GEM®/3 Programmer's Toolkit™

Release 3.1 Supplement

DIGITAL RESEARCH®

GEM®/3 Programmer's Toolkit™

Release 3.1 Supplement

Foreword

This supplement updates the information contained in the documentation set of GEM® Programmer's Toolkit™. Recent changes to the toolkit software have both enhanced existing features and added new functionality.

The GEM 3.1 Programmer's Toolkit Supplement describes the new install library utility (INSTLIB), new function calls added to the GEM programming libraries, and updates to the GEM Applications Environment Services (AES) and GEM Virtual Device Interface (VDI).

In Chapters 1 and 2 of this supplement, there is information about how to use the new install library utility to install the sources of the new GEM bindings on your hard disk. You can choose from any of the following C language compilers and versions:

- Borland Turbo C® — 1.0 to 2.0
- Microsoft® C Compiler — 5.x
- MetaWare™ High C™ — 1.4 and 1.5

The installation utility also lets you select which libraries you want to install. How you choose to have the libraries built depends on optimization.

Chapters 2 through 8 deal with the new function calls added to the GEM Programming Libraries. The library descriptions are divided into these categories:

- Extended Object Library
- Extended Raster Library
- Transformation Library
- Miscellaneous Library
- DOS Library
- Expanded Memory System (EMS) Library

Chapters 9 through 12 contain the information required to bring the GEM Application Environment Services Reference Guide and the GEM Virtual Device Interface Reference Guide up to Release 3.1 of the GEM system software (GEM/3).

Contents

Section 1 Installation

Before you can install the new libraries, you must install the GEM system software on your hard disk according to the instructions listed the *GEM®/3 Desktop™ Installation Guide*. Once you have successfully installed the GEM Desktop, you will use the GEM Install application INSTALL.APP to install the Install Library application (INSTLIB.APP).

Follow these basic steps to generate GEM bindings for your compiler from sources.

1. Install the GEM system software.
2. From the GEM Desktop, run INSTALL.APP to install the Install Library application (INSLIB.APP). The Desktop will automatically create a folder (directory) called TOOLKIT off of the root directory of the drive you specify.
3. Start INSTLIB.APP.
4. Select the compiler you want to use.
5. Select how you want the library organized.
6. Select the libraries you want to use.

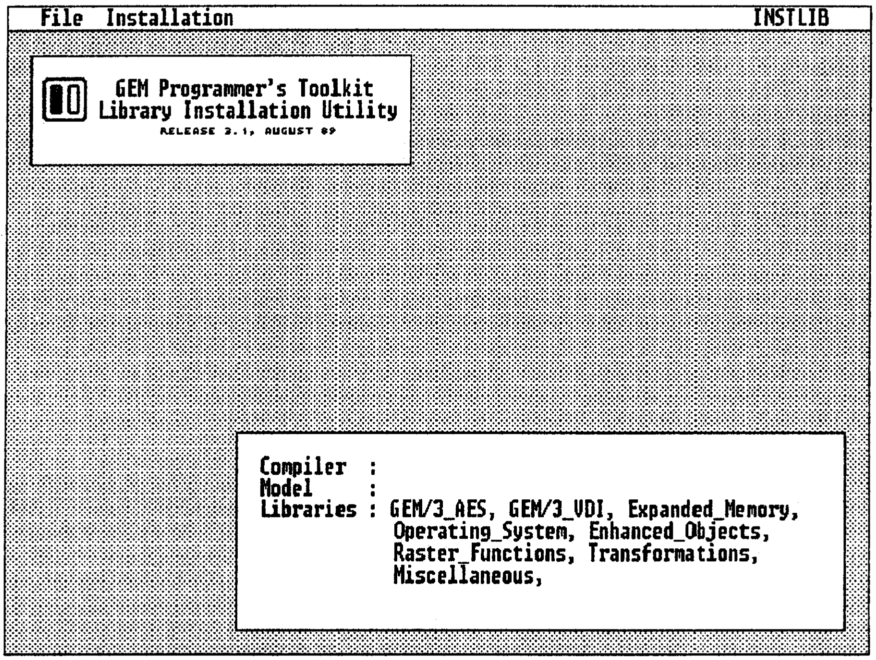
After the installation procedures are complete, you must edit one file (PORTAB.H) to ensure that the correct compiler is selected for the bindings.

Starting the Install Library Application (INSTLIB.APP)

Use INSTLIB.APP, the install library application, to install the sources of the GEM bindings on your hard disk for use with your favorite compiler.

1. Type **GEM** at the DOS prompt to load the GEM Desktop.
2. Change the directory to \TOOLKIT\BINDINGS on the drive where you have installed the Programmer's Toolkit.

3. Double-click on the INSTLIB.APP icon. You will see the following screen displayed:



GEM Programmer's Toolkit
Library Installation Utility
RELEASE 3.1, AUGUST 89

```
Compiler  :
Model     :
Libraries : GEM/3_AES, GEM/3_VDI, Expanded_Memory,
            Operating_System, Enhanced_Objects,
            Raster_Functions, Transformations,
            Miscellaneous,
```

Note the information box at the bottom of this screen. The first time you load INSTLIB.APP, all the libraries are installed, but the Compiler and Model fields are unassigned. The next time you load INSTLIB.APP, all three fields will be cleared so you can select new options if you have modified the bindings.

When you are ready to select your compiler type and library model, click on **Installation** in the menu bar to see the Installation Menu. There are four commands in the Installation Menu:

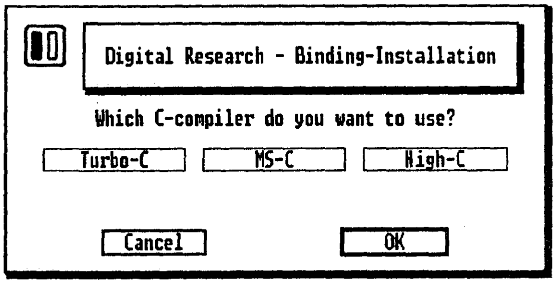
- Compiler
- Library model
- Libraries
- Install

If this is your first installation, you will notice that the **Libraries** option is grayed-out; this is because all the libraries will be installed for you automatically. The next time you load INSTLIB.APP, this option will be available. To continue with the installation process, click on the **Compiler** option in the Installation Menu.

Note: The dialog handling in INSTLIB.APP differs from standard GEM dialog handling. INSTLIB.APP has been built with the new FORM_EXDO call that comes with version 3.1 of the GEM Programmer's Toolkit. With this new dialog handling, you can press Ctrl-A, Ctrl-B, Ctrl-C and so on to select buttons in dialogs. You can press Shift-Enter instead of Enter to automatically select the default button, and press the End key to automatically cancel the current operation.

Selecting the Compiler

After you select **Compiler** from the Installation Menu, you will see this dialog.


Digital Research - Binding-Installation

Which C-compiler do you want to use?

To install the sources, select one of the three compilers by clicking on its button. Click on the compiler of your choice to highlight your selection.

Click on the OK button to continue; otherwise, click on Cancel to return to the INSTLIB Main Menu.

After you select your compiler, it will be listed in the information box at the Main Menu.

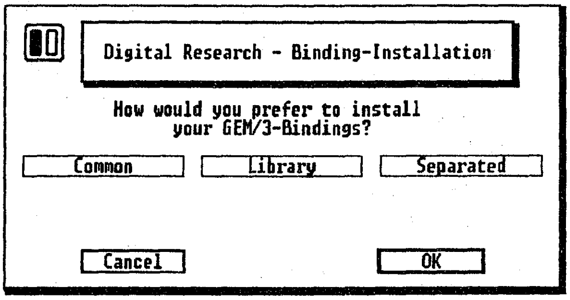
Selecting Library Organization

From INSTLIB's Main Menu, click on **Installation** in the menu bar. From the Installation Menu, select the **Library model** command; you will see this dialog.

Before selecting the library model, you should be aware that you will need at least 6 megabytes of free disk space if you select the Separated library organization. If you select Common or Library, you will need at least 4 megabytes of free disk space. The compile time will vary depending on your system configuration, but generally the compile time takes at least 1.5 hours. *Detailed descriptions of the binding options follow this section.*

From this dialog, select how you want your library organized. Click once on the binding installation of your choice. Double-click on any of the three library buttons to see a brief description of the binding options.

When you have finished, click on the OK button to return to the Main Menu; otherwise, click on the Cancel button. When you return to the Main Menu, the Model field in the information box will reflect your selection.


Digital Research - Binding-Installation

How would you prefer to install
your GEM/3-Bindings?

If this is your first installation, you are ready to select the **Install** option from the Installation Menu. When you click on Install, the installation process begins; INSTLIB provides on-screen messages indicating which files are being written to your hard disk. The sources will be copied onto your hard disk in \TOOLKIT\GEMLIB in the appropriate subdisrectories.

When the installation process is complete, you will see a message telling you to modify the PORTAB.H file. For more information, refer to "Editing PORTAB.H" later in this section.

COMMON

If you select COMMON, each collection of library routines are held in one source file; the entire GEM library will contain only seven files. This is how the earlier version of the toolkit handled bindings. The disadvantage of this method is that if you write a small application, all the AES and/or VDI functions are linked in with the application. This can cause applications to grow unnecessarily large because routines that are not called are still included.

The COMMON installation requires the least disk space and compile time. About 24 files will be created (including header and ancilliary files).

LIBRARY

If you select LIBRARY, the larger sources (AESBIND and VDIBIND) are split into several modules containing functions that are part of a specific category (SHEL_???, APPL_???, and so on). Use this option to save space if you are not using specific libraries. If you use APPL_INIT, APPL_WRITE or any other APPL_ calls, then all the routines in that category will be bound to the application regardless of whether they are called or not.

The LIBRARY installation is a compromise of COMMON and SEPARATE. It does not require a large amount of disk space and it significantly reduces compile time. Applications built using the libraries created with this option will not be fully size optimized. About 43 files will be created.

SEPARATED

If you select SEPARATED, all sources are split into several modules; each module contains one function. Only those functions that are used in the application source are bound to the application. This is how all standard libraries (like STDLIB of Microsoft C Compiler and Turbo C) are now built.

This kind of installation requires at least 6 Megabytes of disk space and about one and a half hours to compile the bindings. A SEPARATED installation provides the greatest saving in terms of smaller applications. About 319 files will be created.

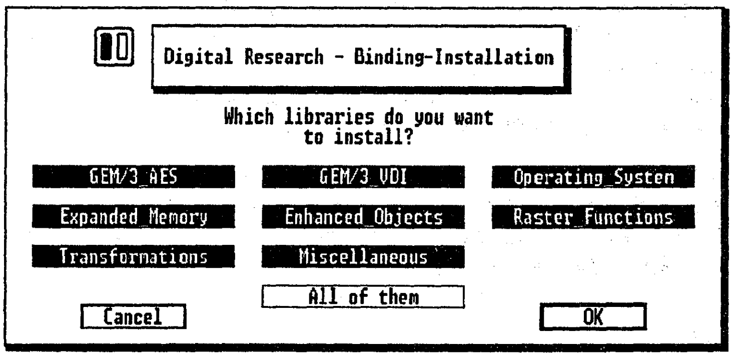
Note: If disk space is an issue and you do not intend to modify the source of the bindings in the future, you can delete all the library source files after you have built your GEM/3 libraries.

Selecting Libraries

If this is not your first installation, you have the option of installing a subset of all the GEM libraries. With this feature, you can easily modify portions of your the bindings. During the actual installation of the libraries, you can choose not to overwrite the existing sources and save yourself a substantial amount of time.

If you want to install all the GEM libraries or a subset, load INSTLIB.APP (as described in "Starting the Installation Application" earlier in this section).

From the INSTLIB Main Menu, select **Installation** from the menu bar. From the Installation Menu, click on the **Libraries** option. You will see this dialog.


Digital Research - Binding-Installation

Which libraries do you want to install?

From this dialog, select the libraries you want to install. Select any combination of libraries that you want. As you click on each button, it is highlighted. If you double-click on a button, you will see a brief description of that library.

When you have selected all the libraries that you want to install, click on the OK button; otherwise, click on Cancel.

You will return to the Main Menu. The Libraries field in the information box will be updated to reflect your current library selections.

Installing the Libraries

To start the installation process, click on **Install** in the Installation Menu. INSTLIB.APP provides complete information about what is happening as it installs the libraries.

Editing PORTAB.H

When the source installation is complete, you will see a message instructing you to edit PORTAB.H so the bindings agree with your compiler type. From INSTLIB's Main Menu, select **Quit** from the File Menu.

The file PORTAB.H is located in \TOOLKIT\INC. The identifier, which specifies the compiler in use, is located at the beginning of this file. Set #define for the selected compiler to 1 (one) and the other compiler's identifiers to 0 (zero). For example, if you selected Turbo C as your compiler, the identifier would look like this:

```
#define     TURBO_C     1     /*selected*/
#define     MS_C        0     /*not selected*/
#define     HIGH_C      0     /*not selected*/
```

Note: Remember to save PORTAB.H after you modify it.

