real coordinates
IMRRAL	1660	Set current resolution/scaling algorithm
IMRREF	1662	Reflect extracted image
IMRSCL	1659	Scale extracted image
IMXFER	1615	Transfer data between two images, applying a projection
ISCTL	182	Set image quality-control parameters
ISENAB	189	Enable or disable image cursor
ISESCA	185	Control echoing of scanner image
ISFLD	180	Define image field
ISIBOX	193	Initialize image box cursor
ISILOC	191	Initialize image locator cursor
ISLDE	186	Load external read-only image
ISQBOX	192	Query image box cursor
ISQCOM	194	Query image compressions supported by the device
ISQFLD	181	Query image field
ISQFOR	184	Query image formats supported by the device
ISQLOC	190	Query image locator cursor position
ISQRES	188	Query supported image resolutions
ISQSCA	187	Query image scanner device
ISSE	1201	Run the Image Symbol Editor
ISXCTL	183	Extended set image quality control parameters
MSCPOS	1112	Set cursor position
MSDFLD	1108	Create or delete a mapped field
MSGET	1110	Retrieve data from a map
MSPCRT	1102	Create a page for mapping
MSPQRY	308	Query current page
MSPUT	1109	Place data into a mapped field
MSQADS	1105	Query application data structure definition
MSQFIT	1106	Query map fit
MSQFLD	1111	Query mapped field characteristics
MSQGRP	1103	Query mapgroup characteristics
MSQMAP	1104	Query map characteristics
MSQMOD	1107	Query modified fields
MSQPOS	1113	Query cursor position
MSREAD	1101	Present mapped data
PSDSS	203	Load a symbol set into a PS store from the application program
PSLSS	204	Load a symbol set into a PS store from auxiliary storage
PSLSSC	205	Conditionally load a symbol set into a PS store from auxiliary storage
PSQSS	211	Query status of device stores
PSRSS	208	Release a symbol set from a PS store
PSRSV	206	Reserving or releasing a PS store
PTNCRT	1021	Create a partition
PTNDEL	1025	Delete a partition
PTNMOD	1023	Modify the current partition
PTNQRY	1022	Query the current partition
PTNQUN	1026	Query unique partition identifier
PTNSEL	1024	Select a partition
PTSCRT	1001	Create a partition set
PTSDEL	1004	Delete a partition set
PTSQPI	1008	Query partition identifiers

Call name	APL code	Description
PTSQPN	1009	Query partition numbers
PTSQPP	1007	Query partition viewing priorities
PTSQRY	1002	Query partition set attributes
PTSQUN	1005	Query unique partition set identifier
PTSSEL	1003	Select a partition set
PTSSPP	1006	Set partition viewing priorities
SPINIT	115	Initialize GDDM with SPIB
SPMXMP	438	Control the use of mixed fields by mapping
SSQF	212	Query a symbol set on auxiliary storage
SSREAD	213	Read a symbol set from auxiliary storage
SSWRT	214	Write a symbol set to auxiliary storage
WSCRT	1040	Create an operator window
WSDEL	1041	Delete operator window
WSIO	151	Windowed device input/output
WSMOD	1043	Modify the current operator window
WSQRY	1044	Query the current operator window
WSQUN	1045	Query unique operator window identifier
WSQWI	1046	Query operator window identifiers
WSQWN	1050	Query operator window numbers
WSQWP	1049	Query operator window viewing priorities
WSSEL	1051	Select an operator window
WSSWP	1052	Set operator window viewing priorities

GDDM Base APL codes, in numeric order

This table lists the APL codes for the GDDM Base calls in numeric order.

The table on page 214 lists the APL codes for the GDDM Base calls in alphabetic order of call name.

APL code	Call name	Description
101	ASREAD	Device output/input
102	FSFRCE	Update the display
103	FSREST	Retransmit data
104	FSSAVE	Save current page contents
105	FSSHOW	Display a saved picture
106	FSCHEK	Check picture complexity before output
107	FSQERR	Query last error
108	FSTRCE	Control internal trace
109	FSALRM	Sound the terminal alarm
110	FSQDEV	Query device characteristics
111	ASTYPE	Override alphanumeric character-code assignments
112	ESLIB	Library management
113	ESPCB	Identify program communication block
114	FSEXIT	Specify an error exit, or error threshold, or both
115	SPINIT	Initialize GDDM with SPIB
116	FSTERM	Terminate GDDM processing
117	FSINIT	Initialize GDDM processing
118	FSRNIT	Reinitialize GDDM
119	FSSHOR	Extended FSSHOW
120	GSREAD	Await graphics input
121	FSQURY	Query device characteristics
122	FSQSYS	Query systems environment
123	ESSUDS	Specify source-format user default specification
124	ESEUDS	Specify encoded user default specification
127	ESACRT	Create application group
128	ESADEL	Delete application group
129	ESAQRY	Query the current application group
130	ESASEL	Select an application group
132	FSTRAN	Translate character string
133	ESQCPG	Query code page of a GDDM object
134	ESSCPG	Set code page of a GDDM object
135	ESQEUD	Query encoded user default specification
151	WSIO	Windowed device input/output

APL code	Call name	Description
180	ISFLD	Define image field
181	ISQFLD	Query image field
182	ISCTL	Set image quality-control parameters
183	ISXCTL	Extended set image quality control parameters
184	ISQFOR	Query image formats supported by the device
185	ISESCA	Control echoing of scanner image
186	ISLDE	Load external read-only image
187	ISQSCA	Query image scanner device
188	ISQRES	Query supported image resolutions
189	ISENAB	Enable or disable image cursor
190	ISQLOC	Query image locator cursor position
191	ISILOC	Initialize image locator cursor
192	ISQBOX	Query image box cursor
193	ISIBOX	Initialize image box cursor
194	ISQCOM	Query image compressions supported by the device
201	GSDSS	Load a graphics symbol set from the application program
202	GSLSS	Load a graphics symbol set from auxiliary storage
203	PSDSS	Load a symbol set into a PS store from the application program
204	PSLSS	Load a symbol set into a PS store from auxiliary storage
205	PSLSSC	Conditionally load a symbol set into a PS store from auxiliary storage
206	PSRSV	Reserving or releasing a PS store
207	GSRSS	Release a graphics symbol set
208	PSRSS	Release a symbol set from a PS store
209	GSQNSS	Query the number of loaded symbol sets
210	GSQSS	Query loaded symbol sets
211	PSQSS	Query status of device stores
212	SSQF	Query a symbol set on auxiliary storage
213	SSREAD	Read a symbol set from auxiliary storage
214	SSWRT	Write a symbol set to auxiliary storage
215	GSCPG	Set current code page
216	GSQCPG	Query code page
280	APDEF	Define a field list
281	APDEL	Delete a field list
282	APMOD	Modify a field list
283	APQIDS	Query field list identifiers
284	APQNUM	Query field list numbers
285	APQRY	Query a field list
286	APQSIZ	Query a field list size
287	APQUID	Query unique field list identifier
301	FSPCLR	Clear the current page
302	FSPCRT	Create a page
303	FSPDEL	Delete a page
304	FSPQRY	Query specified page
305	FSPSEL	Select a page
306	FSQCPG	Query current page identifier
307	FSQUPG	Query unique page identifier
308	MSPQRY	Query current page
309	FSPWIN	Set page window
310	FSQWIN	Query page window
313	FSENAB	Enable/disable device input
401	ASDFLD	Define or delete a single field
402	ASDFMT	Define alphanumeric fields, deleting all existing fields
403	ASDTRN	Define I/O translation tables
404	ASFCLR	Clear fields
405	ASRFMT	Define multiple fields without deleting existing fields
406	ASDFLT	Set default field attributes
407	ASFCOL	Define field color
408	ASFEND	Define field end attribute
409	ASFHLT	Define field highlighting
410	ASFIN	Define input null-to-blank conversion
411	ASFINT	Define field intensity
412	ASFMOD	Change field status
413	ASFOUT	Define output blank-to-null conversion
414	ASFPSS	Define primary symbol set for a field
415	ASFTRN	Assign translation table set to a field
416	ASFTYP	Define field type

APL code	Call name	Description
417	ASRATT	Define field attributes
418	ASQFLD	Query field attributes
419	ASQMAX	Query the number of fields
420	ASQMOD	Query modified fields
421	ASCCOL	Specify character colors within a field
422	ASCGET	Get field contents
423	ASCHLT	Specify character highlights within a field
424	ASCPUT	Specify field contents
425	ASCSS	Specify character symbol sets within a field
426	ASMODE	Define the operator reply mode
427	ASQCOL	Query character colors for a field
428	ASQHLT	Query character highlights for a field
429	ASQSS	Query character symbol sets for a field
430	ASFCUR	Position the cursor
431	ASQCUR	Query cursor position
432	ASGPUT	Specify double-character field contents
433	ASGGET	Get double-character field contents
434	ASFTRA	Define field transparency attribute
435	ASQNMF	Query the number of modified fields
436	ASFBDY	Define field outline
437	ASFSEN	Define field mixed-string attribute
438	SPMXMP	Control the use of mixed fields by mapping
439	DSCMF	User Control function
440	DSQCMF	Query user control function
443	ASQLEN	Query length of field contents
501	GSCLP	Enable and disable clipping
502	GSFLD	Define the graphics field
503	GSPS	Define the picture space
504	GSVIEW	Define a viewport
505	GSWIN	Define a graphics window
506	GSCLR	Clear the graphics field
507	GSSCLS	Close the current segment
508	GSSDEL	Delete a segment
509	GSSEG	Create a segment
510	GSCA	Set current character angle
511	GSCB	Set character-box size
512	GSCD	Set current character direction
513	GSCM	Set current character mode
514	GSCOL	Set current color
515	GSCS	Set current symbol set
516	GSLT	Set current line type
517	GSLW	Set current line width
518	GSMIX	Set current foreground color-mixing mode
519	GSMS	Set the current type of marker symbol
520	GSPAT	Set current shading pattern
521	GSARC	Draw a circular arc
522	GSAREA	Start a shaded area
523	GSCHAP	Draw a character string at current position
524	GSCHAR	Draw a character string at a specified point
525	GSENDA	End a shaded area
526	GSLINE	Draw a straight line
527	GSMARK	Draw a marker symbol
528	GSMOVE	Move without drawing
529	GSMRKS	Draw a series of marker symbols
530	GSPLNE	Draw a series of lines
531	GSVECM	Vectors
532	GSQCA	Query character angle
533	GSQCB	Query character-box size
534	GSQCD	Query character direction
535	GSQCEL	Query default graphics cell size
536	GSQCLP	Query the clipping state
537	GSQCM	Query the current character mode
538	GSQCOL	Query the current color
539	GSQCP	Query the current position
540	GSQCS	Query the current symbol-set identifier
541	GSQCUR	Query the cursor position

APL code	Call name	Description
542	GSQLT	Query the current line type
543	GSQLW	Query the current line width
544	GSQMAX	Query the number of segments
545	GSQMIX	Query the current color mixing mode
546	GSQMS	Query the current marker symbol
547	GSQPAT	Query the current shading pattern
548	GSQPS	Query the picture-space definition
549	GSQVIE	Query the current viewport definition
550	GSQWIN	Query the current window definition
551	GSELPS	Draw an elliptic arc
552	GSIMG	Draw a graphics image
553	GSPUT	Restore graphics data
554	GSGETS	Start retrieval of graphics data
555	GSGET	Retrieve graphics data
556	GSGETE	End retrieval of graphics data
557	GSPFLT	Draw a curved fillet
558	GSCH	Set current character shear
559	GSQCH	Query character shear
560	GSQTB	Query the text box
561	GSFLW	Set current fractional line width
562	GSQFLW	Query the current fractional line width
563	GSMSC	Set marker scale
564	GSQMSC	Query marker scale
565	GSIMGS	Draw a scaled graphics image
566	GSTAG	Set current primitive tag
567	GSQTAG	Query current tag
568	GSILOC	Initialize locator
569	GSIPIK	Initialize pick device
570	GSIDVI	Initial data value, integer
571	GSIDVF	Initial data value, float
572	GSENAB	Enable or disable a logical input device
573	GSFLSH	Clear the graphics input queue
574	GSQSIM	Query existence of simultaneous queue entry
575	GSQCHO	Query choice device data
576	GSQLOC	Query graphics locator data
577	GSQPIK	Query pick data
578	GSSATI	Set initial segment attributes
579	GSQATI	Query initial segment attributes
580	GSSATS	Modify segment attributes
581	GSQATS	Query segment attributes
582	GSSPOS	Set segment position
583	GSQPOS	Query segment position
584	GSUWIN	Define a uniform graphics window
585	GSQFLD	Query the graphics field
586	GSQSSD	Query symbol set data
587	GSSORG	Set segment origin
588	GSSAGA	Set all geometric attributes
589	GSQAGA	Query all geometric attributes
590	GSSTFM	Set segment transform
591	GSQTFM	Query segment transform
592	GSSAVE	Save a segment
593	GSLOAD	Load segments
594	GSISTR	Initialize string device
595	GSISTK	Initialize stroke device
596	GSQSTR	Query string data
597	GSQSTK	Query stroke data
598	GSARCC	Specify aspect-ratio control (for copy)
601	FSCLS	Close alternate device
602	FSCOPY	Send page to alternate device
603	FSLOG	Send character string to alternate device
604	FSOPEN	Open alternate device
605	GSCOPY	Send graphics to alternate device
606	FSLOGC	Send character string with carriage-control character to alternate device
632	GSSINC	Include a segment
633	GSSCPY	Copy a segment
634	GSSPRI	Set segment priority

APL code	Call name	Description
635	GSQPRI	Query segment priority
636	GSMB	Set marker-box size
637	GSQMB	Query marker box
638	GSCORR	Explicit correlation of tag to primitive
639	GSQORG	Query segment origin
643	GSQLID	Query logical input device
644	GSTA	Set text alignment
645	GSQTA	Query the current text alignment
646	GSCBS	Set character-box spacing
647	GSAM	Set attribute mode
648	GSQAM	Query the current attribute mode
649	GSPOP	Restore attributes
650	GSQCBS	Query character-box spacing
651	GSSCT	Set current transform
653	GSCALL	Call a segment
654	GSQPKS	Query pick structure
655	GSCORS	Explicit correlation of structure
656	GSQBND	Query the current data boundary definition
657	GSBND	Define a data boundary
658	GSSVL	Define segment viewing limits
659	GSQSVL	Query the current segment viewing limits
660	GSDEFS	Start the drawing defaults definition
661	GSDEFE	End drawing defaults definition
662	FSUPDM	Set update mode
663	FSQUPD	Query update mode
664	GSBMIX	Set current background color-mixing mode
665	GSQBMX	Query the current background color-mixing mode
666	GSSEN	Set mixed string attribute of graphics text
667	GSQSEN	Query mixed string attribute of graphics text
668	GSCP	Set current position
901	DSOPEN	Open a device
902	DSCLS	Close a device
903	DSUSE	Specify device usage
904	DSDROP	Discontinue device usage
905	DSQUID	Query unique device identifier
906	DSQUSE	Query device usage
907	DSQDEV	Query device characteristics
908	DSRNIT	Reinitialize a device
1001	PTSCRT	Create a partition set
1002	PTSQRY	Query partition set attributes
1003	PTSSEL	Select a partition set
1004	PTSDEL	Delete a partition set
1005	PTSQUN	Query unique partition set identifier
1006	PTSSPP	Set partition viewing priorities
1007	PTSQPP	Query partition viewing priorities
1008	PTSQPI	Query partition identifiers
1009	PTSQPN	Query partition numbers
1021	PTNCRT	Create a partition
1022	PTNQRY	Query the current partition
1023	PTNMOD	Modify the current partition
1024	PTNSEL	Select a partition
1025	PTNDEL	Delete a partition
1026	PTNQUN	Query unique partition identifier
1040	WSCRT	Create an operator window
1041	WSDEL	Delete operator window
1043	WSMOD	Modify the current operator window
1044	WSQRY	Query the current operator window
1045	WSQUN	Query unique operator window identifier
1046	WSQWI	Query operator window identifiers
1049	WSQWP	Query operator window viewing priorities
1050	WSQWN	Query operator window numbers
1051	WSSEL	Select an operator window
1052	WSSWP	Set operator window viewing priorities
1101	MSREAD	Present mapped data
1102	MSPCRT	Create a page for mapping
1103	MSQGRP	Query mapgroup characteristics

APL code	Call name	Description
1104	MSQMAP	Query map characteristics
1105	MSQADS	Query application data structure definition
1106	MSQFIT	Query map fit
1107	MSQMOD	Query modified fields
1108	MSDFLD	Create or delete a mapped field
1109	MSPUT	Place data into a mapped field
1110	MSGET	Retrieve data from a map
1111	MSQFLD	Query mapped field characteristics
1112	MSCPOS	Set cursor position
1113	MSQPOS	Query cursor position
1196	CDPU	Control the printing of Composite Documents
1201	ISSE	Run the Image Symbol Editor
1600	IMAGID	Get and reserve a unique image identifier
1601	IMACRT	Create an image
1602	IMARES	Convert the resolution attributes of an image
1603	IMADEL	Delete the image associated with the identifier
1604	IMACLR	Clear a rectangle in an image
1605	IMATRM	Trim an image down to the specified rectangle
1607	IMASAV	Save image on auxiliary storage
1608	IMARST	Restore image from auxiliary storage
1609	IMAPTS	Start data entry into an image
1610	IMAPT	Enter data into an image
1611	IMAPTE	End data entry into an image
1612	IMAGTS	Start retrieval of data from an image
1613	IMAGT	Retrieve image data from an image
1614	IMAGTE	End retrieval of data from an image
1615	IMXFER	Transfer data between two images, applying a projection
1619	IMAQRY	Query attributes of an image
1620	IMARF	Change resolution flag of an image
1650	IMPCRT	Create an empty projection
1651	IMPGID	Get and reserve a unique projection identifier
1652	IMPDEL	Delete projection
1653	IMPSAV	Save projection on auxiliary storage
1654	IMPRST	Restore projection from auxiliary storage
1655	IMREX	Define rectangular sub-image in pixel coordinates
1656	IMREXR	Define rectangular sub-image in real coordinates
1657	IMRPL	Define place position in pixel coordinates
1658	IMRPLR	Define place position in real coordinates
1659	IMRSCL	Scale extracted image
1660	IMRRAL	Set current resolution/scaling algorithm
1661	IMRORN	Turn an extracted image clockwise through a number of right angles
1662	IMRREF	Reflect extracted image
1663	IMRNEG	Negate the pixels of an extracted image
1664	IMRCVB	Define bi-level conversion algorithm
1665	IMRBRI	Define brightness conversion algorithm
1666	IMRCON	Define contrast conversion algorithm

GDDM-PGF calls and associated APL codes

GDDM-PGF APL codes, in alphabetic order

This table lists the APL codes for the GDDM-PGF calls in alphabetic order of call name.