Section 2
Compiler Notes

The following notes provide information about how to use the GEM/3 bindings with the supported compilers:

- Turbo C
- Microsoft C
- High C

Turbo C

Using the GEM/3 library with Turbo C is particularly well supported. To use the library, however, you should be aware of the following information.

TURBOC.CFG (TCC Configuration File)

You must create a file named TURBOC.CFG in the directory where you have installed Turbo C. This file holds options and directives which are used by the Turbo C compiler (TCC.EXE) when it is started.

To use the GEM/3 library, TURBOC.CFG must contain the following additional directives required for GEM/3 programming:

```
-ml or -ms
```

The -ml directive in TURBO.CFG tells TCC.EXE to use the large memory model; -ms indicates the small memory model. It is recommended, that you use the large model if you are not fully familiar with the procedure for writing GEM/3 applications or with operation of the GEM/3 Toolkit.

```
-I?:\TOOLKIT\INC
```

Replace the question mark (?) in the preceding example with the drive letter that contains the GEM/3 bindings. This directive defines the include path that TCC.EXE uses to search for header files. You can specify this directive as many times as necessary. You should also specify the Turbo C include directory in TURBOC.CFG (for example -I?:\TC\INCLUDE).

Note: Be sure that PORTAB.H is not located in the Turbo C include directory. If it is, you must either rename it or place the GEM/3 include directory in front of the Turbo C include directory. This ensures that the PORTAB.H file that comes with your GEM/3 Programmer's Toolkit is the one you use for any GEM/3 development.

```
-L?:\TOOLKIT\GEMLIB
```

Replace the question mark (?) in the preceding example with the drive letter that contains the GEM/3 bindings. This directive defines the path that TCC.EXE uses to search for the GEM libraries. If you copy the GEM/3 library you built to the Turbo C library directory (?:\TC\LIB), then you will not need to specify this line. You must specify the Turbo C library directory, regardless of whether you specified the GEM library directory or not.

Note: Do not modify other lines in TURBOC.CFG if you are not familiar with their meanings.

Examples

This example lists the contents of C:\TC\TURBOC.CFG for building a large memory model library on Drive D with Turbo C installed on Drive C.

```
-ml
-IC:\TC\INCLUDE
-ID:\TOOLKIT\INC
-LC:\TC\LIB
-LD:\TOOLKIT\GEMLIB
```

BUILTINS.MAK (MAKE Definition File)

To use Turbo C's MAKE utility, you must create a file to hold the standard rules MAKE uses for compiling and assembling your source files. The file BUILTINS.MAK should be located at the same directory where your MAKE.EXE is located and should contain the following:

```
.c.obj:
        tcc -c $*
.asm.obj:
        tasm $* /MX;
```

Note: Be sure that the spaces before "tcc" and "tasm" are created by pressing the tab key, not the spacebar.

If you use Microsoft's Macro Assembler (MASM) as your assembler, replace the Turbo Assembler®'s TASM command with the MASM command:

Instead of:

```
        masm $* /MX;
```

Write:

```
        tasm $* /MX;
```

You must also modify the makefile for the Miscellaneous Library. Modify the line that calls the assembler for building FARDRAW.OBJ, including all text between the second and third slashes (/d__SMALL__) as follows.

```
tasm /dTC /d__SMALL__/ mx fardraw.asm fardraw.obj
```

Microsoft C

Using Microsoft C with the GEM/3 library is as straightforward as using Turbo C. Follow the steps below so that Microsoft C can find all the required paths.

TOOLS.INI (Microsoft C Initialization File)

All Microsoft C utilities can read a common configuration file. This file is named TOOLS.INI. The path to this file must be specified in the environment variable INIT. For example, if the TOOLS.INI file is in E:\MSC\BIN, the variable INIT should be set as:

```
SET INIT=E:\MSC\BIN
```

You can put this command in a batch file, such as AUTOEXEC.BAT. The TOOLS.INI file is created automatically when you invoke the toolkit installation utility INSTLIB.APP in \TOOLKIT\BINDINGS. The environment variable INIT is set automatically when you start MAKELIB.BAT.

If you have your own TOOLS.INI file that you want to use, remove TOOLS.INI from the GEMLIB directory and delete the following line from MAKELIB.BAT:

```
SET INIT=\TOOLKIT\GEMLIB
```

Examples

You must edit TOOLS.INI to set the compiler switch option to correspond to the memory model you are building. The first example sets the compiler switch option in TOOLS.INI for the small memory model.

Edit TOOLS.INI for the small memory model like this:

```
[MAKE]
.c.obj:
      cl -c -AS -Gs -Oas -Zl -I\TOOLKIT\INC $*.c
.asm.obj:
      masm $* /DMSC /MX;
```

In this example, edit TOOLS.INI for the large memory model:

```
[MAKE]
.c.obj:
        cl -c -AL -Gs -Oas -Zl -I\TOOLKIT\INC $*.c
.asm.obj:
        masm $* /DMSC /MX;
```

INCLUDE and LIB (Microsoft C Environment Variables)

You must also set the environment variables LIB and INCLUDE as follows so Microsoft C can find the required header files and libraries:

```
SET INCLUDE = ?:\TOOLKIT\INC;?:\MSC\INCLUDE
```

and

```
SET LIB = ?:\MSC\LIB;?:\TOOLKIT\GEMLIB
```

Replace the question mark (?) in the preceding examples with the drive letter that contains the header files and librariess.

Setting environment variables can cause the operating system error, "Out of environment space." If this occurs, type the following line in your CONFIG.SYS file:

```
SHELL=COMMAND.COM/e:512/p
```

Save CONFIG.SYS and reboot your system. The SHELL command with these parameters specified will increase your environment space, allowing you to set all the needed environment variables for Microsoft C.

You must also modify the makefile for the Miscellaneous Library. Modify the line that calls the assembler for building FARDRAW.OBJ, including all text between the second and third slashes (/d__SMALL__) as follows.

```
masm /dMSC /d__SMALL__ /mx fardraw.asm fardraw.obj
```

MetaWare High C

There are a few restrictions when using this GEM/3 Programmer's Toolkit with the MetaWare High C compiler that are described at the end of this section. To use the GEM/3 bindings with MetaWare High C, you should be aware of the following.

MetaWare MAKE

Because MetaWare does not have its own MAKE utility, you will have to obtain one from another vendor. The makefiles for High C supplied with these GEM/3 bindings are configured for the NDMAKE utility. NDMAKE can be obtained from these sources:

US MAIL:	D. G. Kneller 1032 Irving Street 439 San Francisco, CA 94122
UUCP:	...ucbvax!ucsfcgl!kneller
ARPANET:	kneller@cgl.ucsf.edu
BITNET:	kneller@ucsfgl.BITNET

You can use any other MAKE utility (for example Borland's or Microsoft's), but you will have to edit the makefiles. Refer to the makefiles for GEM/3 bindings, Turbo C, or Microsoft C as an example.

Before you run MAKELIB.BAT, you must set up the High C compiler. To configure the High C compiler, start CONFIG.EXE located in the High C directory.

Specify the following settings (if necessary):

```
C     Memory_model          'BIG'
D     Tpages                150
M     Ipath                 '?:\toolkit\inc\'
J     Angle-include path    '?:\HC\INCLUDE\'
```

Note: The 'BIG' memory model in High C corresponds to the 'LARGE' memory model in Turbo C and Microsoft C.

All the paths specified in the configuration must end with a backslash (\); otherwise High C will not find the paths. The question marks (?) should be replaced with the drive letter on which the specified directories are located.

Runtime Startup for High C (INIT.OBJ)

When you install your High C compiler, the runtime startup code is built for the MetaWare High C heap manager. Because GEM/3 also needs memory for resources, you must assemble the INIT.ASM file located in the directory \HC\LIB\SRC.

Before assembling, edit INIT.ASM to define the following macros. Both statements appear as comments in the assembler file, so be sure to remove the semicolon at the beginning of each line.

```
;USE_DOS_ALLOC = 1
```

This tells High C to use the DOS alloc-functions when enlarging the heap.

```
;STACK_SIZE = XXXX
```

XXXX is the number of bytes you want to have available for the stack. 2000 is a typical stack size for most applications. You must specify the size because the stack is not automatically aligned when High C uses the DOS functions.

CALLINT Function of MetaWare High C (CALLINT.OBJ)

Because Metaware High C allows functions to pop their own parameters, you will also have to modify CALLINT.ASM located in \HC\LIB\SRC. At the end of this source, you will find that callint() returns and pops two bytes off the stack ("return 2"). Because the functions VDI() and GEM() also pop their passed word, edit CALLINT.ASM and delete the digit 2 from the callint statement.

Note: If you do not make this deletion, your system will crash!

Memory Model Constraints

MetaWare High C parses differently than Microsoft C or Turbo C by parsing the keyword 'far'. This makes it difficult to use High C to create GEM/3 bindings for the small memory model without editing the binding sources.

To avoid this restriction, modify all definitions with the keyword 'far' as described in the High C manual.

Creating MetaWare High C bindings for the small memory model can be difficult. This is due to the runtime start-up module for High C (INIT.ASM). INIT.ASM does not use DOS memory functions for heap management in the small memory model. The bindings are created using the small memory model. You can also bind an application. Because INIT.ASM does not free any memory after loading the program, GEM is unable to obtain memory from DOS for resources or variable data.

Compiling the Bindings

After you have modified PORTAB.H, change the current directory to \TOOLKIT\GEMLIB and type the following command:

`MAKELIB L`	To build bindings for the large memory model
`MAKELIB S`	To build bindings for the small memory model

This command starts MAKELIB.BAT compiles all the binding sources and creates the GEM library. Specifying the **L** or **S** parameter with the MAKELIB command changes only the name of the library (LTCGEM.LIB to STCGEM.LIB).

To ensure that your compiler is building the specified memory model, modify either TURBOC.CFG (changing -ml to -ms) or TOOLS.INI (changing -AL to -AS). For examples, refer to "TURBOC.CFG" or "TOOLS.INI" earlier in this section for more information.

Section 3
Extended Object Library

This section describes changes and additions to the Extended Object Library provided with this revised GEM/3 Programmers Toolkit. The Extended Object Library contains utility functions for the manipulation of object structures.

The descriptions assume a knowledge of GEM library call structures and parameter conventions. For further details of these and other GEM system calls, refer to the *GEM Application Environment Services Reference Guide* .

OB_DOSTATE

This function sets the specific state (SELECTED, HIDETREE, DISABLED, and so on) in the word ob_state of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`state`	state to set in ob_state

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_dostate();

OBJECT FAR *tree;

WORD index, state;

ob_dostate(tree, index, state);
```

OB_UNDOSTATE

This function clears the specific state (SELECTED, HIDETREE, DISABLED, and so on) in the word ob_state of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`state`	state to clear in ob_state

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_undostate();
OBJECT FAR *tree;
WORD index, state;
ob_undostate(tree, index, state);
```

OB_ISSTATE

This function gets the state (SELECTED, HIDETREE, DISABLED, and so on) in the word ob_state of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`state`	states to be tested

Output Arguments

`ret`	TRUE if state is set
	FALSE if state is not set

Sample Call to C Language Binding

```
WORD ob_isstate();
OBJECT FAR *tree;
WORD index, state, ret;
ret = ob_isstate(tree, index,state);
```

OB_DOFLAG

This function sets the specific flag (SELECTABLE, EXIT, TOUCHEXIT, and so on) in the word ob_flag of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`flag`	flag to set in ob_flag

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_doflag();

OBJECT FAR *tree;

WORD index, flag;

ob_doflag(tree, index, flag);
```

OB_UNDOFLAG

This function clears the specific flag (SELECTABLE, EXIT, TOUCHEXIT, and so on) in the word ob_flag of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`flag`	flag to set in ob_flag

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_undoflag();

OBJECT FAR *tree;

WORD index, flag;

ob_undoflag(tree, index, flag);
```

OB_ISFLAG

This function gets the flag (SELECTABLE, EXIT, TOUCHEXIT, and so on) in the word ob_flag of an object.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`flag`	flags to be tested

Output Arguments

`ret`	TRUE if flag is set
	FALSE if flag is not set

Sample Call to C Language Binding

```
WORD ob_isflag();
OBJECT FAR *tree;
WORD index, flag;
ret = ob_isflag(tree, index,flag);
```

OB_XYWH

This function returns the x,y,w,h rectangle of a given object. The function takes a pointer to a structure of type GRECT; on return, this contains the object's current x,y,w,h parameters.