The table on page 227 lists the APL codes for the GDDM-PGF calls in numeric order.

Call name	APL code	Description
CHAATT	735	Axis line attributes
CHAREA	721	Chart area
CHART	1200	Invoke Interactive Chart Utility
CHBAR	795	Plot a bar chart
CHBARX	782	Plot a bar chart with numeric x-axis values
CHBATT	722	Set framing box attributes
CHCGRD	723	Basic character spacing/size
CHCOL	761	Component basic color table
CHCONV	719	Convert coordinate values
CHDATT	757	Datum line attributes
CHDCTL	805	Control the format of values, and the overall size of table charts
CHDRAX	790	Specific control of axis drawing
CHDTAB	806	Construct a table chart
CHFINE	799	Curve fitting smoothness
CHGAP	765	Spacing between bars
CHGATT	754	Grid line attributes
CHGGAP	766	Spacing between bar groups
CHHATT	729	Heading text attributes
CHHEAD	730	Heading text
CHHIST	793	Histograms
CHHMAR	724	Horizontal margins
CHKATT	727	Legend text attributes
CHKEY	726	Legend key labels
CHKEYP	728	Legend base position
CHKMAX	770	Maximum legend width/height
CHKOFF	771	Legend offsets
CHLATT	747	Axis label text attributes
CHLC	800	Component line color table
CHLT	762	Component line type table
CHLW	772	Component line width table
CHMARK	763	Component marker table
CHMISS	804	Missing values on a table chart
CHMKSC	781	Set marker scale values
CHNATT	760	Specify attributes for notes
CHNOFF	759	Specify offsets for CHNOTE
CHNOTE	758	Construct a character string at a designated position
CHNUM	794	Set number of components
CHPAT	764	Component shading pattern table
CHPCTL	710	Control pie chart slices
CHPEXP	775	Exploded slices in pie charts
CHPIE	796	Pie charts
CHPIER	768	Reduce pie chart size
CHPLOT	791	Line graphs and scatter plots
CHPOLR	783	Plot a polar chart
CHQARE	802	Query chart area
CHQPOS	718	Query positional information
CHQRNG	801	Query axis ranges
CHRNIT	704	Reinitialize PG routines
CHSET	798	Specify chart options
CHSSEG	717	Set segment number
CHSTRT	703	Reset the processing state to state-1
CHSURF	792	Surface charts
CHTATT	736	Axis title text attributes
CHTERM	705	Terminate the PG routines
CHTHRS	776	Bar value threshold limit
CHTOWR	785	Plot a tower chart
CHTPRJ	720	Tower chart projection

Call name	APL code	Description
CHVATT	769	Attributes of values text in bar and pie charts
CHVCHR	767	Number of characters in bar values
CHVDIG	774	Set decimal digits for bars and tables
CHVENN	797	Venn diagram
CHVMAR	725	Vertical margins
CHXDAY	752	X-axis day labels
CHXDLB	773	X-axis data labels
CHXDTM	755	X-axis datum line
CHXINT	741	X-axis interception point
CHXLAB	748	X-axis label text
CHXLAT	711	X-axis label attributes
CHXMTH	750	X-axis month labels
CHXRNG	739	X-axis explicit range
CHXSCL	745	X-axis scale factor
CHXSEL	731	X-axis selection
CHXSET	733	X-axis options
CHXTAT	713	X-axis title attributes
CHXTIC	743	X-axis scale mark interval
CHXTTL	737	X-axis title specification
CHYDAY	753	Y-axis day labels
CHYDTM	756	Y-axis datum line
CHYINT	742	Y-axis interception point
CHYLAB	749	Y-axis label text
CHYLAT	712	Y-axis label attributes
CHYMTH	751	Y-axis month labels
CHYRNG	740	Y-axis explicit range
CHYSCL	746	Y-axis scale factor
CHYSEL	732	Y-axis selection
CHYSET	734	Y-axis options
CHYTAT	714	Y-axis title attributes
CHYTIC	744	Y-axis scale mark interval
CHYTTL	738	Y-axis title specification
CHZDLB	779	Z-axis data labels
CHZGAP	709	Spacing between towers
CHZLAT	715	Z-axis label attributes
CHZRNG	707	Z-axis explicit range
CHZSET	777	Z-axis options
CHZTIC	706	Z-axis scale mark interval
CSCCRT	1203	Create a chart
CSCDEL	1210	Delete a chart
CSCHA	1219	Set character values for a chart
CSDEL	1227	Delete item for a chart
CSDIR	1224	Build object directory list
CSFLT	1207	Set floating-point values for a chart
CSINT	1205	Set integer values for a chart
CSLOAD	1222	Restore saved chart information
CSNUM	1211	Set control value for a chart
CSQCHA	1220	Query character values for a chart
CSQCHL	1221	Query character lengths for a chart
CSQCS	1226	Query CSxxxx call information
CSQDIR	1225	Query object directory list
CSQFLT	1208	Query floating-point values for a chart
CSQINT	1206	Query integer values for a chart
CSQNUM	1212	Query control value for chart
CSQUID	1209	Query unique chart identifier
CSQXDT	1216	Query independent (x) data values for a chart
CSQXSL	1229	Query independent (x) data selection for a chart
CSQYDT	1217	Query dependent (y) data values for a chart
CSQZDT	1218	Query data group (z) values for a chart
CSQZSL	1231	Query data group (z) selection for a chart
CSSAVE	1223	Save chart information
CSSICU	1204	Start an ICU session for a chart
CSXDT	1213	Set independent (x) data values for a chart
CSXSL	1228	Set independent (x) data selection for a chart
CSYDT	1214	Set dependent (y) data values for a chart
CSZDT	1215	Set data group (z) data values for a chart

Call name	APL code	Description
CSZSL	1230	Set data group (z) selection for a chart
VSSE	1202	Run the Vector Symbol Editor

GDDM-PGF APL codes, in numeric order

This table lists the APL codes for the GDDM-PGF calls in numeric order.

The table on page 225 lists the APL codes for the GDDM-PGF calls in alphabetic order of call name.

APL code	Call name	Description
703	CHSTRT	Reset the processing state to state-1
704	CHRNIT	Reinitialize PG routines
705	CHTERM	Terminate the PG routines
706	CHZTIC	Z-axis scale mark interval
707	CHZRNG	Z-axis explicit range
709	CHZGAP	Spacing between towers
710	CHPCTL	Control pie chart slices
711	CHXLAT	X-axis label attributes
712	CHYLAT	Y-axis label attributes
713	CHXTAT	X-axis title attributes
714	CHYTAT	Y-axis title attributes
715	CHZLAT	Z-axis label attributes
717	CHSSEG	Set segment number
718	CHQPOS	Query positional information
719	CHCONV	Convert coordinate values
720	CHTPRJ	Tower chart projection
721	CHAREA	Chart area
722	CHBATT	Set framing box attributes
723	CHCGRD	Basic character spacing/size
724	CHHMAR	Horizontal margins
725	CHVMAR	Vertical margins
726	CHKEY	Legend key labels
727	CHKATT	Legend text attributes
728	CHKEYP	Legend base position
729	CHHATT	Heading text attributes
730	CHHEAD	Heading text
731	CHXSEL	X-axis selection
732	CHYSEL	Y-axis selection
733	CHXSET	X-axis options
734	CHYSET	Y-axis options
735	CHAATT	Axis line attributes
736	CHTATT	Axis title text attributes
737	CHXTTL	X-axis title specification
738	CHYTTL	Y-axis title specification
739	CHXRNG	X-axis explicit range
740	CHYRNG	Y-axis explicit range
741	CHXINT	X-axis interception point
742	CHYINT	Y-axis interception point
743	CHXTIC	X-axis scale mark interval
744	CHYTIC	Y-axis scale mark interval
745	CHXSCL	X-axis scale factor
746	CHYSCL	Y-axis scale factor
747	CHLATT	Axis label text attributes
748	CHXLAB	X-axis label text
749	CHYLAB	Y-axis label text
750	CHXMTH	X-axis month labels
751	CHYMTH	Y-axis month labels
752	CHXDAY	X-axis day labels
753	CHYDAY	Y-axis day labels
754	CHGATT	Grid line attributes
755	CHXDTM	X-axis datum line
756	CHYDTM	Y-axis datum line
757	CHDATT	Datum line attributes

APL code	Call name	Description
758	CHNOTE	Construct a character string at a designated position
759	CHNOFF	Specify offsets for CHNOTE
760	CHNATT	Specify attributes for notes
761	CHCOL	Component basic color table
762	CHLT	Component line type table
763	CHMARK	Component marker table
764	CHPAT	Component shading pattern table
765	CHGAP	Spacing between bars
766	CHGGAP	Spacing between bar groups
767	CHVCHR	Number of characters in bar values
768	CHPIER	Reduce pie chart size
769	CHVATT	Attributes of values text in bar and pie charts
770	CHKMAX	Maximum legend width/height
771	CHKOFF	Legend offsets
772	CHLW	Component line width table
773	CHXDLB	X-axis data labels
774	CHVDIG	Set decimal digits for bars and tables
775	CHPEXP	Exploded slices in pie charts
776	CHTHRS	Bar value threshold limit
777	CHZSET	Z-axis options
779	CHZDLB	Z-axis data labels
781	CHMKSC	Set marker scale values
782	CHBARX	Plot a bar chart with numeric x-axis values
783	CHPOLR	Plot a polar chart
785	CHTOWR	Plot a tower chart
790	CHDRAX	Specific control of axis drawing
791	CHPLOT	Line graphs and scatter plots
792	CHSURF	Surface charts
793	CHHIST	Histograms
794	CHNUM	Set number of components
795	CHBAR	Plot a bar chart
796	CHPIE	Pie charts
797	CHVENN	Venn diagram
798	CHSET	Specify chart options
799	CHFINE	Curve fitting smoothness
800	CHLC	Component line color table
801	CHQRNG	Query axis ranges
802	CHQARE	Query chart area
804	CHMISS	Missing values on a table chart
805	CHDCTL	Control the format of values, and the overall size of table charts
806	CHDTAB	Construct a table chart
1200	CHART	Invoke Interactive Chart Utility
1202	VSSE	Run the Vector Symbol Editor
1203	CSCCRT	Create a chart
1204	CSSICU	Start an ICU session for a chart
1205	CSINT	Set integer values for a chart
1206	CSQINT	Query integer values for a chart
1207	CSFLT	Set floating-point values for a chart
1208	CSQFLT	Query floating-point values for a chart
1209	CSQUID	Query unique chart identifier
1210	CSCDEL	Delete a chart
1211	CSNUM	Set control value for a chart
1212	CSQNUM	Query control value for chart
1213	CSXDT	Set independent (x) data values for a chart
1214	CSYDT	Set dependent (y) data values for a chart
1215	CSZDT	Set data group (z) data values for a chart
1216	CSQXDT	Query independent (x) data values for a chart
1217	CSQYDT	Query dependent (y) data values for a chart
1218	CSQZDT	Query data group (z) values for a chart
1219	CSCHA	Set character values for a chart
1220	CSQCHA	Query character values for a chart
1221	CSQCHL	Query character lengths for a chart
1222	CSLOAD	Restore saved chart information
1223	CSSAVE	Save chart information
1224	CSDIR	Build object directory list
1225	CSQDIR	Query object directory list

APL code	Call name	Description
1226	CSQCS	Query CSxxxx call information
1227	CSDEL	Delete item for a chart
1228	CSXSL	Set independent (x) data selection for a chart
1229	CSQXSL	Query independent (x) data selection for a chart
1230	CSZSL	Set data group (z) selection for a chart
1231	CSQZSL	Query data group (z) selection for a chart

Appendix J. Request control parameter codes

This appendix lists the request control parameter (RCP) codes corresponding to the function calls that can be invoked by an application program using GDDM or GDDM-PGF. The RCP codes are the function codes to be specified when invoking GDDM and GDDM-PGF by means of the system programmer interface (SPI) described in the *GDDM Base Programming Reference, Volume 1.* The codes are also set by GDDM in error records passed to user error exits, or produced in response to FSQERR calls; again, see the *GDDM Base Programming Reference, Volume 1.*

Assembler-language tables ADMURCPB and ADMURCPO, provided on the GDDM and GDDM-PGF installation tapes, define RCP codes. Table ADMURCPB, on the GDDM installation tape, defines the RCP codes for GDDM only. ADMURCPO, on the GDDM-PGF installation tape, defines the RCP codes for both GDDM and GDDM-PGF.

The RCP codes are defined as symbolic Assembler-language EQUATE statements of mnemonics (QQxxxxxx) to numeric values. The "xxxxxx" of the mnemonic is the name of the GDDM or GDDM-PGF function; for example,

```
QQFSINIT EQU X'0C000001'
```

Note: These mnemonics cannot be used under DOS assemblers.

To preserve the internal structure of GDDM, and to make it easy to package subsets of GDDM functions, the processing code of these API calls have been reorganized:

FSALRM
FSREST
ASTYPE.

This reorganization changes the RCP codes for these calls, but for compatibility the existing RCP codes are still supported.

GDDM RCP codes

GDDM Base RCP codes, listed alphabetically

The RCP codes for GDDM are listed below in alphabetic order of call name.

The table on page 237 lists the GDDM Base RCP codes in numeric order of RCP code.

Notes:

1. Two functions (SPINIT and SPMXMP) are of type "S," which means that they can be invoked **only** using the SPI; all other functions can be accessed by either the SPI or the API.
2. Each call has the prefix "QQ"; this has been omitted here for clarity.

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
APDEF	0C380000	204996608	Define a field list
APDEL	0C380100	204996864	Delete a field list
APMOD	0C380200	204997120	Modify a field list
APQIDS	0C380300	204997376	Query field list identifiers
APQNUM	0C380400	204997632	Query field list numbers
APQRY	0C380500	204997888	Query a field list
APQSIZ	0C380600	204998144	Query a field list size
APQUID	0C380700	204998400	Query unique field list identifier
ASCCOL	0C080601	201852417	Specify character colors within a field
ASCGET	0C080903	201853187	Get field contents
ASCHLT	0C080600	201852416	Specify character highlights within a field
ASCPUT	0C080603	201852419	Specify field contents
ASCSS	0C080602	201852418	Specify character symbol sets within a field
ASDFLD	0C080700	201852672	Define or delete a single field
ASDFLT	0C080200	201851392	Set default field attributes
ASDFMT	0C080801	201852929	Define alphanumeric fields, deleting all existing fields
ASDTRN	0C080300	201851648	Define I/O translation tables
ASFBDY	0C08050B	201852171	Define field outline
ASFCLR	0C080400	201851904	Clear fields
ASFCOL	0C080502	201852162	Define field color
ASFCUR	0C080100	201851136	Position the cursor
ASFEND	0C080505	201852165	Define field end attribute
ASFHLT	0C080504	201852164	Define field highlighting
ASFIN	0C080507	201852167	Define input null-to-blank conversion
ASFINT	0C080501	201852161	Define field intensity
ASFMOD	0C081100	201855232	Change field status
ASFOUT	0C080506	201852166	Define output blank-to-null conversion
ASFPSS	0C080503	201852163	Define primary symbol set for a field
ASFSEN	0C08050A	201852170	Define field mixed-string attribute

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
ASFTRA	0C080509	201852169	Define field transparency attribute
ASFTRN	0C080508	201852168	Assign translation table set to a field
ASFTYP	0C080500	201852160	Define field type
ASGGET	0C081603	201856515	Get double-character field contents
ASGPUT	0C081503	201856259	Specify double-character field contents
ASMODE	0C080D00	201854208	Define the operator reply mode
ASQCOL	0C080901	201853185	Query character colors for a field
ASQCUR	0C080F00	201854720	Query cursor position
ASQFLD	0C080A00	201853440	Query field attributes
ASQHLT	0C080900	201853184	Query character highlights for a field
ASQLEN	0C081800	201857024	Query length of field contents
ASQMAX	0C080E00	201854464	Query the number of fields
ASQMOD	0C080B00	201853696	Query modified fields
ASQNMF	0C080E01	201854465	Query the number of modified fields
ASQSS	0C080902	201853186	Query character symbol sets for a field
ASRATT	0C080802	201852930	Define field attributes
ASREAD	0C100000	202375168	Device output/input
ASRFMT	0C080800	201852928	Define multiple fields without deleting existing fields
ASTYPE	0C081300	201855744	Override alphanumeric character-code assignments
CDPU	40000000	1073741824	Control the printing of Composite Documents
DSCLS	0C000201	201327105	Close a device
DSCMF	0C080C01	201853953	User Control function
DSDROP	0C000203	201327107	Discontinue device usage
DSOPEN	0C000200	201327104	Open a device
DSQCMF	0C080C02	201853954	Query user control function
DSQDEV	0C000206	201327110	Query device characteristics
DSQUID	0C000204	201327108	Query unique device identifier
DSQUSE	0C000205	201327109	Query device usage
DSRNIT	0C000207	201327111	Reinitialize a device
DSUSE	0C000202	201327106	Specify device usage
ESACRT	000A0000	655360	Create application group
ESADEL	000B0000	720896	Delete application group
ESAQRY	000C0000	786432	Query the current application group
ESASEL	000D0000	851968	Select an application group
ESEUDS	00080000	524288	Specify encoded user default specification
ESLIB	08142000	135536640	Library management
ESPCB	081C1000	136056832	Identify program communication block
ESQCPG	00100000	1048576	Query code page of a GDDM object
ESQEUD	00120000	1179648	Query encoded user default specification
ESSCPG	00110000	1114112	Set code page of a GDDM object
ESSUDS	00070000	458752	Specify source-format user default specification
FSALRM	0C080000	201850880	Sound the terminal alarm
FSCHEK	0C100002	202375170	Check picture complexity before output
FSCLS	0C180004	202899460	Close alternate device
FSCOPY	0C180001	202899457	Send page to alternate device
FSENAB	0C040E00	201592320	Enable/disable device input
FSEXIT	00030000	196608	Specify an error exit, or error threshold, or both
FSFRCE	0C100001	202375169	Update the display
FSINIT	0C000001	201326593	Initialize GDDM processing
FSLOG	0C180003	202899459	Send character string to alternate device
FSLOGC	0C180005	202899461	Send character string with carriage-control character to alternate device
FSOPEN	0C180000	202899456	Open alternate device
FSPCLR	0C040003	201588739	Clear the current page
FSPCRT	0C040000	201588736	Create a page
FSPDEL	0C040002	201588738	Delete a page
FSPQRY	0C040004	201588740	Query specified page
FSPSEL	0C040001	201588737	Select a page
FSPWIN	0C040C00	201591808	Set page window
FSQCPG	0C040005	201588741	Query current page identifier
FSQDEV	0C040500	201590016	Query device characteristics
FSQERR	00040000	262144	Query last error
FSQSYS	00060000	393216	Query systems environment
FSQUPD	0C0C1A01	202119681	Query update mode
FSQUPG	0C040900	201591040	Query unique page identifier
FSQURY	0C040501	201590017	Query device characteristics
FSQWIN	0C040C01	201591809	Query page window

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
FSREST	0C080C00	201853952	Retransmit data
FSRNIT	0C000002	201326594	Reinitialize GDDM
FSSAVE	0C100004	202375172	Save current page contents
FSSHOR	0C100007	202375175	Extended FSSHOW
FSSHOW	0C100005	202375173	Display a saved picture
FSTERM	0C000000	201326592	Terminate GDDM processing
FSTRAN	000F0000	983040	Translate character string
FSTRCE	00020000	131072	Control internal trace
FSUPDM	0C0C1A00	202119680	Set update mode
GSAM	0C0C1311	202117905	Set attribute mode
GSARC	0C0C0600	202114560	Draw a circular arc
GSARCC	0C0C000B	202113035	Specify aspect-ratio control (for copy)
GSAREA	0C0C0408	202114056	Start a shaded area
GSBMIX	0C0C1317	202117911	Set current background color-mixing mode
GSBND	0C0C000D	202113037	Define a data boundary
GSCA	0C0C0708	202114824	Set current character angle
GSCALL	0C0C1402	202118146	Call a segment
GSCB	0C0C0707	202114823	Set character-box size
GSCBS	0C0C130F	202117903	Set character-box spacing
GSCD	0C0C0709	202114825	Set current character direction
GSCH	0C0C070C	202114828	Set current character shear
GSCHAP	0C0C0501	202114305	Draw a character string at current position
GSCHAR	0C0C0500	202114304	Draw a character string at a specified point
GSCLP	0C0C0203	202113539	Enable and disable clipping
GSCLR	0C0C0303	202113795	Clear the graphics field
GSCM	0C0C0705	202114821	Set current character mode
GSCOL	0C0C0701	202114817	Set current color
GSCOPY	0C180002	202899458	Send graphics to alternate device
GSCORR	0C0C1500	202118400	Explicit correlation of tag to primitive
GSCORS	0C0C1501	202118401	Explicit correlation of structure
GSCP	0C0C1319	202117913	Set current position
GSCPG	0C040D00	201592064	Set current code page
GSCS	0C0C0706	202114822	Set current symbol set
GSDEFE	0C0C1901	202119425	End drawing defaults definition
GSDEFS	0C0C1900	202119424	Start the drawing defaults definition
GSDSS	0C040301	201589505	Load a graphics symbol set from the application program
GSELPS	0C0C0601	202114561	Draw an elliptic arc
GSENAB	0C0C0D00	202116352	Enable or disable a logical input device
GSENDA	0C0C0409	202114057	End a shaded area
GSFLD	0C0C0000	202113024	Define the graphics field
GSFLSH	0C0C0E00	202116608	Clear the graphics input queue
GSFLW	0C0C070E	202114830	Set current fractional line width
GSGET	0C0C0B02	202115842	Retrieve graphics data
GSGETE	0C0C0B01	202115841	End retrieval of graphics data
GSGETS	0C0C0B00	202115840	Start retrieval of graphics data
GSIDVF	0C0C0C05	202116101	Initial data value, float
GSIDVI	0C0C0C04	202116100	Initial data value, integer
GSILOC	0C0C0C00	202116096	Initialize locator
GSIMG	0C0C0A00	202115584	Draw a graphics image
GSIMGS	0C0C0A04	202115588	Draw a scaled graphics image
GSIPIK	0C0C0C01	202116097	Initialize pick device
GSISTK	0C0C0C07	202116103	Initialize stroke device
GSISTR	0C0C0C06	202116102	Initialize string device
GSLINE	0C0C0401	202114049	Draw a straight line
GSLOAD	0C0C1201	202117633	Load segments
GSLSS	0C040300	201589504	Load a graphics symbol set from auxiliary storage
GSLT	0C0C0703	202114819	Set current line type
GSLW	0C0C0704	202114820	Set current line width
GSMARK	0C0C0406	202114054	Draw a marker symbol
GSMB	0C0C1307	202117895	Set marker-box size
GSMIX	0C0C0702	202114818	Set current foreground color-mixing mode
GSMOVE	0C0C0400	202114048	Move without drawing
GSMRKS	0C0C0407	202114055	Draw a series of marker symbols
GSMS	0C0C070B	202114827	Set the current type of marker symbol
GSMSC	0C0C071D	202114845	Set marker scale
GSPAT	0C0C070A	202114826	Set current shading pattern

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
GSPFLT	0C0C0602	202114562	Draw a curved fillet
GSPLNE	0C0C0402	202114050	Draw a series of lines
GSPOP	0C0C1313	202117907	Restore attributes
GSPS	0C0C0001	202113025	Define the picture space
GSPUT	0C0C0900	202115328	Restore graphics data
GSQAGA	0C0C1104	202117380	Query all geometric attributes
GSQAM	0C0C1312	202117906	Query the current attribute mode
GSQATI	0C0C030A	202113802	Query initial segment attributes
GSQATS	0C0C030C	202113804	Query segment attributes
GSQBMX	0C0C1316	202117910	Query the current background color-mixing mode
GSQBND	0C0C000E	202113038	Query the current data boundary definition
GSQCA	0C0C0718	202114840	Query character angle
GSQCB	0C0C0717	202114839	Query character-box size
GSQCBS	0C0C1310	202117904	Query character-box spacing
GSQCD	0C0C0719	202114841	Query character direction
GSQCEL	0C0C0202	202113538	Query default graphics cell size
GSQCH	0C0C071C	202114844	Query character shear
GSQCHO	0C0C0F00	202116864	Query choice device data
GSQCLP	0C0C0204	202113540	Query the clipping state
GSQCM	0C0C0715	202114837	Query the current character mode
GSQCOL	0C0C0711	202114833	Query the current color
GSQCP	0C0C0700	202114816	Query the current position
GSQCPG	0C040D01	201592065	Query code page
GSQCS	0C0C0716	202114838	Query the current symbol-set identifier
GSQCUR	0C0C0101	202113281	Query the cursor position
GSQFLD	0C0C000A	202113034	Query the graphics field
GSQFLW	0C0C070F	202114831	Query the current fractional line width
GSQLID	0C0C0C09	202116105	Query logical input device
GSQLOC	0C0C0F01	202116865	Query graphics locator data
GSQLT	0C0C0713	202114835	Query the current line type
GSQLW	0C0C0714	202114836	Query the current line width
GSQMAX	0C0C0100	202113280	Query the number of segments
GSQMB	0C0C1308	202117896	Query marker box
GSQMIX	0C0C0712	202114834	Query the current color mixing mode
GSQMS	0C0C071B	202114843	Query the current marker symbol
GSQMSC	0C0C071E	202114846	Query marker scale
GSQNSS	0C040102	201588994	Query the number of loaded symbol sets
GSQORG	0C0C0316	202113814	Query segment origin
GSQPAT	0C0C071A	202114842	Query the current shading pattern
GSQPIK	0C0C0F02	202116866	Query pick data
GSQPKS	0C0C0F05	202116869	Query pick structure
GSQPOS	0C0C030E	202113806	Query segment position
GSQPRI	0C0C0313	202113811	Query segment priority
GSQPS	0C0C0004	202113028	Query the picture-space definition
GSQSEN	0C0C1B01	202119937	Query mixed string attribute of graphics text
GSQSIM	0C0C0E01	202116609	Query existence of simultaneous queue entry
GSQSS	0C040103	201588995	Query loaded symbol sets
GSQSSD	0C0C0102	202113282	Query symbol set data
GSQSTK	0C0C0F04	202116868	Query stroke data
GSQSTR	0C0C0F03	202116867	Query string data
GSQSVL	0C0C1315	202117909	Query the current segment viewing limits
GSQTA	0C0C130E	202117902	Query the current text alignment
GSQTAG	0C0C1001	202117121	Query current tag
GSQTB	0C0C0502	202114306	Query the text box
GSQTFM	0C0C1105	202117381	Query segment transform
GSQVIE	0C0C0005	202113029	Query the current viewport definition
GSQWIN	0C0C0006	202113030	Query the current window definition
GSREAD	0C100003	202375171	Await graphics input
GSRSS	0C040401	201589761	Release a graphics symbol set
GSSAGA	0C0C1102	202117378	Set all geometric attributes
GSSATI	0C0C0309	202113801	Set initial segment attributes
GSSATS	0C0C030B	202113803	Modify segment attributes
GSSAVE	0C0C1200	202117632	Save a segment
GSSCLS	0C0C0301	202113793	Close the current segment
GSSCPY	0C0C1400	202118144	Copy a segment
GSSCT	0C0C1107	202117383	Set current transform

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
GSSDEL	0C0C0302	202113794	Delete a segment
GSSEG	0C0C0300	202113792	Create a segment
GSSEN	0C0C1B00	202119936	Set mixed string attribute of graphics text
GSSINC	0C0C1401	202118145	Include a segment
GSSORG	0C0C0311	202113809	Set segment origin
GSSPOS	0C0C030D	202113805	Set segment position
GSSPRI	0C0C0312	202113810	Set segment priority
GSSTFM	0C0C1103	202117379	Set segment transform
GSSVL	0C0C1314	202117908	Define segment viewing limits
GSTA	0C0C130D	202117901	Set text alignment
GSTAG	0C0C1000	202117120	Set current primitive tag
GSUWIN	0C0C0007	202113031	Define a uniform graphics window
GSVECM	0C0C040A	202114058	Vectors
GSVIEW	0C0C0003	202113027	Define a viewport
GSWIN	0C0C0002	202113026	Define a graphics window
IMACLR	3C010008	1006698504	Clear a rectangle in an image
IMACRT	3C010001	1006698497	Create an image
IMADEL	3C010007	1006698503	Delete the image associated with the identifier
IMAGID	3C010002	1006698498	Get and reserve a unique image identifier
IMAGT	3C010015	1006698517	Retrieve image data from an image
IMAGTE	3C010016	1006698518	End retrieval of data from an image
IMAGTS	3C010014	1006698516	Start retrieval of data from an image
IMAPT	3C010012	1006698514	Enter data into an image
IMAPTE	3C010013	1006698515	End data entry into an image
IMAPTS	3C010011	1006698513	Start data entry into an image
IMAQRY	3C010004	1006698500	Query attributes of an image
IMARES	3C010006	1006698502	Convert the resolution attributes of an image
IMARF	3C01000C	1006698508	Change resolution flag of an image
IMARST	3C01000B	1006698507	Restore image from auxiliary storage
IMASAV	3C01000A	1006698506	Save image on auxiliary storage
IMATRM	3C010009	1006698505	Trim an image down to the specified rectangle
IMPCRT	3C030003	1006829571	Create an empty projection
IMPDEL	3C030004	1006829572	Delete projection
IMPGID	3C030001	1006829569	Get and reserve a unique projection identifier
IMPRST	3C030006	1006829574	Restore projection from auxiliary storage
IMPSAV	3C030005	1006829573	Save projection on auxiliary storage
IMRBRI	3C030202	1006830082	Define brightness conversion algorithm
IMRCON	3C030203	1006830083	Define contrast conversion algorithm
IMRCVB	3C030201	1006830081	Define bi-level conversion algorithm
IMREX	3C030101	1006829825	Define rectangular sub-image in pixel coordinates
IMREXR	3C030102	1006829826	Define rectangular sub-image in real coordinates
IMRNEG	3C030109	1006829833	Negate the pixels of an extracted image
IMRORN	3C030107	1006829831	Turn an extracted image clockwise through a number of right angles
IMRPL	3C030103	1006829827	Define place position in pixel coordinates
IMRPLR	3C030204	1006830084	Define place position in real coordinates
IMRRAL	3C030106	1006829830	Set current resolution/scaling algorithm
IMRREF	3C030108	1006829832	Reflect extracted image
IMRSCL	3C030105	1006829829	Scale extracted image
IMXFER	3C010017	1006698519	Transfer data between two images, applying a projection
ISCTL	0C300002	204472322	Set image quality-control parameters
ISENAB	0C301200	204476928	Enable or disable image cursor
ISESCA	0C300B00	204475136	Control echoing of scanner image
ISFLD	0C300000	204472320	Define image field
ISIBOX	0C301600	204477952	Initialize image box cursor
ISILOC	0C301400	204477440	Initialize image locator cursor
ISLDE	0C300C00	204475392	Load external read-only image
ISQBOX	0C301500	204477696	Query image box cursor
ISQCOM	0C301800	204478464	Query image compressions supported by the device
ISQFLD	0C300001	204472321	Query image field
ISQFOR	0C301700	204478208	Query image formats supported by the device
ISQLOC	0C301300	204477184	Query image locator cursor position
ISQRES	0C300E00	204475904	Query supported image resolutions
ISQSCA	0C300D00	204475648	Query image scanner device
ISXCTL	0C300003	204472323	Extended set image quality control parameters
MSCPOS	0C280600	203949568	Set cursor position
MSDFLD	0C280500	203949312	Create or delete a mapped field