Input Arguments

`tree`	object tree that contains specified object
`index`	index of object within tree
`prect`	far-pointer to a GRECT structure

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_xywh();

OBJECT FAR *tree;

WORD index;

GRECT FAR *prect;

ob_xywh(tree, index, prect);
```

OB_GET_TEXT

This functions returns a far pointer to the string pointed to by an object structure within a tree. The function uses the object type when returning the correct pointer.

Note: Objects that contain text pointers are G_TEXT, G_FTEXT, G_BOXTEXT, G_FBOXTEXT, G_STRING, G_BUTTON and G_TITLE.

The `clear` parameter requests the function to initially clear the string; TRUE clears and FALSE leaves the string unchanged.

Input Arguments

`tree`	object tree that contains specified object
`index`	index of object within tree
`clear`	initially clear string?

Output Arguments

`ptr`	far-pointer that points to the (cleared) string

Sample Call to C Language Binding

```
BYTE FAR *ob_get_text();
OBJECT FAR *tree;
WORD index, clear;
BYTE FAR *ptr;
ptr = ob_get_text(tree, index, clear);
```

OB_SET_TEXT

This function sets the text pointer of an required object to the `ptr` parameter. The object type is checked by the function before assigning the pointer.

Note: Objects that contain text pointers are G_TEXT, G_FTEXT, G_BOX-TEXT, G_FBOXTEXT, G_STRING, G_BUTTON and G_TITLE.

Input Arguments

`tree`	object tree that contains the specified object
`index`	index of object within tree
`ptr`	far-pointer to a string

Output Arguments

none

Sample Call to C Language Binding

```
VOID ob_set_text();

OBJECT FAR *tree;

WORD index;

BYTE FAR *ptr;

ob_set_text(tree, index, ptr);
```

OB_DRAW_DIALOG

This function draws an entire dialog with an optional growing box. The function takes an x,y,w,h rectangle which specifies the smallest start box of the growing rectangles. If the rectangle coordinates are all zero, the growing boxes will not be drawn.

Input Arguments

`tree`	object tree to be drawn
`x, y, w, h`	start growing rectangle

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID ob_draw_dialog();

OBJECT FAR *tree;

WORD x, y, w, h;

ob_draw_dialog(tree, x, y, w, h);
```

OB_UNDRAW_DIALOG

This function removes a previously drawn dialog box from the screen and draws an optional shrink box. The function takes an x,y,w,h rectangle which specifies the smallest start box of the shrinking rectangles. If the rectangle coordinates are all zero, the shrinking boxes will not be drawn.

Input Arguments

`tree`	object tree to be drawn
`x, y, w, h`	end shrinking rectangle

Output Arguments

```
none
```

Sample Call to C Language Binding

```
VOID ob_undraw_dialog();

OBJECT FAR *tree;

WORD x, y, w, h;

ob_undraw_dialog(tree, x, y, w, h);
```

Section 4
Extended Raster Library

This section describes modifications and enhancements to the Extended Raster Library provided with this revised GEM/3 Programmer's Toolkit. The Extended Raster library contains utility functions to manipulate coordinate structures (GRECT).

The descriptions assume a knowledge of GEM library call structures and parameter conventions. For more information about these and other GEM system calls, refer to the GEM Applications Environment Services Reference Guide.

RC_EQUAL

This function compares two rectangles to see if they are equal or not. Two pointers to structures of type GRECT are passed as parameters to this function.

Input Arguments

`prec1, prec2`	pointers to structures of type GRECT

Output Arguments

`ret`	TRUE if the rectangles are equal
	FALSE if the rectangles are not equal

Sample Call to C Language Binding

```
WORD rc_equal();

GRECT FAR *prec1, FAR *prec2;

ret = rc_equal(prec1, prec2);
```

RC_COPY

This functions copies the x,y,w,h coordinates of **psbox** to **pdbox**. The parameters are both structures of type GRECT. This function is nothing more than a structure copy.

Input Arguments

psbox, pdbox	pointer to source and destination GRECT

Output Arguments

none

Sample Call to C Language Binding

```
VOID rc_copy();
GRECT FAR *psbox, FAR *pdbox;
rc_copy(psbox, pdbox);
```

RC_INTERSECT

This function computes the intersection of two rectangles. The intersection is the area that is common to both rectangles.

The function returns TRUE, if there is a common area, and FALSE if there is not. If a common area exists, its coordinates are returned in the GRECT structure `p2`.

Input Arguments

`p1`	coordinates of the first rectangle
`p2`	coordinates of the second rectangle

Output Arguments

`ret`	TRUE if there is an intersection
	FALSE if there is no intersection

Sample Call to C Language Binding

```
WORD rc_intersect();

GRECT FAR *p1, FAR *p2;

WORD ret;

ret = rc_intersect(p1, p2);
```

RC_INSIDE

This function determines, whether a given x,y position is inside the given rectangle. If x,y is inside **prec**, the function returns TRUE. If not, FALSE is returned.

Input Arguments

x,y	position to check
prec	pointer to rectangle coordinates

Output Arguments

ret	TRUE if position is inside FALSE if position is not inside

Sample Call to C Language Binding

```
WORD rc_inside();

GRECT FAR *prec;

WORD ret;

ret = rc_inside(x, y, prec);
```

RC_GRECT_TO_ARRAY

This function transforms the supplied absolute coordinates of an x^1y^1, x^2y^2 array into a x,y,w,h rectangle form. The x^1y^1, x^2y^2 array contains the upper-left corner and the lower-right corner of a rectangle.

On return from the function, **prec** will hold the upper-left corner and the width and height dimensions.

Input Arguments

xy	x^1y^1, x^2y^2 array
prec	pointer to a structure of type GRECT

Output Arguments

none

Sample Call to C Language Binding

```
VOID rc_grect_to_array();

WORD FAR *xy;

GRECT FAR *prec;

rc_grect_to_array(prec, xy);
```

Section 5
Transformation Library

This section describes enhancements and modifications to the Transformation Library provided with this revised GEM/3 Programmer's Toolkit. The Transformation Library contains utility calls for manipulating coordinate systems.

The descriptions assume a knowledge of the GEM library call structures and parameter conventions. For further details of these and other GEM system calls, refer to the *GEM Application Environment Services Reference Guide.*

X_SXFORM

This function initializes the transformation library. A call to this function is required before using the transformation calls, otherwise unpredictable values will be returned.

The transformation library can be used for calculating the differences between device dependent coordinate and user coordinate systems. This call to x_sxform sets up both coordinate systems.

Input Arguments

`user_x, user_y, user_w, user_h`
: x,y,w,h rectangle which defines user coordinate space

`dev_x, dev_y, dev_w, dev_h`
: x,y,w,h rectangle which defines device coordinate space

`w_microns, h_microns`
: width and height of a pixel in microns (found in work_out[3] and work_out[4]). For more information, refer to V_OPNWK in the *GEM Virtual Device Interface Reference Guide.*

Output Arguments

`ret`	TRUE if initialization is successful FALSE if initialization fails

Sample Call to C Language Binding

```
WORD x_sxform();

WORD user_x, user_y, user_w, user_h;

WORD dev_x, dev_y, dev_w, dev_h;

WORD w_microns, h_microns;

x_sxform(user_x, user,_y, user_w, user_h, dev_x, dev_y,
dev_w, dev_h, w_microns, h_microns);
```

X_SASPECT

This function matches an aspect ratio on the device with one specified in user units. The match is done in physical units rather than pixels, so a square specified in user units will look square when displayed on the device. Calculating the aspect ratio match in this manner compensates for devices which have non-square pixels.

Input Arguments

`user_w, user_h`	width and height in user coordinates

Output Arguments

`dev_w, dev_h`	width and height in device coordinates

Sample Call to C Language Binding

```
WORD x_saspect();

WORD user_w, user_h;

WORD *dev_w, *dev_h;

x_saspect(user_w, user_h, dev_w, dev_h);
```

X_YTOX

This function returns the number of pixels in the x direction physically equal to "y" number of pixels in the y direction.

Input Arguments

y	number of pixels in the y direction

Output Arguments

x	number of pixels in the x direction

Sample Call to C Language Binding

```
WORD x_ytox();

WORD y, x;

x = x_ytox(y);
```

X_UDX_XFORM

This function transforms an x value from user space into device space.

Input Arguments

`user_x`	x coordinate of user raster

Output Arguments

`dev_x`	x coordinate of device raster

Sample Call to C Language Binding

```
WORD x_udx_xform();
WORD user_x, dev_x;
dev_x = x_udx_xform(user_x);
```

X_UDY_XFORM

This function transforms a y value from user space into device space.

Input Arguments

`user_y`	y coordinate of user raster

Output Arguments

`dev_y`	y coordinate of device raster

Sample Call to C Language Binding

```
WORD x_udy_xform();
WORD user_y, dev_y;
dev_y = x_udy_xform(user_y);
```

X_DUX_XFORM

This function transforms an x value from device space into user space.

Input Arguments

`dev_x`	x coordinate of device raster

Output Arguments

`user_x`	x coordinate of user raster

Sample Call to C Language Binding

```
WORD x_dux_xform();

WORD dev_x, user_x;

user_x = x_dux_xform(dev_x);
```

X_DUY_XFORM

This function transforms a y value from device space into user space.

<u>Input Arguments</u>

`dev_y`	y coordinate of device raster

<u>Output Arguments</u>

`user_y`	y coordinate of user raster

<u>Sample Call to C Language Binding</u>

```
WORD x_duy_xform();

WORD dev_y, user_y;

user_y = x_duy_xform(dev_y);
```

X_UDX_SCALE

This function scales an x distance from user space into device space.

Input Arguments

`user_dx`	x distance in user space

Output Arguments

`dev_dx`	x distance in device space

Sample Call to C Language Binding

```
WORD x_udx_scale();
WORD user_dx, dev_dx;
dev_dx = x_udx_scale(user_dx);
```

X_UDY_SCALE

This function scales a y distance from user space into device space.

Input Arguments

`user_dy`	y distance in user space

Output Arguments

`dev_dy`	y distance in device space

Sample Call to C Language Binding

```
WORD x_udy_scale();
WORD user_dy, dev_dy;
dev_dy = x_udy_scale(user_dy);
```

X_DUX_SCALE

This function scales an x distance from device space into user space.

Input Arguments

`dev_dx`	x distance in device space

Output Arguments

`user_dx`	x distance in user space

Sample Call to C Language Binding

```
WORD x_dux_scale();
WORD dev_dx, user_dx;
user_dx = x_dux_scale(dev_dx);
```

X_DUY_SCALE

This function scales a y distance from device space into user space.

Input Arguments

`dev_dy`	y distance in device space

Output Arguments

`user_dy`	y distance in user space

Sample Call to C Language Binding

```
WORD x_duy_scale();
WORD dev_dy, user_dy;
user_dy = x_duy_scale(dev_dy);
```

X_MUL_DIV

This function allows you to get floating point accuracy without going to the performance expense of floating point.

The calculation performed is: (((m1 * 2 * m2) / d1) + 1) / 2

Input Arguments

m1, m2	multiplicators
d1	divisor

Output Arguments

result	result of the above calculation

Sample Call to C Language Binding

```
WORD x_mul_div();
WORD m1, m2, d1, result;
result = x_mul_div(m1, m2, d1);
```

Section 6
Miscellaneous Library

This section describes modifications and enhancements to the Miscellaneous Library provided with the revised GEM/3 Programmer's Toolkit.

These descriptions assume a knowledge of the GEM library call structures and parameter conventions. For more information about these and other GEM system calls, refer to the *GEM Application Environment Services Reference Guide.*

FARDR_START

This function allows a C programmer to use user-defined objects. For more detailed information about user-defined objects, refer to *GEM Applications Environment Services Reference Guide.*

When using Digital Research® products X/GEM™ on FlexOS™ (and in some Atari® environments) the pointer to the PARMBLK structure is passed to the drawing code on the stack. This allows access to it as a parameter from C. GEM on DOS does not handle this similarly. Instead of passing the pointer on the stack, GEM passes it in the register pair AX:BX.

The fardr_start() function lets you use this pointer in a C program. You should call this function as the first action in your drawing code. The function will return the pointer to the PARMBLK structure so that you can assign its value to a local variable. The function also saves all AES registers, and sets the segment registers to the application's data segment.

You must call fardr_end() before you leave your drawing code.

Input Arguments

`none`

Output Arguments

`pb`	pointer to PARMBLK-structure

Sample Call to C Language Binding

```
PARMBLK FAR *fardr_start();

PARMBLK FAR *pb;

pb = fardr_start();
```

FARDR_END

This functions restores the registers and segments previously saved by fardr_start(), so that the AES finds the correct environment.

This function directly returns control to the AES. All statements after the call to fardr_end() will not be executed.

Input Arguments

`ret`	return code which you want to give to the AES

Output Arguments

none

Sample Call to C Language Binding

```
VOID fardr_end();

WORD ret;

fardr_end(ret);
```

FARDR_CS

This function returns the code segment value of your running application. You will need this function when you are building small memory model applications with Microsoft C. Microsoft C does not allow you to cast a code segment pointer from small to large memory model. If you want your source portable between different compilers and memory models, assign your drawing routine like this:

```
#if MS_C && M_I86SM
        applblk.ab_code = MKFP(fardr_cs(), drawing_routine);
#else
        applblk.ab_code = (WORD (FAR *)())drawing_routine;
#endif
```

You will find an example of assigning your drawing rountine in this form in USERDEF.C.

NOTE: This function is available only in the Microsoft C GEM library.

Input Arguments

`none`

Output Arguments

`cs_value`	current code segment

Sample Call to C Language Binding

```
UWORD fardr_cs();

UWORD cs_value;

cs_value fardr_cs();
```

FORM_EXDO

This is nearly the original source of the AES function form_do(). It has been improved to allow the selection of objects by control keys.

The function inspects the extended ob_type of an object. This should have been previously set with an ASCII value which corresponds to the control key sequence required to activate the object.

A value of one means Ctrl-A, two means Ctrl-B, and so on. The remaining functionality is unchanged. For more information, refer to the source code of form_exdo and FDTEST.APP. Additionally, see form_do() in the *GEM Application Environment Services Reference Guide*.

Input Arguments

`tree`	object tree containing the form to be handled
`edix`	first editable field

Output Arguments

`ret`	the object which caused exit from the form

Sample Call to C Language Binding

```
WORD form_exdo();

OBJECT FAR *tree;

WORD edix;

WORD ret;

ret = form_exdo(tree, edix);
```

FIX_ICON

This function converts all icons and bit images contained in an object tree from device-independent to device-dependent format.

The function takes as parameters the VDI handle of the previously opened screen workstation, and a pointer to the tree containing the icons and/or bit images.

Input Arguments

`vdi_handle`	VDI-handle of (virtual) workstation
`tree`	pointer to an object tree

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID fix_icon();

WORD vdi_handle;

OBJECT FAR *tree;

fix_icon(vdi_handle, tree);
```

EVNT_EVENT

evnt_event() is a short form of the evnt_multi() call. Instead of passing a lot of parameters to it, you only have to pass a pointer to a structure of type MEVENT (See your AES.H header file for a description of this structure). Using evnt_event() requires you to set only the structure members that are used for the event you are waiting for. So if you want to receive only messages, all you have to do is, to set

```
mevent.e_flags = MU_MESAG;
```

and

```
mevent.e_mepbuf = msg;
```

where msg should be of type WORD msg[8]

Input Arguments

`mevent`	structure of type MEVENT

Output Arguments

`event`	events that occurred

Sample Call to C Language Binding

```
WORD evnt_event();

MEVENT mevent;

WORD event;

event = evnt_event(&mevent);
```

Section 7
DOS Function Library

Section 7 describes the enhancements and modifications to the DOS Function Library provided with the revised GEM/3 Programmer's Toolkit. The DOS Function Library contains utility calls that let you bypass the standard C run-time library so your applications can use the DOS interface for disk I/O.

These descriptions assume a knowledge of the GEM library call structures and parameter conventions. For more information about these and other GEM system calls, refer to the *GEM Application Environment Services Reference Guide.*

DOS_CHDIR

Change directory. DOS-Call (hex) 3B.

Input Arguments

`pdrvpath`	drive and path to be set

Output Arguments

`ret`	TRUE if unable to change directory FALSE if change directory is successful

Sample Call to C Language Binding

```
WORD dos_chdir();

BYTE FAR *pdrvpath;

WORD ret;

ret = dos_chdir(pdrvpath);
```

DOS_GDIR

Get current directory. DOS-Call (hex) 47.

Input Arguments

`drive`	0 = default, 1 = A, 2 = B ...

Output Arguments

`pdrvpath`	pointer where path could be stored
`ret`	TRUE if get current directory fails FALSE if get current directory is successful

Sample Call to C Language Binding

```
WORD dos_gdir();

WORD drive, ret;

BYTE FAR *pdrvpath;

ret = dos_gdir(drive, pdrvpath);
```

DOS_GDRV

Get current drive. DOS-Call (hex) 19.

Input Arguments

`none`

Output Arguments

`drive`	current drive (0 = A, 1 = B, ...)

Sample Call to C Language Binding

```
WORD dos_gdrv();

WORD drive;

drive = dos_gdrv();
```

DOS_SDRV

Set current drive. DOS-Call (hex) 0E.

Input Arguments

drive	drive to be set (0 = A, 1 = B, ...)

Output Arguments

ret	TRUE if set current drive fails FALSE if set current drive is successful

Sample Call to C Language Binding

```
WORD dos_sdrv();

WORD drive, ret;

ret = dos_sdrv(drive);
```

DOS_SDTA

Set disk transfer adress. DOS-Call (hex) 1A.

Input Arguments

`ldta`	pointer to 44 bytes space

Output Arguments

`ret`	TRUE if set disk transfer address fails FALSE if set disk transfer address is successful

Sample Call to C Language Binding

```
WORD dos_sdta();

VOID FAR *ldta;

WORD ret;

ret = dos_sdta(ldta);
```

DOS_GDTA

Get disk transfer adress. DOS-Call (hex) 1A.

Input Arguments

`none`

Output Arguments

`ldta`	pointer to current disk transfer buffer

Sample Call to C Language Binding

```
VOID FAR *dos_gdta();

VOID FAR *ldta;

ldta = dos_gdta();
```

DOS_SFIRST

Search first directory entry. DOS-Call (hex) 4E.

Input Arguments

`pspec`	pointer to path and wildcard
`attr`	file attribute (must match)

Output Arguments

`ret`	TRUE if first directory entry is not found FALSE if first directory entry is found

Sample Call to C Language Binding

```
WORD dos_dfirst();

BYTE FAR *pspec;

WORD attr, ret;

ret = dos_sfirst(pspec, attr);
```

DOS_SNEXT

Search next directory entry. DOS-Call (hex) 4F.

Input Arguments

`none`

Output Arguments

`ret`	TRUE if next directory entry is not found FALSE if next directory entry is found

Sample Call to C Language Binding

```
WORD dos_snext();

WORD ret;

ret = dos_snext();
```

DOS_OPEN

Open an existing file. DOS-Call (hex) 3D.

Input Arguments

`pname`	pointer to path and filename
`access`	0 = read, 1 = write, 2 = read and write

Output Arguments

`handle`	handle of opened file

Sample Call to C Language Binding

```
WORD dos_open();
BYTE FAR *pname;
WORD access, ret;
handle = dos_open(pname, access);
```

DOS_CLOSE

Close previously opened or created file. DOS-Call (hex) 3E.

Input Arguments

`handle`	handle of opened file

Output Arguments

`ret`	TRUE if file is closed FALSE if file cannot be closed

Sample Call to C Language Binding

```
WORD dos_close();

WORD handle, ret;

ret = dos_close(handle);
```

READ_PIECE

Read a block of maximum 65535 bytes. DOS-Call (hex) 3F.

Input Arguments

`handle`	handle returned by dos_open or dos_create
`cnt`	number of bytes to be read
`pbuffer`	pointer to a buffer big enough to hold `cnt` bytes

Output Arguments

`read`	number of bytes that have been read

Sample Call to C Language Binding

```
UWORD read_piece();
WORD handle;
UWORD cnt, read;
VOID FAR *pbuffer;
read = read_piece(handle, cnt, pbuffer);
```

DOS_READ

Read a block larger than 65535 bytes. DOS_READ calls READ_PIECE several times.

Input Arguments

`handle`	handle returned by dos_open or dos_create
`cnt`	number of bytes to be read
`pbuffer`	pointer to a buffer big enough to hold `cnt` bytes

Output Arguments

`read`	number of bytes that have been read

Sample Call to C Language Binding

```
LONG dos_read();

WORD handle;

LONG cnt, read;

VOID FAR *pbuffer;

read = dos_read(handle, cnt, pbuffer);
```

DOS_LSEEK

Move file pointer. DOS-Call (hex) 42.

Input Arguments

`handle`	handle of opened file
`smode`	0 = from beginning of file
	1 = from current position
	2 = from end of file
`sofst`	offset to be seeked to

Output Arguments

```
newofst new offset
```

Sample Call to C Language Binding

```
LONG dos_lseek();
WORD handle, smode;
LONG sofst, newofst;
newofst = dos_lseek(handle, smode, sofst);
```

DOS_WAIT

Get termination code of subprocess. DOS-Call (hex) 4D.

Input Arguments

`none`

Output Arguments

`termcode`	termination code

Sample Call to C Language Binding

```
WORD dos_wait();

WORD termcode;

termcode = dos_wait();
```

DOS_ALLOC

Allocate memory. DOS-Call (hex) 48.

Input Arguments

`nbytes`	number of bytes to be allocated

Output Arguments

`ptr`	pointer to allocated memory

Sample Call to C Language Binding

```
VOID FAR *dos_alloc();
LONG nbytes;
VOID FAR *ptr;
ptr = dos_alloc(nbytes);
```

DOS_AVAIL

Get amount of free memory. DOS-Call (hex) 48.

Input Arguments

`none`

Output Arguments

`nfree`	number of free bytes

Sample Call to C Language Binding

```
LONG dos_avail();

LONG nfree;

nfree = dos_avail();
```

DOS_FREE

Free previously allocated memory. DOS-Call (hex) 49.

Input Arguments

`ptr`	pointer to allocated memory

Output Arguments

`ret`	TRUE if unable to free memory FALSE if memory is freed

Sample Call to C Language Binding

```
WORD dos_free();

VOID FAR *ptr;

ret = dos_free(ptr);
```

DOS_SPACE

Get disk free space. DOS-Call (hex) 36.

Input Arguments

`drv`	drive to be checked

Output Arguments

`ptotal`	pointer to a long (holds total of disk space)
`pavail`	pointer to a long (holds available disk space)
`ret`	TRUE if unable to get free disk space FALSE if get free disk space is successful

Sample Call to C Language Binding

```
WORD dos_space();

WORD drv, ret;

LONG FAR *ptotal, FAR *pavail;

ret = dos_space(drv, ptotal, pavail);
```

DOS_RMDIR

Remove directory. DOS-Call (hex) 3A.

Input Arguments

`ppath`	pointer to directory

Output Arguments

`ret`	TRUE if unable to remove directory FALSE if directory is removed

Sample Call to C Language Binding

```
WORD dos_rmdir();

BYTE FAR *ppath;

WORD ret;

ret = dos_rmdir(ppath);
```

DOS_CREATE

Create a new file. DOS-Call (hex) 3C.

Input Arguments

`pname`	pointer to path and filename
`attr`	file's attribute

Output Arguments

`handle`	handle of opened file

Sample Call to C Language Binding

```
WORD dos_create();
BYTE FAR *pname;
WORD attr, ret;
handle = dos_create(pname, attr);
```

DOS_MKDIR

Create new directory. DOS-Call (hex) 39.

Input Arguments

`ppath`	pointer to directory's name

Output Arguments

`ret`	TRUE if unable to create directory FALSE if directory created

Sample Call to C Language Binding

```
WORD dos_mkdir();

BYTE FAR *ppath;

WORD ret;

ret = dos_mkdir(ppath);
```

DOS_DELETE

Delete a file. DOS-Call (hex) 41.

Input Arguments

`pname`	pointer to name of file to be deleted

Output Arguments

`ret`	TRUE if unable to delete file FALSE if file is deleted

Sample Call to C Language Binding

```
WORD dos_delete();

BYTE FAR *pname;

WORD ret;

ret = dos_delete(pname);
```

DOS_RENAME

Rename a file. DOS-Call (hex) 56.

Input Arguments

poname	pointer to name of file to be renamed
pnname	new file name

Output Arguments

ret	TRUE if unable to rename file FALSE if file is renamed

Sample Call to C Language Binding

```
WORD dos_rename();
BYTE FAR *poname, FAR *pnname;
WORD ret;
ret = dos_delete(poname, pnname);
```

WRITE_PIECE

Write a block of maximum 65535 bytes. DOS-Call (hex) 3F.

Input Arguments

`handle`	handle returned by dos_open or dos_create
`cnt`	number of bytes to be written
`pbuffer`	pointer to buffer containing cnt bytes

Output Arguments

`write`	number of bytes that have been written

Sample Call to C Language Binding

```
UWORD write_piece();
WORD handle;
UWORD cnt, write;
VOID FAR *pbuffer;
write = write_piece(handle, cnt, pbuffer);
```

DOS_WRITE

Write a block larger than 65535 bytes. DOS_WRITE calls WRITE_PIECE several times.

Input Arguments

`handle`	handle returned by dos_open or dos_create
`cnt`	number of bytes to be written
`pbuffer`	pointer to a buffer containing cnt bytes

Output Arguments

`write`	number of bytes that have been written

Sample Call to C Language Binding

```
LONG dos_write();
WORD handle;
LONG cnt, write;
VOID FAR *pbuffer;
write = dos_write(handle, cnt, pbuffer);
```

DOS_CHMOD

Change a file's attribute. DOS-Call (hex) 43.

Input Arguments

`pname`	pointer to path and filename
`func`	0 = get attribute, 1 = set attribute
`attr`	files new attribute

Output Arguments

`nattr`	attribute that has been set

Sample Call to C Language Binding

```
WORD dos_chmod();
BYTE FAR *pname;
WORD func, attr, nattr;
nattr = dos_chmod(pname, func, attr);
```

DOS_SETDT

Set a file's date and time. DOS-Call (hex) 57.

Input Arguments

`handle`	handle of file (from dos_open or dos_create)
`time`	time to be set
`date`	date to be set

Output Arguments

`ret`	TRUE if unable to set file's date and time FALSE if file's date and time are set

Sample Call to C Language Binding

```
WORD dos_setdt();

WORD handle, date, time, ret;

ret = dos_setdt(handle, time, date);
```

DOS_GETDT

Get a file's date and time. DOS-Call (hex) 57.

Input Arguments

`handle`	handle of file (from dos_open or dos_create)
`time`	time of file
`date`	date of file

Output Arguments

`ret`	TRUE if unable to get file's date and time FALSE if able to get file's date and time

Sample Call to C Language Binding

```
WORD dos_getdt();
WORD handle, FAR *date, FAR *time, ret;
ret = dos_getdt(handle, time, date);
```

DOS_EXEC

Call a subprocess. DOS-Call (hex) 4B.

Input Arguments

`pname`	pointer to name of file to be executed
`para`	pointer to parameters that should be passed
`envrn`	segment address of environment variables

Output Arguments

`ret`	TRUE if unable to call subprocess FALSE if subprocess is called

Sample Call to C Language Binding

```
WORD dos_exec();
BYTE FAR *pname;
BYTE FAR *para;
UWORD envrn;
WORD ret;
ret = dos_exec(pname, para, envrn);
```

DOS_GETDATE

Get current date. DOS-Call (hex) 2A.

Input Arguments

`yr`	current year
`mo`	current month
`dy`	current day
`dw`	day of week (sunday = 0)

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID dos_getdate();
WORD FAR *yr, FAR *mo, FAR *dy, FAR *dw;
dos_getdate(yr, mo, dy, dw);
```

DOS_SETDATE

Set date. DOS-Call (hex) 2B.