Call name	RCP code (Hex.)	RCP code (Decimal)	Function
MSGET	0C280502	203949314	Retrieve data from a map
MSPCRT	0C280100	203948288	Create a page for mapping
MSPQRY	0C040006	201588742	Query current page
MSPUT	0C280501	203949313	Place data into a mapped field
MSQADS	0C280302	203948802	Query application data structure definition
MSQFIT	0C280303	203948803	Query map fit
MSQFLD	0C280503	203949315	Query mapped field characteristics
MSQGRP	0C280300	203948800	Query mapgroup characteristics
MSQMAP	0C280301	203948801	Query map characteristics
MSQMOD	0C280400	203949056	Query modified fields
MSQPOS	0C280601	203949569	Query cursor position
MSREAD	0C280000	203948032	Present mapped data
PSDSS	0C040202	201589250	Load a symbol set into a PS store from the application program
PSLSS	0C040200	201589248	Load a symbol set into a PS store from auxiliary storage
PSLSSC	0C040201	201589249	Conditionally load a symbol set into a PS store from auxiliary storage
PSQSS	0C040101	201588993	Query status of device stores
PSRSS	0C040400	201589760	Release a symbol set from a PS store
PSRSV	0C040203	201589251	Reserving or releasing a PS store
PTNCRT	0C240000	203685888	Create a partition
PTNDEL	0C240101	203686145	Delete a partition
PTNMOD	0C240002	203685890	Modify the current partition
PTNQRY	0C240001	203685889	Query the current partition
PTNQUN	0C240102	203686146	Query unique partition identifier
PTNSEL	0C240100	203686144	Select a partition
PTSCRT	0C200000	203423744	Create a partition set
PTSDEL	0C200101	203424001	Delete a partition set
PTSQPI	0C200400	203424768	Query partition identifiers
PTSQPN	0C200401	203424769	Query partition numbers
PTSQPP	0C200301	203424513	Query partition viewing priorities
PTSQRY	0C200001	203423745	Query partition set attributes
PTSQUN	0C200102	203424002	Query unique partition set identifier
PTSSEL	0C200100	203424000	Select a partition set
PTSSPP	0C200300	203424512	Set partition viewing priorities
SPINIT	00050000	327680	Initialize GDDM with SPIB
SPMXMP	0C081401	201856001	Control the use of mixed fields by mapping
SSQF	0C040100	201588992	Query a symbol set on auxiliary storage
SSREAD	0C040B00	201591552	Read a symbol set from auxiliary storage
SSWRT	0C040B01	201591553	Write a symbol set to auxiliary storage
WSCRT	0C2C0000	204210176	Create an operator window
WSDEL	0C2C0100	204210432	Delete operator window
WSIO	0C100008	202375176	Windowed device input/output
WSMOD	0C2C0200	204210688	Modify the current operator window
WSQRY	0C2C0300	204210944	Query the current operator window
WSQUN	0C2C0400	204211200	Query unique operator window identifier
WSQWI	0C2C0500	204211456	Query operator window identifiers
WSQWN	0C2C0600	204211712	Query operator window numbers
WSQWP	0C2C0700	204211968	Query operator window viewing priorities
WSSEL	0C2C0800	204212224	Select an operator window
WSSWP	0C2C0900	204212480	Set operator window viewing priorities

GDDM Base RCP codes, listed numerically

This table lists the GDDM Base RCP codes in numeric order of RCP code.

The table on page 231 lists the GDDM Base RCP codes in alphabetic order of call name.

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
00020000	131072	FSTRCE	Control internal trace
00030000	196608	FSEXIT	Specify an error exit, or error threshold, or both
00040000	262144	FSQERR	Query last error
00050000	327680	SPINIT	Initialize GDDM with SPIB
00060000	393216	FSQSYS	Query systems environment
00070000	458752	ESSUDS	Specify source-format user default specification
00080000	524288	ESEUDS	Specify encoded user default specification
000A0000	655360	ESACRT	Create application group
000B0000	720896	ESADEL	Delete application group
000C0000	786432	ESAQRY	Query the current application group
000D0000	851968	ESASEL	Select an application group
000F0000	983040	FSTRAN	Translate character string
00100000	1048576	ESQCPG	Query code page of a GDDM object
00110000	1114112	ESSCPG	Set code page of a GDDM object
00120000	1179648	ESQEUD	Query encoded user default specification
08142000	135536640	ESLIB	Library management
081C1000	136056832	ESPCB	Identify program communication block
0C000000	201326592	FSTERM	Terminate GDDM processing
0C000001	201326593	FSINIT	Initialize GDDM processing
0C000002	201326594	FSRNIT	Reinitialize GDDM
0C000200	201327104	DSOPEN	Open a device
0C000201	201327105	DSCLS	Close a device
0C000202	201327106	DSUSE	Specify device usage
0C000203	201327107	DSDROP	Discontinue device usage
0C000204	201327108	DSQUID	Query unique device identifier
0C000205	201327109	DSQUSE	Query device usage
0C000206	201327110	DSQDEV	Query device characteristics
0C000207	201327111	DSRNIT	Reinitialize a device
0C040000	201588736	FSPCRT	Create a page
0C040001	201588737	FSPSEL	Select a page
0C040002	201588738	FSPDEL	Delete a page
0C040003	201588739	FSPCLR	Clear the current page
0C040004	201588740	FSPQRY	Query specified page
0C040005	201588741	FSQCPG	Query current page identifier
0C040006	201588742	MSPQRY	Query current page
0C040100	201588992	SSQF	Query a symbol set on auxiliary storage
0C040101	201588993	PSQSS	Query status of device stores
0C040102	201588994	GSQNSS	Query the number of loaded symbol sets
0C040103	201588995	GSQSS	Query loaded symbol sets
0C040200	201589248	PSLSS	Load a symbol set into a PS store from auxiliary storage
0C040201	201589249	PSLSSC	Conditionally load a symbol set into a PS store from auxiliary storage
0C040202	201589250	PSDSS	Load a symbol set into a PS store from the application program
0C040203	201589251	PSRSV	Reserving or releasing a PS store
0C040300	201589504	GSLSS	Load a graphics symbol set from auxiliary storage
0C040301	201589505	GSDSS	Load a graphics symbol set from the application program
0C040400	201589760	PSRSS	Release a symbol set from a PS store
0C040401	201589761	GSRSS	Release a graphics symbol set
0C040500	201590016	FSQDEV	Query device characteristics
0C040501	201590017	FSQURY	Query device characteristics
0C040900	201591040	FSQUPG	Query unique page identifier
0C040B00	201591552	SSREAD	Read a symbol set from auxiliary storage
0C040B01	201591553	SSWRT	Write a symbol set to auxiliary storage
0C040C00	201591808	FSPWIN	Set page window
0C040C01	201591809	FSQWIN	Query page window
0C040D00	201592064	GSCPG	Set current code page
0C040D01	201592065	GSQCPG	Query code page
0C040E00	201592320	FSENAB	Enable/disable device input
0C080000	201850880	FSALRM	Sound the terminal alarm
0C080100	201851136	ASFCUR	Position the cursor
0C080200	201851392	ASDFLT	Set default field attributes

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
0C080300	201851648	ASDTRN	Define I/O translation tables
0C080400	201851904	ASFCLR	Clear fields
0C080500	201852160	ASFTYP	Define field type
0C080501	201852161	ASFINT	Define field intensity
0C080502	201852162	ASFCOL	Define field color
0C080503	201852163	ASFPSS	Define primary symbol set for a field
0C080504	201852164	ASFHLT	Define field highlighting
0C080505	201852165	ASFEND	Define field end attribute
0C080506	201852166	ASFOUT	Define output blank-to-null conversion
0C080507	201852167	ASFIN	Define input null-to-blank conversion
0C080508	201852168	ASFTRN	Assign translation table set to a field
0C080509	201852169	ASFTRA	Define field transparency attribute
0C08050A	201852170	ASFSEN	Define field mixed-string attribute
0C08050B	201852171	ASFBDY	Define field outline
0C080600	201852416	ASCHLT	Specify character highlights within a field
0C080601	201852417	ASCCOL	Specify character colors within a field
0C080602	201852418	ASCSS	Specify character symbol sets within a field
0C080603	201852419	ASCPUT	Specify field contents
0C080700	201852672	ASDFLD	Define or delete a single field
0C080800	201852928	ASRFMT	Define multiple fields without deleting existing fields
0C080801	201852929	ASDFMT	Define alphanumeric fields, deleting all existing fields
0C080802	201852930	ASRATT	Define field attributes
0C080900	201853184	ASQHLT	Query character highlights for a field
0C080901	201853185	ASQCOL	Query character colors for a field
0C080902	201853186	ASQSS	Query character symbol sets for a field
0C080903	201853187	ASCGET	Get field contents
0C080A00	201853440	ASQFLD	Query field attributes
0C080B00	201853696	ASQMOD	Query modified fields
0C080C00	201853952	FSREST	Retransmit data
0C080C01	201853953	DSCMF	User Control function
0C080C02	201853954	DSQCMF	Query user control function
0C080D00	201854208	ASMODE	Define the operator reply mode
0C080E00	201854464	ASQMAX	Query the number of fields
0C080E01	201854465	ASQNMF	Query the number of modified fields
0C080F00	201854720	ASQCUR	Query cursor position
0C081100	201855232	ASFMOD	Change field status
0C081300	201855744	ASTYPE	Override alphanumeric character-code assignments
0C081401	201856001	SPMXMP	Control the use of mixed fields by mapping
0C081503	201856259	ASGPUT	Specify double-character field contents
0C081603	201856515	ASGGET	Get double-character field contents
0C081800	201857024	ASQLEN	Query length of field contents
0C0C0000	202113024	GSFLD	Define the graphics field
0C0C0001	202113025	GSPS	Define the picture space
0C0C0002	202113026	GSWIN	Define a graphics window
0C0C0003	202113027	GSVIEW	Define a viewport
0C0C0004	202113028	GSQPS	Query the picture-space definition
0C0C0005	202113029	GSQVIE	Query the current viewport definition
0C0C0006	202113030	GSQWIN	Query the current window definition
0C0C0007	202113031	GSUWIN	Define a uniform graphics window
0C0C000A	202113034	GSQFLD	Query the graphics field
0C0C000B	202113035	GSARCC	Specify aspect-ratio control (for copy)
0C0C000D	202113037	GSBND	Define a data boundary
0C0C000E	202113038	GSQBND	Query the current data boundary definition
0C0C0100	202113280	GSQMAX	Query the number of segments
0C0C0101	202113281	GSQCUR	Query the cursor position
0C0C0102	202113282	GSQSSD	Query symbol set data
0C0C0202	202113538	GSQCEL	Query default graphics cell size
0C0C0203	202113539	GSCLP	Enable and disable clipping
0C0C0204	202113540	GSQCLP	Query the clipping state
0C0C0300	202113792	GSSEG	Create a segment
0C0C0301	202113793	GSSCLS	Close the current segment
0C0C0302	202113794	GSSDEL	Delete a segment
0C0C0303	202113795	GSCLR	Clear the graphics field
0C0C0309	202113801	GSSATI	Set initial segment attributes
0C0C030A	202113802	GSQATI	Query initial segment attributes
0C0C030B	202113803	GSSATS	Modify segment attributes

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
0C0C030C	202113804	GSQATS	Query segment attributes
0C0C030D	202113805	GSSPOS	Set segment position
0C0C030E	202113806	GSQPOS	Query segment position
0C0C0311	202113809	GSSORG	Set segment origin
0C0C0312	202113810	GSSPRI	Set segment priority
0C0C0313	202113811	GSQPRI	Query segment priority
0C0C0316	202113814	GSQORG	Query segment origin
0C0C0400	202114048	GSMOVE	Move without drawing
0C0C0401	202114049	GSLINE	Draw a straight line
0C0C0402	202114050	GSPLNE	Draw a series of lines
0C0C0406	202114054	GSMARK	Draw a marker symbol
0C0C0407	202114055	GSMRKS	Draw a series of marker symbols
0C0C0408	202114056	GSAREA	Start a shaded area
0C0C0409	202114057	GSENDA	End a shaded area
0C0C040A	202114058	GSVECM	Vectors
0C0C0500	202114304	GSCHAR	Draw a character string at a specified point
0C0C0501	202114305	GSCHAP	Draw a character string at current position
0C0C0502	202114306	GSQTB	Query the text box
0C0C0600	202114560	GSARC	Draw a circular arc
0C0C0601	202114561	GSELPS	Draw an elliptic arc
0C0C0602	202114562	GSPFLT	Draw a curved fillet
0C0C0700	202114816	GSQCP	Query the current position
0C0C0701	202114817	GSCOL	Set current color
0C0C0702	202114818	GSMIX	Set current foreground color-mixing mode
0C0C0703	202114819	GSLT	Set current line type
0C0C0704	202114820	GSLW	Set current line width
0C0C0705	202114821	GSCM	Set current character mode
0C0C0706	202114822	GSCS	Set current symbol set
0C0C0707	202114823	GSCB	Set character-box size
0C0C0708	202114824	GSCA	Set current character angle
0C0C0709	202114825	GSCD	Set current character direction
0C0C070A	202114826	GSPAT	Set current shading pattern
0C0C070B	202114827	GSMS	Set the current type of marker symbol
0C0C070C	202114828	GSCH	Set current character shear
0C0C070E	202114830	GSFLW	Set current fractional line width
0C0C070F	202114831	GSQFLW	Query the current fractional line width
0C0C0711	202114833	GSQCOL	Query the current color
0C0C0712	202114834	GSQMIX	Query the current color mixing mode
0C0C0713	202114835	GSQLT	Query the current line type
0C0C0714	202114836	GSQLW	Query the current line width
0C0C0715	202114837	GSQCM	Query the current character mode
0C0C0716	202114838	GSQCS	Query the current symbol-set identifier
0C0C0717	202114839	GSQCB	Query character-box size
0C0C0718	202114840	GSQCA	Query character angle
0C0C0719	202114841	GSQCD	Query character direction
0C0C071A	202114842	GSQPAT	Query the current shading pattern
0C0C071B	202114843	GSQMS	Query the current marker symbol
0C0C071C	202114844	GSQCH	Query character shear
0C0C071D	202114845	GSMSC	Set marker scale
0C0C071E	202114846	GSQMSC	Query marker scale
0C0C0900	202115328	GSPUT	Restore graphics data
0C0C0A00	202115584	GSIMG	Draw a graphics image
0C0C0A04	202115588	GSIMGS	Draw a scaled graphics image
0C0C0B00	202115840	GSGETS	Start retrieval of graphics data
0C0C0B01	202115841	GSGETE	End retrieval of graphics data
0C0C0B02	202115842	GSGET	Retrieve graphics data
0C0C0C00	202116096	GSILOC	Initialize locator
0C0C0C01	202116097	GSIPIK	Initialize pick device
0C0C0C04	202116100	GSIDVI	Initial data value, integer
0C0C0C05	202116101	GSIDVF	Initial data value, float
0C0C0C06	202116102	GSISTR	Initialize string device
0C0C0C07	202116103	GSISTK	Initialize stroke device
0C0C0C09	202116105	GSQLID	Query logical input device
0C0C0D00	202116352	GSENAB	Enable or disable a logical input device
0C0C0E00	202116608	GSFLSH	Clear the graphics input queue
0C0C0E01	202116609	GSQSIM	Query existence of simultaneous queue entry