Input Arguments

`yr`	year to be set
`mo`	month to be set
`dy`	day to be set

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID dos_setdate();
WORD yr, mo, dy;
dos_setdate(yr, mo, dy);
```

DOS_GETTIME

Get current time. DOS-Call (hex) 2C.

Input Arguments

`hr`	current hour
`mi`	current minute
`se`	current second
`hn`	current hundredth of a second

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID dos_gettime();
WORD FAR *hr, FAR *mi, FAR *se, FAR *hn;
dos_gettime(hr, mi, se, hn);
```

DOS_SETTIME

Set time. DOS-Call (hex) 2D.

Input Arguments

`hr`	hour to be set
`mi`	minute to be set
`se`	second to be set
`hn`	hundredth of a second to be set

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID dos_settime();
WORD hr, mi, se, hn
dos_settime(hr, mi, se, hn);
```

DOS_VERSION

Get version of operating system. DOS-Call (hex) 30.

Input Arguments

`vh`	high word of version number
`vl`	low word of version number
`oem`	OEM code
`user`	user code

Output Arguments

`none`

Sample Call to C Language Binding

```
VOID dos_version();
WORD FAR *vh, FAR *vl, FAR *oem, FAR *user;
dos_version(vh, vl, oem, user);
```

Section 8
EMS Library

This section describes modifications and enhancements to the EMS (Expanded Memory System) Library provided with the revised GEM/3 Programmer's Toolkit. The EMS Library contains utility calls for DOS-specific expanded memory management.

The descriptions in this section assume a knowledge of the GEM library call structures and parameter conventions. For more information about these and other GEM system calls, refer to the *GEM Application Environment Services Reference Guide.*

EMS_INST

This functions checks whether an EMS manager is installed or not.

Input Arguments

`none`

Output Arguments

`ret`	TRUE if EMS manager is installed FALSE if EMS manager not present

Sample Call to C Language Binding

```
WORD ems_inst();

WORD ret;

ret = ems_inst();
```

EMS_ERRCODE

This functions returns the error code of the last EMS operation. The description of the EMS error codes is at the end of this section.

Input Arguments

`none`

Output Arguments

`errcode`	see "EMS Error Codes" at the end of this section

Sample Call to C Language Binding

```
WORD ems_errcode();

WORD errcode;

errcode = ems_errcode();
```

EMS_NUM_PAGE

This function returns the number of pages that are held by the EMS manager. Each page has a size of 16 Kbytes.

Input Arguments

`none`

Output Arguments

`npages`	number of 16 Kbyte pages

Sample Call to C Language Binding

```
WORD ems_num_page();

WORD npages;

npages = ems_num_page();
```

EMS_FREE_PAGE

This function returns the number of EMS pages that are still available.

Input Arguments

```
none
```

Output Arguments

`fpages`	number of free EMS pages

Sample Call to C Language Binding

```
WORD ems_free_page();

WORD fpages;

fpages = ems_free_page();
```

EMS_FRAME_SEG

This function returns the segment address of the page frame. The page frame segment is the base address where four pages of 16 Kbyte memory are mapped.

Input Arguments

none

Output Arguments

baseframe	segment address of base page frame

Sample Call to C Language Binding

```
WORD ems_frame_seg();

WORD baseframe;

baseframe = ems_frame_seg();
```

EMS_ALLOC

This function allows allocation of memory pages in the expanded memory area. The function returns the EMS handle used to access allocated EMS memory.

Input Arguments

npages	number of pages to allocate

Output Arguments

handle	> 0 Memory handle to use
	FALSE if not enough memory available

Sample Call to C Language Binding

```
WORD ems_alloc();

WORD npages, handle;

handle = ems_alloc(npages);
```

EMS_MAP

Map logical page **`logp`** belonging to handle **`handle`** to physical page **`physp`**.

Input Arguments

`handle`	handle returned by ems_alloc
`logp`	number of logical page to be mapped
`physp`	number of physical page to be mapped to **`logp`**

Output Arguments

`ret`	TRUE if map is successful FALSE if map fails

Sample Call to C Language Binding

```
WORD ems_map();

WORD handle, logp, physp, ret;

ret = ems_map(handle, logp, physp);
```

EMS_FREE

Free all pages that belong to the specified handle.

Input Arguments

`handle`	EMS handle

Output Arguments

`ret`	TRUE if all pages are freed FALSE if uable to free all pages

Sample Call to C Language Binding

```
WORD ems_free();

WORD handle, ret;

ret = ems_free(handle);
```

EMS_VERSION

Return the version number of the EMS manager. The version number is returned as a two digits. For example, 35 indicates version number 3.5.

Input Arguments

`none`

Output Arguments

`version`	version number of EMS manager

Sample Call to C Language Binding

```
WORD ems_version();

WORD version;

version = ems_version();
```

EMS_SAVE_MAP

Save the state of the current mapping for a specified handle. This is useful when dealing with more than one handle. The function lets you save the mapping for different handles and then restore them as necessary.

Input Arguments

`handle`	handle for which mapping should be saved

Output Arguments

`ret`	TRUE if save is successful FALSE if unable to save

Sample Call to C Language Binding

```
WORD ems_save_map();

WORD handle, ret;

ret = ems_save_map(handle);
```

EMS_RESTORE_MAP

This is the opposite to the function EMS_SAVE_MAP. The function lets you restore a previously saved mapping for a specified handle.

Input Arguments

`handle`	handle for which mapping should be restored

Output Arguments

`ret`	TRUE if restore is successful FALSE if unable to restore

Sample Call to C Language Binding

```
WORD ems_restore_map();

WORD handle;

ret = ems_restore_map(handle);
```

EMS Error Codes

0x80	EMS manager damaged
0x81	EMS hardware error
0x82	unidentified error
0x83	unknown EMM handle
0x84	unknown EMM function
0x85	no more EMM handles
0x86	restore mapping error
0x87	more than the total number of pages requested
0x88	more pages requested than free
0x89	zero pages requested (only below version 4.0)
0x8A	logical page out of range
0x8B	physical page out of range
0x8C	mapping save area full
0x8D	mapping has already been saved
0x8E	restore map handle not known
0x8F	subfunction not known

Section 9
GEM AES and VDI Update

GEM AES Supplement

This part of the supplement contains the information required to bring the *GEM Application Environment Services Reference Guide* up to Release 3.1 of the GEM system software (GEM/3). Changes to the GEM AES include:

- a new call, `menu_click`
- rewording of the description of the `menu_bar` call
- restriction of the mouse button support of three Event Library calls

MENU_CLICK

Set or query whether menus are drop-down or pull-down.

Input Arguments

`click`	Menu type	
	0	Drop-down
	1	Pull-down
`setit`	Determines whether call queries or sets menu type	
	0	Query
	1	Set new value

Output Arguments

`retval`	Menu display mechanism	
	0	Drop down
	1	Pull down

Sample Call to C Binding

```
WORD menu_click();
WORD click, setit;

retval = menu_click(click, setit);
```

Parameter Block Binding

Control	Input	Output
`control(0) = 37`	`int_in(0) = click`	`int_out(0)=retval`
`control(1) = 2`	`int_in(1) = setit`	
`control(2) = 1`		
`control(3) = 0`		
`control(4) = 0`		

MENU_BAR

Activate or deactivate the application's menu bar.

The application should always call MENU_BAR to deactivate the menu bar before making its APPL_EXIT call.

Input Arguments

`showit`	A code for whether the menu bar is activated or deactivated
	`0` Menu bar deactivated
	`1` Menu bar activated
`tree`	Address of the object tree that forms this menu

Output Arguments

`retval`	A coded return message
	`0` Error
	`1` No error

Sample Call to C Binding

```
WORD    menu_bar();
WORD    retval, showit;
LONG    tree;

retval = menu_bar(tree, showit);
```

Parameter Block Binding

Control	Input	Output
`control(0) = 30`	`int_in(0) = showit`	`int_out(0)=retval`
`control(1) = 1`	`addr_in(0) = tree`	
`control(2) = 1`		
`control(3) = 1`		
`control(4) = 0`		

Event Library Calls

The following Event Library call parameters identify the mouse button that was activated at the time the user event occurred:

- EVNT_BUTTON: the output argument **pmb**
- EVNT_MOUSE: the output argument **pmb**
- EVNT_MULTI: the input argument **bmsk**

Note that for all these parameters, the GEM/3 system supports *only* the value `0x0001`, which identifies the left mouse button.

GEM VDI Supplement

This part of the supplement contains the information required to bring the *GEM Virtual Device Interface Reference Guide* up to Release 3.1 of the GEM system software (GEM/3).

Changes and Corrections

The following changes and corrections to the *GEM VDI Reference Guide* bring it up to Release 3.1 specifications:

- GEM VDI no longer uses an ASSIGN.SYS file. You can disregard all of page 1-4 and the top of page 1-5 (to the heading "The VDI Entry Point").
- The paragraph and example input in the middle of page 2-15 should be changed as follows:

  To load the VDI and start a VDI-only application (like a test program, shell, or debugger), enter the following command:

  ```
      GEMVDI -FILENAME
  ```

- In the sample C language program (page 2-18), delete the two comment lines that describe modifying ASSIGN.SYS. In addition, the command that runs SAMPLE.C is **`gemvdi -sample.exe`**.
- Correct Table 3-3 (page 3-4) as follows:

  Color index `8` is light gray.
  Color index `9` is dark gray.
  Color index `13` is dark yellow.
  Color index `14` is dark cyan.

- On page 4-21, the definition of the **`radius`** input argument should read "Length of circle's radius in x-axis units."
- On page 5-25, the function name in the last sentence of the first paragraph is incorrect. The correct function name is **`vst_load_fonts (77H)`**.

- For **vql_attributes** (page 8-7), the correct sample call to the C language binding is as follows:

```
WORD vql_attributes();
WORD handle, attrib[6];

vql_attributes(handle, attrib);
```

- The function call **v_bit_image** (page 9-26) applies only to the printer. For the screen, you must use raster operations (see Section 6 of the VDI guide).
- The call **v_xbit_image**, introduced in Release 3.01, is not supported in Release 3.1. Support for bit image rotation is provided in part by the inclusion of file-to-file image rotation in OUTPUT.APP.

GDOS Modifications

Command line switches have been added to the command line to be processed by GEMVDI.EXE and passed on to an application. The command line uses this form:

```
GEMVDI APPNAME [parameter...] [/S=screen] [/M=ab]
[/R=driver] [/I=info_path] [/F=font_path] [-program]
```

APPNAME	A GEM application .APP filename like **DRAW** or **PUBLISHR**. A filename must be given if the command line uses any of the parameters or switches described below.
parameter...	The name of one or more data files, which are passed as parameters of the first parameter specified. For example, the command line **GEMVDI DRAW PICTURE.GEM** starts the GEM® Draw Plus™ application and passes to the application file DRAW.APP the filename PICTURE.GEM as the name of a file to open.
/S=screen	The screen driver **screen** is loaded instead of the default screen.
/M=ab	The screen driver mouse configuration bytes (port and mouse type) are overwritten. The port value (**a**) and mouse type (**b**) are both patched to zero. (See the description later in this supplement of the MOUSE ID field in the file GEM-SETUP.TXT.)
/R=driver	The driver identified by **driver** is made resident for debugging purposes.
/I=inf_path	The GDOS creates font cacheing information files with the extension .INF. If this parameter is specified, these files are placed in the directory **inf_path** instead of \GEMAPPS\FONTS.
/F=font_path	Enables accessing fonts from directory **font_path** instead of \GEMAPPS\FONTS.

`-program`	The VDI should run **`program`** instead of GEM.EXE. (In Release 2.2 of the VDI, the delimiter was / instead of a dash.) No parameters may be passed to programs that are EXEC'd by the GDOS in this manner. If this argument is not given, the command line runs the default program, GEM.EXE, passing to it any parameters on the command line. Note that switches are *not* passed to GEM.EXE.

V_OPNWK (1H)

(Page 3-2 of the *GEM VDI Reference Guide*)

v_opnwk now provides support for run-time selection of output destination and page size, as well as providing feedback to the calling routine about escapement text and landscape rotation capability.

Additional Input Arguments

work_in[11] Output device type in the low-order byte:

0	File
1	Serial port
2	Parallel port
3	Device-specific (direct)
255	No change to default

Page size index in the high-order byte:

0 or 20	Letter size
5	Half size
10	B5 size
30	A4 size
40	Legal size
50	Double size
55	Broad sheet size
255	Use **work_in[101]** and **work_in[102]**

work_in[12+] Output port/name:

If **work_in[11]** is set to 1 or 2, then **work_in[12]** contains the port number. Otherwise, **work_in[12+]** contains the output file name (full path) with one character per word and null word terminator.

work_in[101] X page size in 1/100's inch

work_in[102] Y page size in 1/100's inch

Modified Output Arguments

`work_out[14]`	Set to 11 to indicate that escapement text is available; otherwise set to 10.
`work_out[24]`	Set to 11 to indicate that escapement text is the only kind of text available on the device; otherwise set to 10.
`work_out[44]`	In addition to the existing device type (0 - 4), -1 means landscape output can be handled by the device without rotation by the calling routine.

Sample Call to C Language Binding

```
WORD v_opnwk();
WORD work_in[103], work_out[57], handle;

v_opnwk(work_in, &handle, work_out);
```

Parameter Block Binding

Control	Input	Output
control(0) = 1	intin(n) = work_in[n], where n = 0 thru 102	intout(m)=work_out[m], where m = 0 thru 44
control(1) = 0		ptsout(i)=work_out[45+i], where i = 0 thru 11
control(2) = 6		
control(3) = 103		
control(4) = 45		
control(5) = 0		
control(6) = handle		

V_JUSTIFIED (B-AH)

(Page 4-27 of the *GEM VDI Reference Guide*)

The mapping for unsupported characters has been changed from a question mark to a blank space. See the fourth paragraph on page 4-27.

The information on page 4-28 has been changed considerably.

Input Arguments

`handle`	Device handle	
`word_space`	Word spacing flag:	
	`0x0000`	Do not modify inter-word spacing and do not return output information.
	`0x0001`	Modify inter-word spacing but do not return output information.
	`0x8000`	Do not modify inter-word spacing, but do return output information.
	`0x8001`	Modify inter-word spacing and return output information.
`char_space`	Character spacing flag:	
	`0x0000`	Do not modify inter-character spacing.
	`0x0001`	Modify inter-character spacing.
`string`	ASCII character string	
`x`	x-coordinate of the text alignment point	
`y`	y-coordinate of the text alignment point	
`length`	Requested length of the string, in x-axis units	

Output Arguments

`char_width`	Width of each character in pixels

Sample Call to C Language Binding

```
WORD v_justified();
WORD char_width(n);
WORD handle, x, y, length, word_space, char_space;
BYTE string(n);

v_justified(handle, x, y, string, length,
  word_space, char_space, char_width);
```

Parameter Block Binding

```
Control                      Input                       Output
control(0) = 11              intin(0) = word_space       intout(0) = char_width[0]
control(1) = 2               intin(1) = char_space       .
control(2) = 0               intin(n+2) = string[n]      .
control(3) = 2+n                                         intout(n-1) =
control(4) = 0 if word_space                                    char_width[n-1]
               equals 0x0000 or 0x0001
control(4) = n if word_space
               equals 0x8000 or 0x8001

control(5) = 10              ptsin(0) = x
control(6) = handle          ptsin(1) = y
                             ptsin(2) = length
                             ptsin(3) = 0
```

Note: n represents the number of characters in the string.

Memory Form Definition Block

(Page 6-2 in the *GEM VDI Reference Guide*)

In the Memory Form Definition Block (MFDB), a 32-bit pointer specified as -1L (0xffff:0xffff) defines the raster area as the quarter-screen buffer and indicates that it is located in graphic memory.

When the quarter-screen buffer is in graphic memory, the only legal source or destination of a bit copy operation—**`vro_cpyfm (6DH)`** for example—is the screen. In other words, if one operand of a bit copy operation is graphic memory, the other operand must be screen memory.

Graphic memory is memory that is accessible to the screen driver but not to the operating system. Typically it is located on a graphics card. If the AES has allocated the quarter-screen buffer to graphic memory, applications do not have access to this memory.

VQ_EXTND (66H)

(Page 8-14 in the *GEM VDI Reference Guide*)

Additional information for the availability of the quarter-screen buffer in graphic memory can be obtained through vq_extnd. If the values of output arguments **`work_out [26]`** and **`work_out [27]`** are non-zero, they represent the low word and high word, respectively, of the quarter-screen buffer size in graphic memory.

Modified Ouput Arguments

`work_out [20]`	Extended pixel size units	
	0	No extended precision pixel size information is available
	1	**`work_out [21]`** and **`work_out [22]`** give pixel size in 0.1 micron units
	2	**`work_out [21]`** and **`work_out [22]`** give pixel size in 0.01 micron units
	3	**`work_out [21]`** and **`work_out [22]`** give pixel size in 0.001 micron units
`work_out [21]`	Extended x dot size in **`work_out [20]`** units	
`work_out [22]`	Extended y dot size in **`work_out [20]`** units	
`work_out [23]`	x dots per inch	
`work_out [24]`	y dots per inch	
`work_out [25]`	Bit image rotation capabilities flag	
	0	Not applicable
	1	0-, 90-, 180-, 270-degree bit image rotations
`work_out [26]`	Low word of the quarter-screen buffer size	
`work_out [27]`	High word of the quarter-screen buffer size	

`work_out[28]`	bit 1: Bezier capability	
	`0`	driver has no Bezier capability
	`1`	driver has Bezier capability

Sample Call to C Language Binding

```
WORD vq_extnd();
WORD handle, owflag, work_out[57];

vq_extnd(handle, owflag, work_out);
```

Parameter Block Binding

Control	Input	Output
`control(0) = 102`	`intin(0) = owflag`	`intout(n) = work_out[n]`
`control(1) = 0`		` where n = 0 thru 44`
`control(2) = 6`		`pstout(m) =`
`control(3) = 1`		`work_out[m+45]`
`control(4) = 45`		` where m = 0 thru 11`
`control(5) = 0`		
`control(6) = handle`		

V_PLINE(6H) and V_FILLAREA(9H)

(Pages 4-4 and 4-10 in the *GEM VDI Reference Guide*)

The **`v_pline`** and **`v_fillarea`** calls have been extended to allow at least 1495 points. The actual maximum number is driver-dependent and can be found in **`work_out[14]`** from the extended inquire information option of the **`vq_extnd`** call.

A non-zero **`intin`** count indicates the presence of a list of indices of "jump points," which means that multiple disconnected polygons are supported.

VSF_XPERIMETER (68H)

This call is an extension of the existing **vsf_perimeter** call (see page 5-39 in the *GEM VDI Reference Guide*) that allows line style attributes to be used for filled area outlines. Note that both calls use the same opcode value. They are differentiated from each other by the INTIN count.

Input Arguments

handle	Device handle
on_off	Perimeter visibility flag 0 Turn perimeter outlining off 1 Turn perimeter outlining on -1 Do not change perimeter outlining
f_or_l	Perimeter attributes flag 0 Use normal fill color for perimeter 1 Use line style attributes for perimeter

Output Arguments

set_perimeter	Selected perimeter visibility

Sample Call to C Language Binding

```
WORD vsf_xperimeter;
WORD set_perimeter, handle, on_off, f_or_l;

set_perimeter = vsf_xperimeter(handle, on_off, f_or_l);
```

Parameter Block Binding

Control	Input	Output
control(0) = 104	intin(0) = on_off	intout(0) = set_perimeter
control(1) = 0	intin(1) = f_or_l	
control(2) = 0		
control(3) = 2		
control(4) = 1		
control(5) = 0		
control(6) = handle		

V_ALPHA_TEXT (5-19H)

(Page 9-31 in the *GEM VDI Reference Guide*)

Additions to currently defined control sequences:

`<DC2>6`	Begin superscript
`<DC2>7`	End superscript
`<DC2>8`	Begin subscript
`<DC2>9`	End subscript
`<DC2>A`	Begin letter-quality mode
`<DC2>B`	End letter-quality mode
`<DC2>C`	Begin expanded
`<DC2>D`	End expanded
`<DC2>E`	Begin light
`<DC2>F`	End light
`<DC2>G` thru `<DC2>V`	Reserved - ignored by driver
`<DC2>W`	Set pica
`<DC2>X`	Set elite
`<DC2>Y`	Set condensed
`<DC2>Z`	Set proportional

.OUT File Format

In addition to the control sequences described for the `v_alpha_text` call, the .OUT file format uses the following command to insert graphics into the output stream:

```
<ESC><ESC>GEM,x,y,w,h,D:\PATHNAME\FILENAME.EXT
```

`<ESC>` refers to ADE value 27. `x`, `y`, `w`, and `h` are given in character cell units. The origin of the graphics rectangle is relative to the current cursor position, not the top left corner of the page.

Font Header Format

The following modifications and additions have been made to the Font Header Format (Table F-1).

Modified Fields

(Page F-5 in the *GEM VDI Reference Guide*)

Byte Number	Description
0	Font identifier, if the identifier is less than or equal to 255. If the identifier is greater than 255, bit 5 in byte 67 should be set to 1 to indicate that `full_id` is used (see Flags definition below). In this case, bytes 110 and 111 are used as the full font identifier.
1	Weight: Bit 0 thicken (bold) Bit 1 light Bit 2 skew (italic) Bit 3 underline Bit 4 outline Bit 5 shadow

Additional Fields

(Page F-6 in the *GEM VDI Reference Guide*)

Byte Number	Description
`66 - 67`	Flags; additional values: Bit 4 Set if font data is paged out to disk. Bit 5 Set if font data in file is compressed. Bit 13 Set if `full_id` should be used. Bit 14 - 15 Reserved, must be zero.
`88 - 91`	`next_sect`—If the font data is broken into multiple sections, this pointer points to the next section. If the current section is the last section, set to 0. The data can be broken into sections in the following manner: a header identifying the characters described by the data in the current section, a pointer to the next section, the character data; a header for the next section, another pointer, character data; and so on.
`92 - 109`	Reserved, must be zero.
`110 - 111`	`full_id`—full identification (≥256) to use when bit 13 in Flags is set. In this case, `font_id` in byte 0 will be ignored.
`112 - 149`	Reserved, must be zero.
`150 - 151`	`compressed_size`—If font data is compressed, this is the number of bytes of compressed font data.

Bit Image File Format

(Appendix G of the *GEM VDI Reference Guide*)

The description of WORD 7 of the bit image file header format (Table G-1) is incorrect. The table should read as follows:

Table G-1. Bit Image File Header Format

Word	Contents
0	Image file version number
1	Header length in words
2	Number of planes (source device bits per pixel)
3	Pattern definition length (number of bytes)
4	Source device pixel width (microns)
5	Source device pixel height (microns)
6	Scan line width (pixels)
7	Number of scan lines
8	Bit image flag

The bit image file header (WORD 1) can be eight or nine words long. The optional ninth word allows printer drivers to support the dithered display of grayscale images. Drivers can accommodate files with or without the bit image flag.

In files with a 9-word header, bit 0 of word[8] has these possible values:

1	If a multi-plane image, planes are printed as gray levels.
0	If a multi-plane image, planes are printed as colors.

If the file is not a multi-plane image, bit 0 of word[8] has no meaning.

In a multi-plane image with an 8-word file header (an "old-style" image file), colors are printed as gray levels on a monochrome device, but the mapping of the colors to gray levels is not specified and may be device-dependent.

The information beginning on the next page replaces page G-2 of the *GEM VDI Reference Guide.*

Bit Image File Data Format

The bit image data is composed of descriptors for each scan line. (Word 7 of the file header tells how many scan lines are in the file.) The scan line descriptors are made up of the following:

- a vertical replication count, if the scan line is followed by one or more identical lines
- encoded line descriptor data for each color plane

The vertical replication count is a WORD value formatted as follows:

Byte	Contents
0	NUL
1	NUL
2	FF Hex
3	Count

The count indicates how many identical scan lines are defined by the descriptor data following the vertical replication count.

The encoded data for each color plane follows the vertical replication count and is presented in the following order:

first plane	—	red
second plane	—	green
third plane	—	blue
fourth plane	—	gray

Data is always provided for all bit planes defined in WORD 2 of the file header. The data is presented in any of three formats:

```
solid_run
pattern_run
bit_string
```

Note: Because scan line data is encoded in byte-wide packets (groups of eight pixels), the number of pixels described for each bit plane of a scan line is always a multiple of eight, as the following example demonstrates.

This example is a simple illustration of the workings of the vertical replication count and scan line descriptor data. It uses a hypothetical image file in which WORD 2 of the header is `00 01` (one color plane—in other words, a monochrome screen driver), the scan line width (WORD 6) is `00 28` (40 pixels), and the actual image is a solid horizontal line 34 pixels long and 4 pixels wide (four scan lines).

```
        00 00 FF 04 84 80 01 C0
```

In the vertical replication count (`00 00 FF 04`), the count is `04`, indicating that the descriptor data applies to four consecutive scan lines. The first descriptor (`84`) is a `solid_run` four bytes (32 pixels) long. The second descriptor (`80 01 C0`) is a `bit_string` one byte long, containing two black pixels and six blank pixels. The 32 pixels from the `solid_run` and the two pixels from the `bit_string` add up to the 34 pixels of the solid line, and the remaining six pixels fill out the 40-pixel line.

If WORD 2 of the file header had indicated four color planes, the vertical replication count would have been followed by descriptor pairs (`solid_run` and `bit_string`) for each color plane in turn.

V_COPIES (5-1CH)

This escape function allows the calling routine to specify the number of copies to be made of each page. All pages output before the workstation is closed are printed with the specified number of copies. This function applies to printers only.

Input Arguments

`handle`	Device handle
`count`	Number of copies

Sample Call to C Language Binding

```
WORD v_copies();
WORD handle, count;

v_copies(handle, count);
```

Parameter Block Binding

Control	Input
`control(0) = 5`	`intin(0) = count`
`control(1) = 0`	
`control(2) = 0`	
`control(3) = 1`	
`control(4) = 0`	
`control(5) = 28`	
`control(6) = handle`	

V_ETEXT (B-BH)

This function writes each character of a text string relative to the specified starting position. It is typically used to override the driver's default method of justification. This function applies to printers and plotters only.

Input Arguments

`handle`	Device handle
`x`	X-coordinate of starting position
`y`	Y-coordinate of starting position
`string`	Address of null-terminated text string
`offsets`	Address of WORD array of position offsets Each offset is an **x,y** pair of signed integers that indicate the position of the next character in the string relative to the starting position. The first offset pair affects the position of the first character in the string. Some drivers ignore the **y** component of each pair, in which case **y** is assumed to equal zero.

Sample Call to C Language Binding

```
WORD v_etext();
WORD handle, x, y, *offsets;
BYTE *string;

v_etext(handle, x, y, string, offsets)
```

Parameter Block Binding

Control	Input
`control(0) = 11`	`ptsin(0) = x`
`control(1) = length of string + 1`	`ptsin(1) = y`
	`ptsin(2) = offsets[0]`
`control(2) = 0`	`ptsin(3) = offsets[1]`
`control(3) = length of string`	.
	.
`control(4) = 0`	`ptsin(2n+1) = offsets[2n-1]`
`control(5) = 11`	`intin(0) = string[0]`
`control(6) = handle`	.
	.
	`intin(n-1) = string[n-1]`
	`where n = length of string`

V_ORIENT (5-1B)

This escape function allows the calling routine to select one of two page orientations: portrait (the default) or landscape. The function must be called before the output of any primitives or attributes.

Input Arguments

`handle`	Device handle	
`orientation`	Page orientation	
	`0`	Portrait
	`1`	Landscape

Sample Call to C Language Binding

```
WORD v_orient();
WORD handle, orientation;

v_orient(handle, orientation);
```

Parameter Block Binding

```
Control                    Input
control(0) = 5             intin(0) = orientation
control(1) = 0
control(2) = 0
control(3) = 1
control(4) = 0
control(5) = 27
control(6) = handle
```

V_TRAY (5-1D)

This escape function allows the calling routine to specify a paper tray or request manual feed. All pages output before the workstation is closed will be printed using the specified paper tray source.

Input Arguments

`handle`	Device handle
`tray`	Paper tray selection:

`-1`	manual feed
`0`	default paper tray
`1`	first optional paper tray
`n`	nth optional paper tray (n > 0)

Sample Call to C Language Binding

```
WORD v_tray();
WORD handle, tray;

v_tray(handle, tray);
```

Parameter Block Binding

Control	Input
`control(0) = 5`	`intin(0) = tray`
`control(1) = 0`	
`control(2) = 0`	
`control(3) = 1`	
`control(4) = 0`	
`control(5) = 29`	
`control(6) = handle`	

VST_EX_LOAD_FONTS (77H)

This function is an extension of the existing **`vst_load_fonts`** call (Page 3-18 of the *GEM VDI Reference Guide*), with two additional input arguments to provide control over font paging memory. The current defaults in units of paragraphs are:

	font_max	font_free
for screens:	5120 (80K)	0
for printers:	32767	640 (10K)

The GDOS attempts to allocate **`font_max`** paragraphs or all of available memory (whichever is smaller) less **`font_free`** paragraphs, and uses this amount for font paging.

Depending on your needs, you can use either version of this call. Note that both versions use the same opcode.

Input Arguments

`handle`	Device handle
`select`	Reserved, must be zero
`font_max`	Maximum number of paragraphs to allocate
`font_free`	Minimum number of paragraphs to leave free

Output Arguments

`additional`	Number of additional font identifiers

Sample Call to C Language Binding

```
WORD vst_ex_load_fonts();
WORD handle, select, font_max, font_free;

additional = vst_ex_load_fonts(handle, select, font_max,
font_free);
```

Parameter Block Binding

Control	Input	Output
control(0) = 119	intin[0] = select	intout(0) = additional
control(1) = 0	intin[1] = font_max	
control(2) = 0	intin[2] = font_free	
control(3) = 3		
control(4) = 1		
control(5) = 0		
control[6] = handle		

V_SET_APP_BUFF (FFFF-6H)

This call reserves a memory segment for use by GDOS extensions to produce Bezier curves. When the application makes Bezier calls, the buffer set aside by this call holds the polygon generated from the Bezier anchor points and direction points.

When not making Bezier calls, the application has free access to this buffer. A zero offset, segment, and size disable further use of this buffer and *must* be called to prevent accidental use of this memory when the application exits.

In the absence of this call, the GDOS allocates memory via DOS calls as needed. The size of the buffer varies according to the complexity of the Bezier—typically around 9K bytes.

Input Arguments

`offset`	Offset of buffer (first two bytes of **`address`**)
`segment`	Segment address of buffer (last two bytes of **`address`**)
`nparagraphs`	Number of paragraphs available

Output Arguments

`address`	Start address of memory area

Sample Call to C Language Binding

```
VOID v_set_app_buff();
LONGWORD address;
WORD naparagraphs;

v_set_app_buff(&address, nparagraphs);
```

Parameter Block Binding

Control	Input
`control(0) = -1`	`intin(0) = offset`
`control(1) = 0`	`intin(1) = segment`
`control(2) = 0`	`intin(2) = nparagraphs`
`control(3) = 3`	
`control(4) = 0`	
`control(5) = 6`	
`control(6) = handle`	

V_BEZ_ON (B-CH)

This call enables the GDOS Bezier capabilities. Note that while a handle is provided and the associated device driver is called, the GDOS Bezier extension is enabled for all devices when this call is made. All current GEM 3.1 drivers ignore this call; its only effect is within the GDOS itself.

Input Arguments

`handle`	Device handle

Output Arguments

`retval`	Maximum Bezier depth, a measure of the smoothness of the curve. The value, which can range from `0` to `7`, is an exponent of 2, giving the number of line segments that make up the curve. Thus, if `retval` is `0`, the curve is actually a straight line (one line segment). If `retval` is `7`, the curve is made of 128 line segments.

Sample Call to C Language Binding

```
WORD v_bez_on():
WORD handle, retval;

retval = v_bez_on(handle);
```

Parameter Block Binding

Control

```
control(0)  = 11
control(1)  = 1  (indicates ON)
control(2)  = 0
control(3)  = 0
control(4)  = 4
control(5)  = 13
control(6)  = handle
```

Output

```
intout(0)  = retval
```

V_BEZ_OFF (B-CH)

This call disables the GDOS Bezier capabilities. Any memory allocated by the GDOS for Bezier-generated polygons is released at this time. (See V_SET_APP_BUFF in this supplement for memory allocation information.)

Input Arguments

`handle`	Device handle

Sample Call to C Language Binding

```
WORD v_bez_off();
WORD handle

v_bez_off(handle);
```

Parameter Block Binding

Control

```
control(0) = 11
control(1) = 0 (indicates OFF)
control(2) = 0
control(3) = 0
control(4) = 0
control(5) = 13
control(6) = handle
```

V_BEZ (6-CH)

This call draws an unfilled Bezier on the specified device.

Input Arguments

`handle`	Device handle	
`count`	Number of vertices	
`xyarr`	Array of vertices	
`bezarr`	Array of vertex-type flags	
	`bit 0 = 1`	first point in a 4-point Bezier segment (a curve—the four points are two *anchor points* and two *direction points*)
	`bit 1 = 1`	jump point—a point connecting two regions without drawing a line between them ("move to here" instead of "draw to here")

Output Arguments

`npts`	Number of points in resulting polygon
`nmove`	Number of moves in resulting polygon
`minx`	Minimum x extent of rectangle ("bounding box") surrounding the curve
`miny`	Minimum y extent of bounding box
`maxx`	Maximum x extent of bounding box
`maxy`	Maximum y extent of bounding box

Sample Call to C Language Binding

```
VOID v_bez();
WORD handle, count, xyarr, extent;
CHAR bezarr;

v_bez(handle, count, xyarr, bezarr, extent);
```

Parameter Block Binding

Control	Input	Output
`control(0) = 6`	`intin(0) = bezarr`	`intout(0) = npts`
`control(1) = count`	`ptsin(0) = xyarr`	`intout(1) = nmove`
`control(2) = 2`		`intout(2) = reserved`
`control(3) = (count + 1)/2`		`intout(3) = reserved`
`control(4) = 6`		`intout(4) = reserved`
`control(5) = 13`		`intout(5) = reserved`
`control(6) = handle`		`ptsout(0) = minx`
		`ptsout(1) = miny`
		`ptsout(2) = maxx`
		`ptsout(3) = maxy`

V_BEZ_FILL (9-CH)

This call draws a filled Bezier on the specified device.

Input Arguments

`handle`	Device handle	
`count`	Number of vertices	
`xyarr`	Array of vertices	
`bezarr`	Array of vertex-type flags	
	`bit 0 = 1`	first point in a 4-point Bezier segment (a curve—the four points are two *anchor points* and two *direction points*)
	`bit 1 = 1`	jump point—a point connecting two regions without drawing a line between them ("move to here" instead of "draw to here")

Output Arguments

`npts`	Number of points in resulting polygon
`nmove`	Number of moves in resulting polygon
`minx`	Minimum x extent of rectangle ("bounding box") surrounding the curve
`miny`	Minimum y extent of bounding box
`maxx`	Maximum x extent of bounding box
`maxy`	Maximum y extent of bounding box

Sample Call to C Language Binding

```
VOID v_bez_fill();
WORD handle, count, xyarr, extent;
CHAR bezarr;

v_bez_fill(handle, count, xyarr, bezarr, extent);
```

Parameter Block Binding

Control	Input	Output
control(0) = 9	intin(0) = bezarr	intout(0) = npts
control(1) = count	ptsin(0) = xyarr	intout(1) = nmove
control(2) = 2		intout(2) = reserved
control(3) = (count + 1)/2		intout(3) = reserved
control(4) = 6		intout(4) = reserved
control(5) = 13		intout(5) = reserved
control(6) = handle		ptsout(0) = minx
		ptsout(1) = miny
		ptsout(2) = maxx
		ptsout(3) = maxy

V_BEZ_QUAL (5-63H)

This call specifies the speed/quality tradeoff parameter for Beziers.

Input Arguments

`handle`	Device handle
`prcnt`	Requested speed/quality factor in percent

Output Arguments

`actual`	Actual speed/quality used

Sample Call to C Language Binding

```
WORD v_bez_qual();
WORD (handle, prcnt, actual);

v_bez_qual(handle, prcnt, actual);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = 32	intout(0) = actual
control(1) = 0	intin(1) = 1	
control(2) = 0	intin(2) = prcnt	
control(3) = 3		
control(4) = 1		
control(5) = 99		
control(6) = handle		

VS_BKCOLOR (5-66H)

This call sets the background color for the device associated with **`handle`**, usually a camera device.

Input Arguments

`handle`	Device handle
`color`	Background color index

Sample Call to C Language Binding

```
VOID vs_bkcolor():
WORD handle, color

vs_bkcolor(handle, color);
```

Parameter Block Binding

Control	Input
control(0) = 5	intin(0) = color
control(1) = 0	
control(2) = 0	
control(3) = 1	
control(4) = 0	
control(5) = 102	
control(6) = handle	

VS_GRAYOVERRIDE (85H)

This call overrides the gray level specified with the **`vsf_style`** call, patterns 2,1 through 2,8 (see page 5-36 of the *GEM VDI Reference Guide*). The application should specify the closest index in the normal fill pattern set and follow it with a **`vs_grayoverride`** call to "fine-tune" that gray level one devices that support such fine tuning. This call is currently implemented in the PostScript® driver.