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
0C0C0F00	202116864	GSQCHO	Query choice device data
0C0C0F01	202116865	GSQLOC	Query graphics locator data
0C0C0F02	202116866	GSQPIK	Query pick data
0C0C0F03	202116867	GSQSTR	Query string data
0C0C0F04	202116868	GSQSTK	Query stroke data
0C0C0F05	202116869	GSQPKS	Query pick structure
0C0C1000	202117120	GSTAG	Set current primitive tag
0C0C1001	202117121	GSQTAG	Query current tag
0C0C1102	202117378	GSSAGA	Set all geometric attributes
0C0C1103	202117379	GSSTFM	Set segment transform
0C0C1104	202117380	GSQAGA	Query all geometric attributes
0C0C1105	202117381	GSQTFM	Query segment transform
0C0C1107	202117383	GSSCT	Set current transform
0C0C1200	202117632	GSSAVE	Save a segment
0C0C1201	202117633	GSLOAD	Load segments
0C0C1307	202117895	GSMB	Set marker-box size
0C0C1308	202117896	GSQMB	Query marker box
0C0C130D	202117901	GSTA	Set text alignment
0C0C130E	202117902	GSQTA	Query the current text alignment
0C0C130F	202117903	GSCBS	Set character-box spacing
0C0C1310	202117904	GSQCBS	Query character-box spacing
0C0C1311	202117905	GSAM	Set attribute mode
0C0C1312	202117906	GSQAM	Query the current attribute mode
0C0C1313	202117907	GSPOP	Restore attributes
0C0C1314	202117908	GSSVL	Define segment viewing limits
0C0C1315	202117909	GSQSVL	Query the current segment viewing limits
0C0C1316	202117910	GSQBMX	Query the current background color-mixing mode
0C0C1317	202117911	GSBMIX	Set current background color-mixing mode
0C0C1319	202117913	GSCP	Set current position
0C0C1400	202118144	GSSCPY	Copy a segment
0C0C1401	202118145	GSSINC	Include a segment
0C0C1402	202118146	GSCALL	Call a segment
0C0C1500	202118400	GSCORR	Explicit correlation of tag to primitive
0C0C1501	202118401	GSCORS	Explicit correlation of structure
0C0C1900	202119424	GSDEFS	Start the drawing defaults definition
0C0C1901	202119425	GSDEFE	End drawing defaults definition
0C0C1A00	202119680	FSUPDM	Set update mode
0C0C1A01	202119681	FSQUPD	Query update mode
0C0C1B00	202119936	GSSEN	Set mixed string attribute of graphics text
0C0C1B01	202119937	GSQSEN	Query mixed string attribute of graphics text
0C100000	202375168	ASREAD	Device output/input
0C100001	202375169	FSFRCE	Update the display
0C100002	202375170	FSCHEK	Check picture complexity before output
0C100003	202375171	GSREAD	Await graphics input
0C100004	202375172	FSSAVE	Save current page contents
0C100005	202375173	FSSHOW	Display a saved picture
0C100007	202375175	FSSHOR	Extended FSSHOW
0C100008	202375176	WSIO	Windowed device input/output
0C180000	202899456	FSOPEN	Open alternate device
0C180001	202899457	FSCOPY	Send page to alternate device
0C180002	202899458	GSCOPY	Send graphics to alternate device
0C180003	202899459	FSLOG	Send character string to alternate device
0C180004	202899460	FSCLS	Close alternate device
0C180005	202899461	FSLOGC	Send character string with carriage-control character to alternate device
0C200000	203423744	PTSCRT	Create a partition set
0C200001	203423745	PTSQRY	Query partition set attributes
0C200100	203424000	PTSSEL	Select a partition set
0C200101	203424001	PTSDEL	Delete a partition set
0C200102	203424002	PTSQUN	Query unique partition set identifier
0C200300	203424512	PTSSPP	Set partition viewing priorities
0C200301	203424513	PTSQPP	Query partition viewing priorities
0C200400	203424768	PTSQPI	Query partition identifiers
0C200401	203424769	PTSQPN	Query partition numbers
0C240000	203685888	PTNCRT	Create a partition
0C240001	203685889	PTNQRY	Query the current partition
0C240002	203685890	PTNMOD	Modify the current partition

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
0C240100	203686144	PTNSEL	Select a partition
0C240101	203686145	PTNDEL	Delete a partition
0C240102	203686146	PTNQUN	Query unique partition identifier
0C280000	203948032	MSREAD	Present mapped data
0C280100	203948288	MSPCRT	Create a page for mapping
0C280300	203948800	MSQGRP	Query mapgroup characteristics
0C280301	203948801	MSQMAP	Query map characteristics
0C280302	203948802	MSQADS	Query application data structure definition
0C280303	203948803	MSQFIT	Query map fit
0C280400	203949056	MSQMOD	Query modified fields
0C280500	203949312	MSDFLD	Create or delete a mapped field
0C280501	203949313	MSPUT	Place data into a mapped field
0C280502	203949314	MSGET	Retrieve data from a map
0C280503	203949315	MSQFLD	Query mapped field characteristics
0C280600	203949568	MSCPOS	Set cursor position
0C280601	203949569	MSQPOS	Query cursor position
0C2C0000	204210176	WSCRT	Create an operator window
0C2C0100	204210432	WSDEL	Delete operator window
0C2C0200	204210688	WSMOD	Modify the current operator window
0C2C0300	204210944	WSQRY	Query the current operator window
0C2C0400	204211200	WSQUN	Query unique operator window identifier
0C2C0500	204211456	WSQWI	Query operator window identifiers
0C2C0600	204211712	WSQWN	Query operator window numbers
0C2C0700	204211968	WSQWP	Query operator window viewing priorities
0C2C0800	204212224	WSSEL	Select an operator window
0C2C0900	204212480	WSSWP	Set operator window viewing priorities
0C300000	204472320	ISFLD	Define image field
0C300001	204472321	ISQFLD	Query image field
0C300002	204472322	ISCTL	Set image quality-control parameters
0C300003	204472323	ISXCTL	Extended set image quality control parameters
0C300B00	204475136	ISESCA	Control echoing of scanner image
0C300C00	204475392	ISLDE	Load external read-only image
0C300D00	204475648	ISQSCA	Query image scanner device
0C300E00	204475904	ISQRES	Query supported image resolutions
0C301200	204476928	ISENAB	Enable or disable image cursor
0C301300	204477184	ISQLOC	Query image locator cursor position
0C301400	204477440	ISILOC	Initialize image locator cursor
0C301500	204477696	ISQBOX	Query image box cursor
0C301600	204477952	ISIBOX	Initialize image box cursor
0C301700	204478208	ISQFOR	Query image formats supported by the device
0C301800	204478464	ISQCOM	Query image compressions supported by the device
0C380000	204996608	APDEF	Define a field list
0C380100	204996864	APDEL	Delete a field list
0C380200	204997120	APMOD	Modify a field list
0C380300	204997376	APQIDS	Query field list identifiers
0C380400	204997632	APQNUM	Query field list numbers
0C380500	204997888	APQRY	Query a field list
0C380600	204998144	APQSIZ	Query a field list size
0C380700	204998400	APQUID	Query unique field list identifier
3C010001	1006698497	IMACRT	Create an image
3C010002	1006698498	IMAGID	Get and reserve a unique image identifier
3C010004	1006698500	IMAQRY	Query attributes of an image
3C010006	1006698502	IMARES	Convert the resolution attributes of an image
3C010007	1006698503	IMADEL	Delete the image associated with the identifier
3C010008	1006698504	IMACLR	Clear a rectangle in an image
3C010009	1006698505	IMATRM	Trim an image down to the specified rectangle
3C01000A	1006698506	IMASAV	Save image on auxiliary storage
3C01000B	1006698507	IMARST	Restore image from auxiliary storage
3C01000C	1006698508	IMARF	Change resolution flag of an image
3C010011	1006698513	IMAPTS	Start data entry into an image
3C010012	1006698514	IMAPT	Enter data into an image
3C010013	1006698515	IMAPTE	End data entry into an image
3C010014	1006698516	IMAGTS	Start retrieval of data from an image
3C010015	1006698517	IMAGT	Retrieve image data from an image
3C010016	1006698518	IMAGTE	End retrieval of data from an image
3C010017	1006698519	IMXFER	Transfer data between two images, applying a projection

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
3C030001	1006829569	IMPGID	Get and reserve a unique projection identifier
3C030003	1006829571	IMPCRT	Create an empty projection
3C030004	1006829572	IMPDEL	Delete projection
3C030005	1006829573	IMPSAV	Save projection on auxiliary storage
3C030006	1006829574	IMPRST	Restore projection from auxiliary storage
3C030101	1006829825	IMREX	Define rectangular sub-image in pixel coordinates
3C030102	1006829826	IMREXR	Define rectangular sub-image in real coordinates
3C030103	1006829827	IMRPL	Define place position in pixel coordinates
3C030105	1006829829	IMRSCL	Scale extracted image
3C030106	1006829830	IMRRAL	Set current resolution/scaling algorithm
3C030107	1006829831	IMRORN	Turn an extracted image clockwise through a number of right angles
3C030108	1006829832	IMRREF	Reflect extracted image
3C030109	1006829833	IMRNEG	Negate the pixels of an extracted image
3C030201	1006830081	IMRCVB	Define bi-level conversion algorithm
3C030202	1006830082	IMRBRI	Define brightness conversion algorithm
3C030203	1006830083	IMRCON	Define contrast conversion algorithm
3C030204	1006830084	IMRPLR	Define place position in real coordinates
40000000	1073741824	CDPU	Control the printing of Composite Documents

GDDM-PGF RCP codes

GDDM-PGF RCP codes, listed alphabetically

The RCP codes for GDDM-PGF are listed below in alphabetic order of call name.

The table on page 245 lists the GDDM-PGF RCP codes in numeric order of RCP code.

Notes:

1. All GDDM-PGF RCP codes are of type E, meaning that the functions can be called using all the normal call interfaces.
2. Each call has the prefix "QQ"; this has been omitted here for clarity.

Call name	RCP code (hex.)	RCP code (decimal)	Function
CHAATT	10020701	268568321	Axis line attributes
CHAREA	10020A02	268569090	Chart area
CHART	14000000	335544320	Invoke Interactive Chart Utility
CHBAR	100D0A01	269289985	Plot a bar chart
CHBARX	100D0A07	269289991	Plot a bar chart with numeric x-axis values
CHBATT	100B0A03	269158915	Set framing box attributes
CHCGRD	10020615	268568085	Basic character spacing/size
CHCOL	10020303	268567299	Component basic color table
CHCONV	10170100	269943040	Convert coordinate values
CHDATT	100B0A01	269158913	Datum line attributes
CHDCTL	10150201	269812225	Control the format of values, and the overall size of table charts
CHDRAX	10140100	269746432	Specific control of axis drawing
CHDTAB	100D0A0B	269289995	Construct a table chart
CHFINE	1002061A	268568090	Curve fitting smoothness
CHGAP	10020610	268568080	Spacing between bars
CHGATT	10020702	268568322	Grid line attributes
CHGGAP	10020611	268568081	Spacing between bar groups
CHHATT	10020901	268568833	Heading text attributes
CHHEAD	10020202	268567042	Heading text
CHHIST	100D0A02	269289986	Histograms
CHHMAR	10020612	268568082	Horizontal margins
CHKATT	10020905	268568837	Legend text attributes
CHKEY	10020201	268567041	Legend key labels
CHKEYP	10020801	268568577	Legend base position
CHKMAX	100A0619	269092377	Maximum legend width/height
CHKOFF	100A0618	269092376	Legend offsets
CHLATT	10020903	268568835	Axis label text attributes
CHLC	10020307	268567303	Component line color table
CHLT	10020302	268567298	Component line type table
CHLW	10020305	268567301	Component line width table
CHMARK	10020301	268567297	Component marker table
CHMISS	10150301	269812481	Missing values on a table chart
CHMKSC	10030C01	268635137	Set marker scale values
CHNATT	100B0904	269158660	Specify attributes for notes
CHNOFF	100B0617	269157911	Specify offsets for CHNOTE
CHNOTE	10130100	269680896	Construct a character string at a designated position
CHNUM	1002060F	268568079	Set number of components
CHPAT	10020304	268567300	Component shading pattern table
CHPCTL	10150101	269811969	Control pie chart slices
CHPEXP	10020306	268567302	Exploded slices in pie charts
CHPIE	100D0A06	269289990	Pie charts
CHPIER	10020614	268568084	Reduce pie chart size
CHPLOT	100D0A03	269289987	Line graphs and scatter plots
CHPOLR	100D0A08	269289992	Plot a polar chart
CHQARE	10180100	270008576	Query chart area
CHQPOS	10170200	269943296	Query positional information
CHQRNG	10170300	269943552	Query axis ranges
CHRNIT	10010100	268501248	Reinitialize PG routines
CHSET	10020101	268566785	Specify chart options
CHSSEG	100B0703	269158147	Set segment number
CHSTRT	10110100	269549824	Reset the processing state to state-1

Call name	RCP code (hex.)	RCP code (decimal)	Function
CHSURF	100D0A04	269289988	Surface charts
CHTATT	10020902	268568834	Axis title text attributes
CHTERM	10000100	268435712	Terminate the PG routines
CHTHRS	100B0A05	269158917	Bar value threshold limit
CHTOWR	100D0A09	269289993	Plot a tower chart
CHTPRJ	10160100	269877504	Tower chart projection
CHVATT	10020906	268568838	Attributes of values text in bar and pie charts
CHVCHR	10020616	268568086	Number of characters in bar values
CHVDIG	100B0A04	269158916	Set decimal digits for bars and tables
CHVENN	100D0A05	269289989	Venn diagram
CHVMAR	10020613	268568083	Vertical margins
CHXDAY	1002060B	268568075	X-axis day labels
CHXDLB	10020505	268567813	X-axis data labels
CHXDTM	100E060D	269354509	X-axis datum line
CHXINT	100A0603	269092355	X-axis interception point
CHXLAB	10020503	268567811	X-axis label text
CHXLAT	10020907	268568839	X-axis label attributes
CHXMTH	10020609	268568073	X-axis month labels
CHXRNG	100A0601	269092353	X-axis explicit range
CHXSCL	10020607	268568071	X-axis scale factor
CHXSEL	100F0801	269420545	X-axis selection
CHXSET	10020401	268567553	X-axis options
CHXTAT	10020909	268568841	X-axis title attributes
CHXTIC	100A0605	269092357	X-axis scale mark interval
CHXTTL	10020501	268567809	X-axis title specification
CHYDAY	1002060C	268568076	Y-axis day labels
CHYDTM	100E060E	269354510	Y-axis datum line
CHYINT	100A0604	269092356	Y-axis interception point
CHYLAB	10020504	268567812	Y-axis label text
CHYLAT	10020908	268568840	Y-axis label attributes
CHYMTH	1002060A	268568074	Y-axis month labels
CHYRNG	100A0602	269092354	Y-axis explicit range
CHYSCL	10020608	268568072	Y-axis scale factor
CHYSEL	100F0802	269420546	Y-axis selection
CHYSET	10020402	268567554	Y-axis options
CHYTAT	1002090A	268568842	Y-axis title attributes
CHYTIC	100A0606	269092358	Y-axis scale mark interval
CHYTTL	10020502	268567810	Y-axis title specification
CHZDLB	10020507	268567815	Z-axis data labels
CHZGAP	1002061B	268568091	Spacing between towers
CHZLAT	1002090B	268568843	Z-axis label attributes
CHZRNG	100A061D	269092381	Z-axis explicit range
CHZSET	10020403	268567555	Z-axis options
CHZTIC	100A061C	269092380	Z-axis scale mark interval
CSCCRT	14040000	335806464	Create a chart
CSCDEL	14040004	335806468	Delete a chart
CSCHA	14080008	336068616	Set character values for a chart
CSDEL	14000824	335546404	Delete item for a chart
CSDIR	14000020	335544352	Build object directory list
CSFLT	14080004	336068612	Set floating-point values for a chart
CSINT	14080000	336068608	Set integer values for a chart
CSLOAD	14000010	335544336	Restore saved chart information
CSNUM	14080020	336068640	Set control value for a chart
CSQCHA	140C0008	336330760	Query character values for a.chart
CSQCHL	140C000C	336330764	Query character lengths for a chart
CSQCS	14040014	335806484	Query CSxxxx call information
CSQDIR	14000024	335544356	Query object directory list
CSQFLT	140C0004	336330756	Query floating-point values for a chart
CSQINT	140C0000	336330752	Query integer values for a chart
CSQNUM	140C0020	336330784	Query control value for chart
CSQUID	14040010	335806480	Query unique chart identifier
CSQXDT	140C0010	336330768	Query independent (x) data values for a chart
CSQXSL	140C0030	336330800	Query independent (x) data selection for a chart
CSQYDT	140C0014	336330772	Query dependent (y) data values for a chart
CSQZDT	140C0018	336330776	Query data group (z) values for a chart
CSQZSL	140C0038	336330808	Query data group (z) selection for a chart

Call name	RCP code (hex.)	RCP code (decimal)	Function
CSSAVE	14000014	335544340	Save chart information
CSSICU	14000004	335544324	Start an ICU session for a chart
CSXDT	14080010	336068624	Set independent (x) data values for a chart
CSXSL	14080030	336068656	Set independent (x) data selection for a chart
CSYDT	14080014	336068628	Set dependent (y) data values for a chart
CSZDT	14080018	336068632	Set data group (z) data values for a chart
CSZSL	14080038	336068664	Set data group (z) selection for a chart

GDDM-PGF RCP codes, listed numerically

This table lists the GDDM-PGF RCP codes in numeric order of RCP code.

The table on page 243 lists the GDDM-PGF RCP codes in alphabetic order of call name.

Notes:

1. All GDDM-PGF RCP codes are of type E, meaning that the functions can be called using all the normal call interfaces.
2. Each call has the prefix "QQ"; this has been omitted here for clarity.

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
10000100	268435712	CHTERM	Terminate the PG routines
10010100	268501248	CHRNIT	Reinitialize PG routines
10020101	268566785	CHSET	Specify chart options
10020201	268567041	CHKEY	Legend key labels
10020202	268567042	CHHEAD	Heading text
10020301	268567297	CHMARK	Component marker table
10020302	268567298	CHLT	Component line type table
10020303	268567299	CHCOL	Component basic color table
10020304	268567300	CHPAT	Component shading pattern table
10020305	268567301	CHLW	Component line width table
10020306	268567302	CHPEXP	Exploded slices in pie charts
10020307	268567303	CHLC	Component line color table
10020401	268567553	CHXSET	X-axis options
10020402	268567554	CHYSET	Y-axis options
10020403	268567555	CHZSET	Z-axis options
10020501	268567809	CHXTTL	X-axis title specification
10020502	268567810	CHYTTL	Y-axis title specification
10020503	268567811	CHXLAB	X-axis label text
10020504	268567812	CHYLAB	Y-axis label text
10020505	268567813	CHXDLB	X-axis data labels
10020507	268567815	CHZDLB	Z-axis data labels
10020607	268568071	CHXSCL	X-axis scale factor
10020608	268568072	CHYSCL	Y-axis scale factor
10020609	268568073	CHXMTH	X-axis month labels
1002060A	268568074	CHYMTH	Y-axis month labels
1002060B	268568075	CHXDAY	X-axis day labels
1002060C	268568076	CHYDAY	Y-axis day labels
1002060F	268568079	CHNUM	Set number of components
10020610	268568080	CHGAP	Spacing between bars
10020611	268568081	CHGGAP	Spacing between bar groups
10020612	268568082	CHHMAR	Horizontal margins
10020613	268568083	CHVMAR	Vertical margins
10020614	268568084	CHPIER	Reduce pie chart size
10020615	268568085	CHCGRD	Basic character spacing/size
10020616	268568086	CHVCHR	Number of characters in bar values
1002061A	268568090	CHFINE	Curve fitting smoothness
1002061B	268568091	CHZGAP	Spacing between towers
10020701	268568321	CHAATT	Axis line attributes
10020702	268568322	CHGATT	Grid line attributes
10020801	268568577	CHKEYP	Legend base position
10020901	268568833	CHHATT	Heading text attributes
10020902	268568834	CHTATT	Axis title text attributes

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
10020903	268568835	CHLATT	Axis label text attributes
10020905	268568837	CHKATT	Legend text attributes
10020906	268568838	CHVATT	Attributes of values text in bar and pie charts
10020907	268568839	CHXLAT	X-axis label attributes
10020908	268568840	CHYLAT	Y-axis label attributes
10020909	268568841	CHXTAT	X-axis title attributes
1002090A	268568842	CHYTAT	Y-axis title attributes
1002090B	268568843	CHZLAT	Z-axis label attributes
10020A02	268569090	CHAREA	Chart area
10030C01	268635137	CHMKSC	Set marker scale values
100A0601	269092353	CHXRNG	X-axis explicit range
100A0602	269092354	CHYRNG	Y-axis explicit range
100A0603	269092355	CHXINT	X-axis interception point
100A0604	269092356	CHYINT	Y-axis interception point
100A0605	269092357	CHXTIC	X-axis scale mark interval
100A0606	269092358	CHYTIC	Y-axis scale mark interval
100A0618	269092376	CHKOFF	Legend offsets
100A0619	269092377	CHKMAX	Maximum legend width/height
100A061C	269092380	CHZTIC	Z-axis scale mark interval
100A061D	269092381	CHZRNG	Z-axis explicit range
100B0617	269157911	CHNOFF	Specify offsets for CHNOTE
100B0703	269158147	CHSSEG	Set segment number
100B0904	269158660	CHNATT	Specify attributes for notes
100B0A01	269158913	CHDATT	Datum line attributes
100B0A03	269158915	CHBATT	Set framing box attributes
100B0A04	269158916	CHVDIG	Set decimal digits for bars and tables
100B0A05	269158917	CHTHRS	Bar value threshold limit
100D0A01	269289985	CHBAR	Plot a bar chart
100D0A02	269289986	CHHIST	Histograms
100D0A03	269289987	CHPLOT	Line graphs and scatter plots
100D0A04	269289988	CHSURF	Surface charts
100D0A05	269289989	CHVENN	Venn diagram
100D0A06	269289990	CHPIE	Pie charts
100D0A07	269289991	CHBARX	Plot a bar chart with numeric x-axis values
100D0A08	269289992	CHPOLR	Plot a polar chart
100D0A09	269289993	CHTOWR	Plot a tower chart
100D0A0B	269289995	CHDTAB	Construct a table chart
100E060D	269354509	CHXDTM	X-axis datum line
100E060E	269354510	CHYDTM	Y-axis datum line
100F0801	269420545	CHXSEL	X-axis selection
100F0802	269420546	CHYSEL	Y-axis selection
10110100	269549824	CHSTRT	Reset the processing state to state-1
10130100	269680896	CHNOTE	Construct a character string at a designated position
10140100	269746432	CHDRAX	Specific control of axis drawing
10150101	269811969	CHPCTL	Control pie chart slices
10150201	269812225	CHDCTL	Control the format of values, and the overall size of table charts
10150301	269812481	CHMISS	Missing values on a table chart
10160100	269877504	CHTPRJ	Tower chart projection
10170100	269943040	CHCONV	Convert coordinate values
10170200	269943296	CHQPOS	Query positional information
10170300	269943552	CHQRNG	Query axis ranges
10180100	270008576	CHQARE	Query chart area
14000000	335544320	CHART	Invoke Interactive Chart Utility
14000004	335544324	CSSICU	Start an ICU session for a chart
14000010	335544336	CSLOAD	Restore saved chart information
14000014	335544340	CSSAVE	Save chart information
14000020	335544352	CSDIR	Build object directory list
14000024	335544356	CSQDIR	Query object directory list
14000824	335546404	CSDEL	Delete item for a chart
14040000	335806464	CSCCRT	Create a chart
14040004	335806468	CSCDEL	Delete a chart
14040010	335806480	CSQUID	Query unique chart identifier
14040014	335806484	CSQCS	Query CSxxxx call information
14080000	336068608	CSINT	Set integer values for a chart
14080004	336068612	CSFLT	Set floating-point values for a chart
14080008	336068616	CSCHA	Set character values for a chart

RCP code (Hex.)	RCP code (Decimal)	Call name	Function
14080010	336068624	CSXDT	Set independent (x) data values for a chart
14080014	336068628	CSYDT	Set dependent (y) data values for a chart
14080018	336068632	CSZDT	Set data group (z) data values for a chart
14080020	336068640	CSNUM	Set control value for a chart
14080030	336068656	CSXSL	Set independent (x) data selection for a chart
14080038	336068664	CSZSL	Set data group (z) selection for a chart
140C0000	336330752	CSQINT	Query integer values for a chart
140C0004	336330756	CSQFLT	Query floating-point values for a chart
140C0008	336330760	CSQCHA	Query character values for a chart
140C000C	336330764	CSQCHL	Query character lengths for a chart
140C0010	336330768	CSQXDT	Query independent (x) data values for a chart
140C0014	336330772	CSQYDT	Query dependent (y) data values for a chart
140C0018	336330776	CSQZDT	Query data group (z) values for a chart
140C0020	336330784	CSQNUM	Query control value for chart
140C0030	336330800	CSQXSL	Query independent (x) data selection for a chart
140C0038	336330808	CSQZSL	Query data group (z) selection for a chart

Appendix K. Sample programs

This appendix contains descriptions of the GDDM sample programs that are supplied with this release of GDDM. These programs may be listed by licensees of GDDM as stated in the edition notice to this volume.

These sample GDDM programs are described:

- A program that draws a simple line graph. This program is provided in three languages, with these names:

In COBOL:	ADMUSC1
In FORTRAN:	ADMUSF1
In PL/I:	ADMUSP1.

- A program to display an alphanumeric panel. This program is provided in three languages, with these names:

In COBOL:	ADMUSC2
In FORTRAN:	ADMUSF2
In PL/I:	ADMUSP2.

Four other sample programs are written in PL/I. They are:

- ADMUSP3, which shows line types, colors, and patterns.
- ADMUSP4, which is a graphics editor program that allows pictures to be created. This sample program is designed to run on a 3270-PC/G or 3270-PC/GX work station; the pictures created by this program can be drawn on a plotter attached to one of these work stations.

  For more details of this sample program, see the *GDDM Application Programming Guide*.
- ADMUSP7, which is a program that performs the translation of chart objects between different country extended code pages. If the translation is successful, the chart objects are saved under the original chart name, replacing any previous versions.
- ADMUTMT (for MVS/TSO), and ADMUTMV (for VM/CMS), which is a sample task manager that demonstrates the use of GDDM's windowing functions.

The ADMUSC1, ADMUSF1, and ADMUSP1 sample programs

This program constructs a simple graph on a multicolored grid. The picture is displayed with an alphanumeric input field that requests the name of a printer (print file under VM/CMS). If a printer name is specified, the program generates a print data set comprising two copies of the graph, preceded by a header page. The program also saves the data stream on file.

IMS/VS version

The IMS/VS version of this sample program , has a slightly different interface. The printer LTERM name can be supplied on the transaction invocation. The program displays the picture and, if requested, copies it to a printer. The displayed picture contains an input alphanumeric field into which the next transaction code can be entered.

The source for the IMS/VS version is named ADMUSP1I on the GDDM distribution library. The main procedure name is still ADMUSP1.

The ADMUSC2, ADMUSF2, and ADMUSP2 sample programs

This program displays an alphanumeric panel requesting the name of a saved data-stream file. The file generated by the program ADMUSx1, where "x" is C, F, or P, (called "sample1") can be used; the original picture is then displayed again. After an interrupt, the original panel is redisplayed, awaiting new input. Pressing key PF3 or PF15 terminates the program.

IMS/VS version

The IMS/VS version of this sample program , has a slightly different interface. The name of the saved picture is supplied as a parameter to the transaction. A second, optional, parameter names the LTERM to which the picture is to be sent. If this name is omitted, the picture is sent to the terminal that entered the transaction. As well as the picture, the program generates a simple alphanumeric menu, which is sent to the originating terminal. This contains a field into which the next transaction can be entered.

The source for the IMS/VS version is named ADMUSP2I on the GDDM distribution library. The main procedure name is still ADMUSP2.

The ADMUSP3 sample program

This program uses GDDM to show:

- The 8 or 16 standard colors provided (depending on the display device)
- The 64 user colors in the supplied symbol set ADMCOLSD
- The 16 standard geometric shading patterns provided
- The 64 user geometric shading patterns in the supplied symbol set ADMPATTC
- The 8 standard line types provided
- The 2 standard line widths provided
- The 10 standard marker symbols provided
- A color-mixing table in mix mode.

Each of these is shown on a separate GDDM display page. Displays can be viewed sequentially in the order given above, or individually by selection from a menu panel listing the various options. At any stage, a printed copy can be obtained by following the instructions generated at the bottom of each display.

Source for this sample program is provided only in PL/I. This sample program cannot be run under IMS/VS.

The ADMUSP4 sample program

Information on compiling, link-editing, and running this sample program is given below. More information on it is given in the *GDDM Application Programming Guide*.

The ADMUSP7 sample program

This program displays a panel where the user gives the names and types of chart objects that are to be translated. The types of objects are:

Chart Data; enter 1
Chart Format; enter 2
Both; enter 3.

When the user presses the enter key, the translation starts. The program translates all character strings in the chart objects from the object code page to the application code page. If the translation is successful the program replaces the chart objects, overwriting previous versions saved under the same name. If any of the chart objects are not found, or if the chart type is invalid, the program issues an error message. If any other errors occur, the program does not save the translated chart, but it issues the appropriate GDDM message instead.

The ADMUTMT and ADMUTMV sample program

This program prompts the operator to select a program to run in a window. The sample program uses GDDM windowing calls and some sample Assembler routines, which are also supplied with GDDM, to perform the tasking functions. It supplies a running task manager under which you may run ADMUSP3 and ADMUSP4.

This program requires the following special procedures for compiling, link-editing, and running.

Compiling and link-editing under TSO

1. Assemble the assembler programs ADMUTMIT, ADMUTMTT, ADMUTMPT, ADMUTMAT, ADMUTMDT, ADMUTMST, and ADMUTMCT into an OBJ library.
2. Compile the PL/I program ADMUTMT into the same OBJ library.
3. Link-edit ADMUTMT into a LOAD library by using the following linkage editor control statements (which must start in column 2):

```
INCLUDE SYSLIB(ADMUTMT)
INCLUDE SYSLIB(ADMUTMIT)
INCLUDE SYSLIB(ADMUTMTT)
INCLUDE SYSLIB(ADMUTMPT)
INCLUDE SYSLIB(ADMUTMAT)
INCLUDE SYSLIB(ADMUTMDT)
INCLUDE SYSLIB(ADMUTMST)
INCLUDE SYSLIB(ADMUTMCT)
NAME ADMUTMT(R)
```

4. Compile and link-edit the sample PL/I programs, ADMUSP3 and ADMUSP4 into the same LOAD library as ADMUTMT.

Running under TSO

If the terminal does not have a PA3 key, create a GDDM defaults file, containing the CTLKEY procopt to assign a PF or PA key to invoke User Control. The following profile statement (which starts in column 2) assigns PF2 for this purpose:

```
ADMMNICK FAM=1,PROCOPT=((CTLKEY,1,2))
```

To run the sample task manager use the commands:

```
ALLOC F(ADMDEFS)  DA(GDDM-defaults-file-name) REUS SHR
ALLOC F(ADMSYMBL) DA(GDDM-symol-sets-file-name) REUS SHR
CALL load-library-name(ADMUTMT)
```

Compiling and link-editing under VM/CMS

Because the task manager program is written in PL/I, any other PL/I programs to be run under it must be link edited with PL/I on CMS before being loaded by the task manager, or else the load fails with duplicate PL/I main sections. Therefore this program requires special procedures for compiling, link-editing, and running. See below.

1. Build ADMUTMV TXTLIB by assembling the assembler programs ADMUTMIV, ADMUTMTV, ADMUTMPV, ADMUTMAV, ADMUTMDV, ADMUTMSV, and ADMUTMCV.
2. Compile the PL/I program ADMUTMV, but do not put it into the TXTLIB.
3. The sample programs ADMUSP3 and ADMUSP4 can be run from the sample task manager, but, as the task manager uses the GDDM reentrant interface, and ADMUSP3 and ADMUSP4 use the GDDM non-reentrant interface, these programs must be run from a LOADLIB. First compile these PL/I programs and then build the LOADLIB as follows:

```
FILEDEF SYSLIB DISK ADMNLIB TXTLIB  *
LKED ADMUSP3 ( LIBE ADMUTMV
FILEDEF SYSLIB DISK PLILIB  TXTLIB  *
FILEDEF INCLIB DISK ADMUTMV LOADLIB A
        ( DSORG PO RECFM U
LKED INCUSP3 ( LIBE ADMUTMV
```

   Where INCUSP3 TEXT contains these linkage editor control statements (which must start in column 2):

```
INCLUDE SYSLIB(DMSIBM)
INCLUDE INCLIB(ADMUSP3)
ENTRY DMSIBM
NAME ADMUSP3(R)
```

   This procedure link-edits ADMUSP3 and places the load module in ADMUTMV LOADLIB. The same procedure must be repeated for ADMUSP4.

Running under VM/CMS

If the terminal does not have a PA3 key, create a GDDM defaults file, PROFILE ADMDEFS, containing the CTLKEY processing option to assign a PF or PA key to invoke User Control. The following profile statement (which starts in column 2) assigns PF2 for this purpose:

```
ADMMNICK FAM=1,PROCOPT=((CTLKEY,1,2))
```

To run the sample task manager use the commands:

```
GLOBAL LOADLIB ADMUTMV
GLOBAL TXTLIB  ADMUTMV ADMRLIB ADMGLIB ADMPLIB
       PLILIB CMSLIB
LOAD ADMUTMV ( START
```

Using the sample task manager

The sample task manager displays a panel asking you to select which program to run from a menu of programs. Select one.

At any time when the selected program is waiting for input from you, you can instead call up User Control by pressing PA3 – or the alternative key as defined by the CTLKEY processing option. Make the task manager window active by using the NEXT function and ENDing the User Control session. You can now start to run another program from the menu. You can even run the same program again so that it appears in more than one window.

At any time a program is waiting for input from you, you can instead call up User Control to move or size the operator windows, or make a different operator window active.

To end the sample task manager, first end each program, then end the task manager.

You can change the sample task manager menu so that it runs your own programs. For details, see the prolog of ADMUTMT or ADMUTMV.

Compiling, link-editing, and running the sample programs

The programs should be compiled, link-edited, and run as follows.

Note: See also these chapters for more detail:

- Chapter 2, "Using GDDM under CICS/VS" on page 7
- Chapter 3, "Using GDDM under IMS/VS" on page 23
- Chapter 4, "Using GDDM under MVS/XA" on page 31
- Chapter 5, "Using GDDM under TSO" on page 33
- Chapter 6, "Using GDDM under VM/CMS" on page 41.

Compiling the programs

The source programs do not need to be modified except for:

1. Optional changes to the FSINIT call, as noted under "Link-editing the programs."
2. Replacing the STOP RUN statements in the COBOL programs if they are to run under CICS. The statements should be replaced with GO BACK or EXEC CICS RETURN.
3. Modifying ADMUSP3 if it is to be run on a device with less than 32 rows.

The programs must be compiled by a compiler appropriate to the source language and target subsystem (DOS/VS or OS/VS COBOL, FORTRAN G or H, or PL/I Optimizing Compiler). Note that CICS does not support programs written in FORTRAN.

Release 2 of the OS/VS COBOL compiler has a default option on the PARM parameter called QUOTE that causes a double quote (") to be used as the string delimiter. This is a change from OS/VS COBOL Release 1, and to compile the sample COBOL programs, the APOST option must be explicitly specified.

Also, the PARM options RESIDENT and DYNAMIC must be explicitly set to NORESIDENT and NODYNAMIC.

ADMUSP1, ADMUSP3, and ADMUSP4 use the supplied files of GDDM PL/I entry declarations. The members (containing PL/I declarations for nonreentrant base functions) must be available to the compiler in a source statement library under DOS/VSE, by means of SYSLIB specification under OS/VS, or by means of a GLOBAL MACLIB command under VM/CMS. The compilation of ADMUSP1, ADMUSP3, and ADMUSP4 must be performed with the MACRO option. No errors should result from the compilation steps.

ADMUSP3 is written to run on a device with at least 32 rows. However, because it is only the initial menu panel that requires more than the 24 rows available on an IBM 3278/3279 Model 2, the program can be run on this and other devices if the following change is made:

- Amend the initial value in the second column of "FIELD_DEF" to:

  `(2,3,4,6,8,10,12,14,16,18,20,24,1,22,5)`
- Amend the initial value in the second column of "PRINT_DEF" to:

  `(2,5,7,7,24)`

For information on the ADMUTMT/V sample programs, see "Compiling and link-editing under TSO" on page 250, and "Compiling and link-editing under VM/CMS" on page 250.

Link-editing the programs

Except under VM/CMS, the object code from the compilation must be link-edited with a GDDM interface routine appropriate to the subsystem and to the interface used (reentrant or nonreentrant).

Under OS/VS, the link-edit SYSLIBs must include the GDDM load library. The correct interface module is selected by an INCLUDE control statement specifying the appropriate member, as shown in Table 45 on page 252.

Or, the automatic-library-call facility can be used. For this, the source programs must be changed to replace the references to FSINIT with the appropriate alternative, as shown in Table 46 on page 252. (However, this is not necessary for ADMUTMT/V as they can only run on VM and TSO and they are coded with FSINR already.)

Note that for PL/I, the standard declarations do not include the alternative forms of FSINIT. They must, therefore, always be explicitly declared thus:

```
DCL FSINNC ENTRY EXTERNAL OPTIONS (ASM INTER);
```

Table 45. GDDM load library for link-edit SYSLIBs

Interface	Sample Programs	Required member for subsystem		
		CICS/OS/VS	IMS/VS	TSO
Nonreentrant	ADMUSC1 ADMUSF1 ADMUSF2 ADMUSP1 ADMUSP3 ADMUSP4	ADMASNC	ADMASNJ	ADMASNT
Reentrant	ADMUSC1 ADMUSP2	ADMASRC	ADMASRJ	ADMASRT

Table 46. GDDM automatic library calls

Interface	Sample Programs	Replace FSINIT references by the following for each subsystem		
		CICS/OS/VS	IMS/VS	TSO
Nonreentrant	ADMUSC1 ADMUSF1 ADMUSF2 ADMUSP1 ADMUSP3 ADMUSP4	FSINNC	FSINNPI	FSINN
Reentrant	ADMUSC1 ADMUSP2	FSINRC	FSINRPI	FSINR

Table 47. GDDM interface modules

Interface	Sample Programs	CICS/DOS/VS modules
Nonreentrant	ADMUSC1 ADMUSF1 ADMUSF2 ADMUSP1 ADMUSP3 ADMUSP4	ADMASNB and ADMASLC
Reentrant	ADMUSC1 ADMUSP2	ADMASRB and ADMASLC

Table 48. GDDM global TXTLIBs

Interface	Sample programs	Required VM/CMS library
Nonreentrant	ADMUSC1 ADMUSF1 ADMUSF2 ADMUSP1 ADMUSP3 ADMUSP4	ADMNLIB
Reentrant	ADMUSC1 ADMUSP2	ADMRLIB

Note: Plus,

- If DCSS is available.....no extra TXTLIBs
- If no DCSS is available.....ADMGLIB

Under DOS/VS, GDDM must be included from the relocatable libraries during link-editing.

The correct interface modules should be selected as shown in Table 47 on page 252 and should be included as described in Chapter 2, "Using GDDM under CICS/VS" on page 7. The automatic inclusion of the interface modules by source-program modification is not available under DOS/VS.

Under VM/CMS, there is no link-editing. However, the CMS GLOBAL TXTLIB command must be executed as described in Chapter 6, "Using GDDM under VM/CMS" on page 41 to identify TXTLIBs from which GDDM routines can be loaded. The TXTLIBs required depend on the sample program attributes and the presence of GDDM in a Discontiguous Shared Segment (DCSS). See Table 48 on page 252.

For information on the ADMUTMT/V sample programs, see "Compiling and link-editing under TSO" on page 250, and "Compiling and link-editing under VM/CMS" on page 250.

Running the programs

Note that the COBOL programs must not be run under CICS unless the STOP RUN statements have been replaced by a GO BACK statement or an EXEC CICS RETURN.

When the programs are run, the GDDM load (or core-image) library must be available. The same library is used for nonreentrant and reentrant programs. The first two sample programs make use of a file containing saved data streams. Except under VM/CMS, this file must be created before running the programs. The first program (ADMUSC1, ADMUSF1, ADMUSP1) also optionally generates a print file.

Under CICS, the programs must be added to the Program Control Table (PCT) and Processing Program Table (PPT). The GDDM load library (or core-image library) must be specified when CICS is started. For the saved data stream, the GDDM VSAM file (by default, ADMF) must have been created and entered in the File Control Table (FCT); this is part of the installation procedure.

Under IMS/VS, the programs must be added to the IMS/VS program library, and the transaction codes and ACB set up during IMS/VS system definition. Also, a data base must be assigned and initialized to contain the saved data stream. These actions are part of the installation procedure.

Under TSO, the GDDM load library should be available (for example, in a STEPLIB). It is also necessary to have created a partitioned data set to contain the saved data stream. As specified in Chapter 5, "Using GDDM under TSO" on page 33, this has a record length of 400. A suitable space allocation for the program is one directory block and 100 400-byte data blocks. The file ADMSAVE should be allocated to the data set before execution. If a print is requested, the print queue data set (ADMPRINT.REQUEST.QUEUE) must have been created and initialized.

Under VM/CMS, the GDDM TXTLIBs must be included in the GLOBAL libraries during execution, as described above, together with the language libraries. Program loading may be prolonged if a module is not generated. Files to contain saved data streams and print data are generated dynamically by GDDM, and must not be created or defined by FILEDEF.

To allow enough storage for GDDM, PL/I execution must specify ISASIZE; a value of 10K bytes is usually sufficient.

For information on the ADMUTMT/V sample programs, see "Running under TSO" on page 250, and "Running under VM/CMS" on page 250.

Appendix L. Format of a Composite Document Presentation Data Stream

This appendix describes the form of input accepted within GDDM by the Composite Document Print Utility; this utility controls the printing of a document containing graphics, image, and text. The physical organization of the file varies depending on the environment:

CICS	Temporary data file
VSE Batch	ESDS data set
MVS Batch	V-format sequential data set
TSO	V-format sequential data set
CMS	V-format sequential file.

In each case each record contains a complete structured field. The structured fields must be in the order shown, except where it is stated that the order is optional.

Structured fields are described in detail below.

Structured fields

A document consists of a sequence of structured fields, each of which has the following format:

0 – 1	Length of the structured field in bytes. This is the length of the parameters specific to the type of structured field, plus the 8-byte introducer. In no case may the length of a structured field be more than 8200. (This differs from AFPDS documents, for which the maximum is 8202.)
2 – 4	String identifying the type of structured field. The hexadecimal value for each type is shown in the heading for each structured field; see "Structured field formats" on page 256.
5 – 7	X'000000'.
8 – n	Parameter information as described for each structured field under "Structured field formats" on page 256.

Offsets, for example in error messages, are shown in hexadecimal and are calculated from the start of the structured field, including the two length bytes.

Document structure

In the syntax structure below, the following conventions apply:

::=	Precedes the definition of an item
[ ]	Square brackets indicate optional items
. . .	The item may be repeated.

```
document::=
        begin-document
        [invokable-master-environment-group] . . .
        [page] . . .
        end-document
```

The file cannot contain multiple documents. Anything after the end-document structured field is ignored.

The formats of individual structured fields, such as "begin-document" and "end-document", are defined in the next section, under the heading "Structured field formats" on page 256.

```
invokable-master-environment-group::=
        begin-master-environment-group
        [medium-descriptor]
        [medium-modification-control] . . . (up to two)
        [medium-copy-count]
        [map-medium-overlay]
        [page-descriptor]
        [page-position]
        end-master-environment-group

page::=
        [master-environment-group-invocation] . . .
        begin-page
        [active-environment-group]
        [presentation-text-object]
        [image-object] . . .
        [graphics-object] . . .
        end-page
```

Note: The presentation-text-object, image-object, and graphics-object may occur in any order.

```
master-environment-group-invocation::=
        begin-master-environment-group
        invoke-master-environment-group
        end-master-environment-group

active-environment-group::=
        begin-active-environment-group
        [map-coded-font]
        [page-descriptor]
        [page-position]
        [object-area-descriptor]
        [object-area-position]
        [presentation-text-descriptor]
        [object-area-position]
        end-active-environment-group

presentation-text-object::=
        begin-presentation-text
        [presentation-text-data] . . .
        end-presentation-text

graphics-object::=
        begin-graphics-object
        begin-object-environment-group
        object-area-descriptor
        object-area-position
        [map-coded-font]
        [graphics-data-descriptor]
        end-object-environment-group
        [graphics-data] . . .
        end-graphics-object

image-object::=
        begin-image-object
        begin-object-environment-group
        object-area-descriptor
        object-area-position
        [image-data-descriptor]
```

end-object-environment-group
[image-picture-data] . . .
end-image-object

In addition, **no-operation** structured fields may appear anywhere in the document and are ignored.

Structured field formats

Table 49. Structured field format cross reference

Hex code	Meaning
D3A66B	Object area descriptor (OBD)
D3A688	Medium descriptor (MDD)
D3A69B	Presentation text descriptor (PTD)
D3A6AF	Page descriptor (PGD)
D3A6BB	Graphics data descriptor (GDD)
D3A6FB	Image data descriptor (IDD)
D3A788	Medium modification control (MMC)
D3A89B	Begin presentation text (BPT)
D3A8A8	Begin document (BDT)
D3A8AF	Begin page (BPG)
D3A8BB	Begin graphics object (BGR)
D3A8C7	Begin object environment group (BOG)
D3A8C8	Begin master environment group (BMG)
D3A8C9	Begin active environment group (BAG)
D3A8FB	Begin image object (BIM)
D3A99B	End presentation text (EPT)
D3A9A8	End document (EDT)
D3A9AF	End page (EPG)
D3A9BB	End graphics object (EGR)
D3A9C7	End object environment group (EOG)
D3A9C8	End master environment group (EMG)
D3A9C9	End active environment group (EAG)
D3A9FB	End image object (EIM)
D3AB8A	Map coded font (MCF)
D3ABDF	Map medium overlay (MMO)
D3AC6B	Object area position (OBP)
D3AFC8	Invoke master environment group (IMG)
D3B188	Medium copy count (MCC)
D3B1AF	Page position (PGP)
D3EE9B	Presentation text data (PTX)
D3EEBB	Graphics data (GAD)
D3EEEE	No operation (NOP)
D3EEFB	Image picture data (IPD)

Begin active environment group (D3A8C9) BAG

Indicates the beginning of an active environment group.

0 – 7 Active environment group name (0 – 8 characters).

Begin document (D3A8A8) BDT

Indicates the beginning of the document. It contains the following fields:

0 – 7 Document name.

8 – 9 X'0000'.

10 – n Groups of optional, additional information, in any order. These groups are reserved to describe the level of function in the document.

Begin graphics object (D3A8BB) BGR

Indicates the beginning of a graphics object.

0 – 7 Data Object name (0 – 8 characters).

Begin image object (D3A8FB) BIM

Indicates the beginning of an image object.

0 – 7 Image name (0 – 8 characters).

Begin master environment group (D3A8C8) BMG

Indicates the beginning of a master environment group (MEG). This may be either an invokable MEG, or an invocation of such a MEG.

0 – 7 Master environment group name (0 – 8 characters).

Begin object environment group (D3A8C7) BOG

Indicates the beginning of an object environment group.

0 – 7 Object environment group name (0 – 8 characters).

Begin page (D3A8AF) BPG

Indicates the beginning of a page.

0 – 7 Page name (0 – 8 characters).

Begin presentation text (D3A89B) BPT

Indicates the beginning of a presentation text object.

0 – 7 Data object name (0 – 8 characters).

End active environment group (D3A9C9) EAG

Indicates the end of an active environment group.

0 – 7 Active environment group name (0 – 8 characters).

End document (D3A9A8) EDT

Indicates the end of the document.

0 – 7 Document name (0 – 8 characters).

End graphics object (D3A9BB) EGR

Indicates the end of a graphics object.

0 – 7 Data object name (0 – 8 characters).

End image object (D3A9FB) EIM

Indicates the end of an image object.

0 – 7 Image name (0 – 8 characters).

End master environment group (D3A9C8) EMG

Indicates the end of a master environment group.

0 – 7 Master environment group name (0 – 8 characters).

End object environment group (D3A9C7) EOG

Indicates the end of an object environment group.

0 – 7 Object environment group name (0 – 8 characters).

End page (D3A9AF) EPG

Indicates the end of a page.

0 – 7	Page name (0 – 8 characters).

End presentation text (D3A99B) EPT

Indicates the end of a presentation text object.

0 – 7	Data object name (0 – 8 characters).

Graphics data (D3EEBB) GAD

Contains the graphics orders to be drawn.

0 – n	Up to 8192 bytes of graphics data. This, combined with successive graphics data structured fields if required, contains one or more complete graphics segments. It must not have drawing orders outside segments.

The format is a sequence of orders suitable for processing by GSPUT, with the following exceptions:

- Segment start is one of two formats.
  - The longer of the two forms defined in Appendix D, "GDF order descriptions" on page 165.
  - An extended form of the above. The second byte contains X'0E' as the length of following data. The length of segment field contains the low-order 2 bytes of the length of segment. Two extra bytes at the end contain the high-order bytes.

  In each case the length of segment must be specified exactly, and is not assumed to end when a X'FF' order code is met.
- Segment end is not accepted as indicating the end of a segment, and is ignored. Instead, the length given in the segment start order is used to show the position of the end.

Coordinates must match the format specified in the graphics data descriptor.

A graphic order may span successive graphics data structured fields that make up an object.

Graphics data descriptor (D3A6BB) GDD

Specifies the limits of coordinates in the graphics data. It contains one or two groups of data as follows:

- Drawing order subset, optional.

0	X'F7'.
1	Length of following data.
2 – n	Reserved.

- Window specification, required.

0	X'F6'.
1	Length of following data.
2 – 3	X'0000'.
4	Format of coordinates: X'00' 2 byte integers. X'01' 4 byte floating-point.
5	X'00' Reserved.
6 – 11	(or 6 – 17 if floating-point coordinates) Reserved.
12 – 13	(or 18 – 21 if floating-point coordinates) x-coordinate of left edge in graphics data.
14 – 15	(or 22 – 25 if floating-point coordinates) x-coordinate of right edge in graphics data.
16 – 17	(or 26 – 29 if floating-point coordinates) y-coordinate of bottom edge in graphics data.
18 – 19	(or 30 – 33 if floating-point coordinates) y-coordinate of top edge in graphics data.
20 – 23	(or 34 – 41 if floating-point coordinates) Reserved.

The graphics data is drawn scaled to fit the object area specified in the object area descriptor and object area position fields. If the object area is partly off the page, the object is clipped at the page boundary.

Preservation of aspect ratio or size can be achieved by appropriate selection of parameters when creating this file.

Image data descriptor (D3A6FB) IDD

Specifies the size of the image to be included.

0	X'00'.
1 – 2	Number of pixels in 10 inches in the x-direction.
3 – 4	Number of pixels in 10 inches in the y-direction.
5 – 6	Image size in the x-direction.
7 – 8	Image size in the y-direction.

The image is drawn without scaling, and is trimmed to fit the object area specified in the object area descriptor and object area position fields. If the object area is partly off the page, the image is trimmed at the page boundary.

Invoke master environment group (D3AFC8) IMG

This indicates a change for the current state master environment group (MEG) parameters. It contains the following field:

0 – 7	The name of the invokable MEG whose parameters are to become the current state MEG values.

Image picture data (D3EEFB) IPD

Contains the image orders to be drawn.

0 – n	Up to 8192 bytes of image data. It must be in a format suitable for processing by IMAPT. The contents of a sequence of image picture data structured fields must be a valid sequence that would follow a call to IMAPTS specifying default compression and a format value of −2. This implies that the image data must follow the convention that **1 = black**.

Map coded font (D3AB8A) MCF

Identifies the correspondence between external font names and a resource local identifier. It consists of a repeating group for each font. Each has the following fields:

0 – 1	Length of this repeating group.
2 – n	Groups of additional information, in any order, as follows:

- Fully qualified name, required.

0 – 1	X'0C02'. Identifies the group.
2	Type of name as follows: X'84' Coded font name. X'85' Code page name. X'86' Font name.
3	X'00' Reserved.
4 – 11	External name of the font.

For a text map-coded font, either of the following is required:

- a coded font name
- both a code page name and a font name.

A graphics map-coded font must have only a font name, which is used as a symbol set name.

Fully qualified name for IPDS printers

GDDM uses the following interpretation of the fully qualified name when driving IPDS printers:

- Type of name X'84' Coded font name
- External name

0 – 1	A graphic character set global identifier (GCSGID).
2 – 3	A code page global identifier (CPGID).
4 – 5	A font global identifier (FGID).
6 – 7	A 2-byte character width field. This is the width of the space character in 1/1440 inch units.

- Resource local identifier, required.

0 – 2	X'042405'. Identifies the group.
3	Resource local identifier. It must be in the range 1 – 127 for a text font. When used in a graphics object the value is as follows: 0 — pattern or marker symbol set 65 through 223 — other symbol sets.

- Font descriptor, optional

0 – 1	X'0D1F'. Identifies the group.
2	Font weight class. It must be in the range 1 – 9.
3	Font width class. It must be in the range 1 – 9.
4 – 5	Font vertical point size. It must be in the range 0 – 360.
6 – 7	Average character width. It must be in the range 0 – 360.
8	Font descriptor flags, assigned as follows: X'80' Italic X'40' Underscored X'10' Outline characters X'08' Overstruck X'04' Proportional spaced.
9	Font usage. This is defined in a graphics object only, and is a reserved byte for a text font. The values 1 – 5 correspond to the type parameter on a call to GSLSS. Other values are reserved.
10	Font family. Reserved.
11	Font class. Reserved.
12	Font quality.

Map coded font for IPDS printers

GDDM uses the following interpretation of the map coded font structured field when driving IPDS printers:

1. Font weight class

X'07'	Bold characters (when supported by printer)
other values	Normal characters.

2. Font width class

X'07'	Double wide characters (when supported by printer)
other values	Normal characters.

3. Font descriptor flags

X'80'	Italic characters (when supported by printer)
X'08'	Overstruck characters (when supported by printer).

4. Font quality

X'01'	Low quality, high speed (when supported by printer)
X'02'	Medium quality, medium speed (when supported by printer)
X'03'	High quality, low speed (when supported by printer).

GDDM changes font quality only when it changes pages. For consistent results, all fonts used on any one page should all have the same font quality.

Map medium overlay (D3ABDF) MMO

Identifies the correspondence between an external overlay name and a resource local identifier. It contains one or two groups, each specifying such a pair. If a second group is included, it applies to the reverse of the paper in duplex printing. The format of each group is:

0 – 1	X'0012'.
2 – 17	Two groups of additional information, in any order, as follows:

- Fully qualified name, required.

0 – 3	X'0C028400'. Identifies the group.
4 – 11	External name of the overlay.

- Resource local identifier, required.

0 – 2	X'042401'. Identifies the group.
3	Resource local identifier. It must be in the range 1 – 127.

Medium copy count (D3B188) MCC

Specifies medium modification group references.

0 – 4	X'0001000100'.
5	Medium modification group reference for the front of the paper. It must match the group identifier in a medium modification control structured field.
6	Medium modification group reference for the reverse of the paper, applicable to duplex printing. It must match the group identifier in a medium modification control structured field, or is zero for simplex printing.

Medium descriptor (D3A688) MDD

Specifies the size of the medium. The default is the size of the target device known to GDDM. It contains the following fields:

0 – 1	X'0000'.
2 – 3	Number of medium measurement units in 10 inches in the x-direction. It must be in the range 2400 – 14400.
4 – 5	Number of medium measurement units in 10 inches in the y-direction. It must be in the range 2400 – 14400.
6 – 8	Medium size in the x-direction (in the range 1 – 8388607).
9 – 11	Medium size in the y-direction (in the range 1 – 8388607).

Medium modification control (D3A788) MMC

Specifies the modifications of a medium copy group.

0	Medium modification group identifier, in the range 1 – 127.
1	X'FF'.
2 – n	Modifications, in any order, from the list below. Each type of modification may be specified at most once.

- First source location selector.

0	X'E1'. Keyword identifier.
1	Source selection for the first form in the group. It must be in the range 1 – 3. The default is 1.

- Subsequent source location selector.

0	X'E2'. Keyword identifier.
1	Source selection for subsequent forms in the group. It must be in the range 1 – 3. The default is 1.

- Medium overlay local identifier.

0	X'F2'. Keyword identifier.
1	Local identifier for the overlay required. It must match the local identifier in a map medium overlay structured field. The default is no overlay.

No operation (D3EEEE) NOP

This may be used to add comments to the data stream. It can appear in any position.

0 – n	Up to 8192 bytes of comment data, not examined.

Object area descriptor (D3A66B) OBD

Specifies the size of an object. It contains the following fields:

0 – n	Three groups of additional information, in any order, as follows:

- Descriptor position identifier, required.

0 – 1	X'0343'. Identifies the group.
2	Descriptor position identifier in range 1 – 127.

- Object area measurement units, required.

0 – 3	X'084B0000'. Identifies the group.
4 – 5	Number of object area units in 10 inches in the x-direction. It must be in the range 2400 – 14400.
6 – 7	Number of object area units in 10 inches in the y-direction. It must be in the range 2400 – 14400.

- Object area size, required.

0 – 2	X'094C02'. Identifies the group.
3 – 5	Object area size in the x-direction (in the range 1 – 8388607).
6 – 8	Object area size in the y-direction (in the range 1 – 8388607).

The information for the text object area descriptor must match that of the page descriptor, which is the default if this field is missing.

Object area position (D3AC6B) OBP

Specifies the position of an object on the page.

0	Object area position identifier in range 1 – 127.
1	X'17'.
2 – 4	Object area origin, x (in the range 0 – 8388607).
5 – 7	Object area origin, y (in the range 0 – 8388607).
8 – 12	X'00002D0000'.
13 – 15	Object content origin, x (in the range 0 – 8388607).
16 – 18	Object content origin, y (in the range 0 – 8388607).
19 – 23	X'00002D0001'.

The text object area position, if present, must have both origins zero (the default).

Parts of a graphics or image object that lie outside the page size are not printed.

Page descriptor (D3A6AF) PGD

Specifies the size of the page, which must not be zero and must fit on the GDDM device size at the position given by the page-position structured field. The default is found from the size in the medium descriptor.

0 – 1	X'0000'.
2 – 3	Number of page measurement units in 10 inches in the x-direction. It must be in the range 2400 – 14400.
4 – 5	Number of page measurement units in 10 inches in the y-direction. It must be in the range 2400 – 14400.
6 – 8	Page size in the x-direction (in the range 1 – 8388607).
9 – 11	Page size in the y-direction (in the range 1 – 8388607).

Page position (D3B1AF) PGP

Specifies the position of the page on the form. The defaults are zero. It contains the following fields:

0 – 1	X'0109'.
2 – 4	Page origin in the x-direction (in the range 0 – 8388607).
5 – 7	Page origin in the y-direction (in the range 0 – 8388607).
8 – 9	Page orientation. The permitted values and the corresponding orientations are as follows: X'0000' North; X'2D00' East; X'5A00' South; X'8700' West.

Note: The page origin is ignored for family-4 devices if it is found within an active environment group.

Presentation text data (D3EE9B) PTX

This contains a chain of text controls. It contains the following fields:

0 – 1	X'2BD3'.
2 – n	A sequence of text controls as described below. Byte 1 of the last text control contains the value shown minus 1, to mark the end of the chain.

- Absolute move baseline.

  Sets the distance from the top margin of the paper.

0 – 1	X'04D3'.
2 – 3	Baseline print position.

- Absolute move inline.

  Sets the distance from the left margin of the paper.

0 – 1	X'04C7'.
2 – 3	Inline print position.

- Begin line.

  Sets the print position for a new line. See also **set baseline increment** and **set inline margin.**

0 – 1	X'02D9'.

- Draw baseline rule.

  Draws a rule perpendicular to the top of the page.

0 – 1	X'07E7'.
2 – 3	Length. If the length is positive, the line is drawn in the sequential baseline direction. If the length is negative, the line is drawn away from the sequential baseline direction.
4 – 5	Width. If the width is positive, pixels are added in the positive inline direction. If the width is negative, pixels are added in the negative inline direction.
6	X'00'.

- Draw inline rule.

  Draws a rule parallel to the top of the page.

0 – 1	X'07E5'.
2 – 3	Length. If the length is positive, the line is drawn in the sequential inline direction. If the length is negative, the line is drawn away from the sequential inline direction.
4 – 5	Width. If the width is positive, pixels are added in the positive baseline direction. If the width is negative, pixels are added in the negative baseline direction.
6	X'00'.

- No operation.

  Used to include variable length comments, or to terminate chaining (by changing byte 1 to X'F8').

0 – 1	X'xxF9'. xx denotes the length of the comment.
2 – (xx+1)	Variable length comment.

- Relative move baseline.

  Used to change the current baseline position.

0 – 1	X'04D5'.
2 – 3	Baseline print position move. It can be positive or negative.

- Relative move inline.

  Used to change the current inline position.

0 – 1	X'04C9'.
2 – 3	Inline print position move. It can be positive or negative.

- Repeat string.

  Specifies the repetition of a character string.

0 – 1	X'xxEF'. xx denotes the length of the string to be repeated.
2 – 3	Number of characters to be repeated. For example, if the resulting length is 7 and the string is **ABC**, then **ABCABCA** is printed.
4 – (xx+3)	Variable length string to be repeated.

  For example, to generate the string ABCABCA the field would be:

  ```
  03EF0007ABC
  ```

- Set baseline increment.

  Sets the baseline movement to be used by the begin line control.

0 – 1	X'04D1'.
2 – 3	Baseline increment amount.

- Set coded font local.

  Identifies the coded font to be used for printing subsequent text.

0 – 1	X'03F1'.
2	Local identifier of the coded font to be used. It must match one of those in a map coded font structured field.

- Set inline margin.

  Sets the inline margin to be used by the begin line control.

0 – 1	X'04C1'
2 – 3	Inline margin.

- Set intercharacter increment.

  Used to set intercharacter spacing.

0 – 1	X'04C3'
2 – 3	Increment.

- Set text color.

  Specifies the color of text that follows.

0	Length of set text color control, either 4 or 5.
1	X'75'.
2 – 3	Foreground color.
4	If present, indicates the color precision.

- Set text orientation.

  Only one text orientation may be specified, although the whole page may be rotated.

0 – 5	X'06F700002D00'.

- Set variable space character increment.

  Establish the width of blank characters that appear in transparent data controls.

0 – 1	X'04C5'.
2 – 3	Width of variable space character.

- Transparent data.

  Identifies text to be printed that does not contain any text controls.

0 – 1	X'xxDB'. xx denotes the length of the text.
2 – (xx+1)	Variable length text to be printed.

Presentation text descriptor (D3A69B) PTD

Gives the size of the text block on the page, and may specify defaults to be used for text controls. The fields are as follows:

0 – 1	X'0000'.
2 – 3	Number of text measurement units in 10 inches in the x-direction. It must be in the range 2400 – 14400.
4 – 5	Number of text measurement units in 10 inches in the y-direction. It must be in the range 2400 – 14400.
6 – 8	Text block size in the x-direction, the same as the size in the Page Descriptor.
9 – 11	Text block size in the y-direction, the same as the size in the Page Descriptor.
12 – 13	Presentation text flags.
14 – n	Optional groups of additional information, in any order, specifying initial defaults in place of printer defaults. The format is the same as bytes 2 – n of the presentation text data structured field. The subset that may be included in the presentation text descriptor structured field is as follows:

- Set baseline increment
- Set coded font local
- Set intercharacter increment
- Set inline margin
- Initial addressable position (absolute move baseline and absolute move inline)
- Set text color.

Index

GDDM
Version 2 Release 2
Base Programming Reference (Volume 2)

Order No. SC33-0332-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative, or the IBM branch office serving your locality.*

Number of latest Technical Newsletter for this publication...

Note: *Staples can cause problems with automated mail-sorting equipment.*
Please use pressure-sensitive or other gummed tape to seal this form.

If you want an acknowledgement, give your name and address below.

Name ....

Job Title .... Company ....

Address ....

.... Zip ....

For clarity, please type your comments or write them clearly, so that their meaning is fully understood.
Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the front of this book.)

SC33-0332-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H
180 Kost Road
Mechanicsburg, PA 17055, USA

Fold and tape | Please do not staple | Fold and tape

IBM®

SC33-0332-1
Version 2 Release 2

GDDM Base Programming Reference Volume 2 of 2 Printed in U.S.A. SC33-0332-1

SC33-0332-01