Input Arguments

`handle`	Device handle
`grayval`	Gray value in tenths of a percent
	`0` white
	`1000` black

Sample Call to C Language Binding

```
VOID vs_grayoverride();
WORD handle, grayval;

vs_grayoverride(handle, grayval);
```

Parameter Block Binding

Control

```
control(0) = 133
control(1) = 0
control(2) = 0
control(3) = 1
control(4) = 0
control(5) = 0
control(6) = handle
```

V_PAT_ROTATE (86H)

This call specifies pattern rotation angle. It is implemented only in printer drivers and is restricted to multiples of 90 degrees.

Input Arguments

`handle`	Device handle
`angle`	Angle in tenths of a degree

Sample Call to C Language Binding

```
VOID v_pat_rotate();
WORD handle, angle;

v_pat_rotate(handle, angle);
```

Parameter Block Binding

Control	Input
`control(0) = 134`	`intin(0) = angle`
`control(1) = 0`	
`control(2) = 0`	
`control(3) = 1`	
`control(4) = 0`	
`control(5) = 0`	
`control(6) = handle`	

V_SETRGBI (5-4844H)

This call overrides a previously set color specification with an RGB triple (color devices) or intensity (monochrome devices). This call is currently implemented only for the PostScript driver.

Input Arguments

`handle`	Device handle
`primtype`	Primitive type `17` line `20` marker `22` text `25` fill
`r`	Red component
`g`	Green component
`b`	Blue component
`i`	Intensity

Sample Call to C Language Binding

```
VOID v_setrgbi();
WORD handle, primtype, r, g, b, i;

v_setrgbi(handle, primtype, r, g, b, i);
```

Parameter Block Binding

Control	Input
`control(0) = 5`	`intin(0) = primtype`
`control(1) = 0`	`intin(1) = r`
`control(2) = 0`	`intin(2) = g`
`control(3) = 5`	`intin(3) = b`
`control(4) = 0`	`intin(4) = i`
`control(5) = 0x4844`	
`control(6) = handle`	

V_TOPBOT (5-4845H)

This call is an alternative to **`vst_height`** (page 5-20 of the *GEM VDI Reference Guide*). It uses top to bottom distance for text scaling, instead of top to baseline distance.

Input and output arguments are the same as for **`vst_height`**.

Sample Call to C Language Binding

```
VOID v_topbot();
WORD handle, height, char_width, char_height, cell_width,
cell_height;

v_topbot(handle, height, &char_width, &char_height,
&cell_width, &cell_height);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	ptsin(0) = 0	ptsout(0) = char_width
control(1) = 1	ptsin(1) = height	ptsout(1) = char_height
control(2) = 4		ptsout(2) = cell_width
control(3) = 0		ptsout(3) = cell_height
control(4) = 0		
control(5) = 0x4845		
control(6) = handle		

V_PS_HALFTONE (5-20H)

This call controls the parameters for PostScript halftoning. It provides direct access to analogous PostScript language parameters. It is implemented only for the PostScript driver.

Input Arguments

`handle`	Device handle
`index`	Halftone type
	0 Dot screen
	1 Line screen
	2 Ellipse screen
	3 Custom (user-defined)
`angle`	Halftone screen angle
`frequency`	Halftone screen frequency

Sample Call to C Language Binding

```
VOID v_ps_halftone();
WORD handle, index, angle, frequency;

v_ps_halftone(handle, index, angle, frequency);
```

Parameter Block Binding

Control	Input
control(0) = 5	intin(0) = index
control(1) = 0	intin(1) = angle
control(2) = 0	intin(2) = frequency
control(3) = 3	
control(4) = 0	
control(5) = 32	
control(6) = handle	

Section 10
Files and Devices Update

DDF Files

Bitstream® Fontware® uses Device Description Files (DDF) to contain the device-dependent information that is required for generating the correct fonts in the correct format. DDF files also provide information about the device for the user-interface portion of Fontware. In the following description of the fields that can occur in a DDF file, S/P indicates a field used in both screen and printer DDF files and P indicates a field used only in a printer DDF file.

`menulabel`	S/P	unused
`manufacturer`	S/P	first part of menu label (used in the Printer Model menu in Fontware)
`model`	S/P	last part of menu label (used in the Printer Model menu in Fontware)
`printer`	S/P	Screen/printer flag
	0	screen device
	1	printer device
`hdpi`	S/P	horizontal dots per inch
`vdpi`	S/P	vertical dots per inch
`driverload`	S/P	font management responsibility
	0	Fonts are loaded and managed by GDOS font manager.
	1	Device driver loads and manages its fonts.
`driver`	S/P	Xerox® Ventura Publisher® driver name that is patched into the corresponding width table. This name should be identical to the short name in the **zyxg** patch area of the driver. See "Device Names" at the end of this supplement.

`fmt`	S/P	identifies screen or printer font format conversion program For example, if the value is this field is **GEM**, that identifies the font conversion program as CVTGEM.EXE. If the value is **HPF**, the conversion program is CVTHPF.EXE. The value in this field follows "CVT" in the conversion file program name. The value must be **GEM** if the driver uses the GDOS font manager.
`makefon`	S/P	unused—should be zero
`usepfm`	S/P	unused—should be zero
`devkey`	S/P	Seventh character of the font file name. By convention, a unique letter is assigned by resolution.
`devmode`	S/P	eighth character of the font file name In screen DDF files, P indicates a Portrait font and L and Landscape font. In printer DDF files this value is overridden by GENGEMIF.EXE.
`maxbmap`	S/P	maximum size used by font conversion program specified by `fmt` (above)
`maxoffset`	S/P	maximum offset used by font conversion program
`minoffset`	S/P	minimum offset used by font conversion program
`maxcell`	S/P	maximum size used by font conversion program
`maxsw`	S/P	maximum size used by font conversion program
`gemext`	S/P	font file extension (unused by devices that manage their own fonts)
`rle`	P	run-length encoding—should be `1`
`kerning`	P	kerning flag
	`0`	no kerning
	`1`	kerning

`dta1`	P	Specifies the first program of the intial stage of font processing. Recommended value is **`vfms`**, which generates Ventura width tables.
`dta2`	P	Specifies the second program of the initial stage of font processing. Recommended value is **`vf2wd`**, which generates Ventura width tables.
`dtb1`	P	Specifies the first program of the final stage of font processing.
`dtb2`	P	Specifies the second program of the final stage of font processing.
`orientation`	P	generate Portrait and/or Landscape fonts
`hp_big`	P	generate Series II soft fonts (may be larger than 255 dots)
`usesizes`	P	Not used by raster devices.
`usebco`	P	Not used by raster devices.

NOTE: The preferred naming convention for DDF files is that the file *name* be the value found in the **`gemext`** field. See the sample files on the following pages.

Sample DDF Files

This is a listing of a sample DDF file for a VGA™-type screen driver:

```
VGA.DDF

menulabel=vga
manufacturer=ibm
model=VGA
printer=0
hdpi=91
vdpi=91
driverload=0
driver=VGA
fmt=gem
makefon=0
usepfm=0
devkey=v
devmode=p
maxbmap=512
maxoffset=655
minoffset=-655
maxcell=655
maxsw=655
gemext=vga
```

This is a listing of a sample DDF file for an Epson® LQ-series printer.

```
ELQ.DDF

menulabel=Epson LQ Series
manufacturer=Epson
model=LQ
printer=1
hdpi=180
vdpi=180
driverload=0
driver=Epson LQ
dta1=vfms
dta2=vf2wd
fmt=gem
makefon=0
usepfm=0
devkey=d
devmode=p
maxbmap=512
maxoffset=655
minoffset=-655
maxcell=655
maxsw=655
gemext=elq
```

This is a listing of a sample DDF file for a Hewlett-Packard® LaserJet® Series II printer driver:

```
HPH.DDF

menulabel=HP LaserJet Series II - HP Softfonts
manufacturer=HP
model=LaserJet Series II - HP Softfonts
printer=1
hdpi=300
vdpi=300
driverload=1
driver=HP LJ+, 300 dpi
dta1=vfms
dta2=vf2wd
fmt=hpf
makefon=0
usepfm=0
devkey=1
devmode=p
maxbmap=512
maxoffset=655
minoffset=-655
maxcell=655
maxsw=655
gemext=hph
rle=1
kerning=1
orientation=2
hp_big=1
```

CNF Files

All GEM 3.1 printer drivers read device- and system-dependent configuration files that have filenames of the form **`ddd.CNF`**. **`ddd`** is a driver identification taken from the driver filename, which uses the form **`PDddd9.fff`**. (**`fff`** is a font extension like VGA or EGA.) There are three kinds of CNF files, used by GEM font drivers, Hewlett-Packard soft font drivers, and the PostScript driver.

CNF files are pure ASCII text. Individual entries in the files use this format:

```
KEYWORD (PARM1,PARM2,PARM3,...)
```

KEYWORD describes the function to be adjusted or included, and PARM1 PARM2 PARM3... are modifying or describing parameters. Parameters may be separated by commas. Note that both the CNF files themselves and all entries in the files are optional.

GEM Font Drivers

This is the format of a CNF file for a GEM font driver:

```
MARGINS (XL XR YT YB)
```

`MARGINS` Sets margins that limit graphics output to printable area.

`XL`	left margin in device units
`XR`	right margin in device units
	The **`XL`** and **`XR`** values typically default to 0.5″ in device units. For a 120 dpi printer, they would equal 60.
`YT`	top margin in device units
`YB`	bottom margin in device units
	The **`YT`** and **`YB`** values typically default to zero.

Hewlett-Packard Soft Font Drivers

This is the format of a CNF file for a Hewlett-Packard soft font driver:

```
MARGINS (XL XR YT YB)
HFI(ON_OFF)
DOWNPATH(DIR)
PERMFONT(ID FILENAME)
FONTSPEC(FILENAME ID SIZE ATTR MAP)
```

MARGINS	Same function and parameters as in GEM font driver CNF file.
HFI	Flag indicating whether HP Font Information (HFI) file search is requested. HFI files should be located in the directory that contains the downloadable soft fonts.

ON_OFF = 0	driver search for HFI files disabled (the default)
ON_OFF = 1	driver search for HFI files enabled

DOWNPATH Identifies directory that contains HFI files and soft font files.

DIR Path name of directory. If a relative path specification is given, it is taken as relative to the directory that contains the driver. The default is ".", the directory containing the driver.

PERMFONT Indicates that the soft font has already been downloaded to the printer. In that event, the driver examines the soft font file for character width information but does not send the font down to the printer. This keyword's parameters have no default values.

ID	soft font identifier
FILENAME	soft font filename

FONTSPEC

An alternative to HFI files. Provides the driver with information about a soft font that is available for downloading.

`FILENAME`	soft font filename (with no extension—the driver will use .SFP and .SFL)
`ID`	GEM font identifier
`SIZE`	font size in points
`ATTR = 0`	font attribute: Normal
`ATTR = 1`	font attribute: Bold
`ATTR = 4`	font attribute: Italic
`ATTR = 5`	font attribute: Bold Italic
`MAP = 0`	remap character set flag: HP character set
`MAP = 1`	remap character set flag: GEM/Ventura character set

PostScript Driver

This is the format of the CNF file used by the PostScript driver:

```
MARGINS (XL XR YT YB)
PFI(ON_OFF)
PSFONTS(DIR)
EOFTYPE(TYPE)
IMGTYPE(TYPE)
COLTYPE(TYPE)
FONT(NAME ID ATTR MAP RESFLAG)
```

MARGINS Same function and parameters as in GEM font driver CNF file.

PFI Flag indicating whether PostScript Font Information (PFI) file search is requested. PFI files should be located in the directory that contains the downloadable PostScript ASCII or binary format fonts.

`ON_OFF = 0`	driver search for PFI file disabled (the default)
`ON_OFF = 1`	driver search for PFI file enabled

PSFONTS Identifies directory that contains PFI file and associated PostScript downloadable font files.

`DIR`	Path name of directory. If a relative path specification is given, it is taken as relative to the directory that contains the driver. The default is ".", the directory containing the driver.

EOFTYPE Specifies the driver method to be used for marking end of PostScript job/file.

`TYPE = PC`	Ctrl-D is appended to the PostScript output (the default)
`TYPE = MAC`	no characters are appended to the PostScript output

IMGTYPE Indicates how bitmap image data is to be translated into PostScript.

METHOD = COMPACT

Image data is sent in a compressed form and is decoded by the PostScript interpreter.

METHOD = FAST

Image data is decompressed before translation to the appropriate PostScript string.

COLTYPE Specifies to driver whether "setrgbcolor" or "setgray" PostScript functions should be used.

COLOR	Use "setrgbcolor" function (the default).

When the **COLOR** parameter is set, PostScript handles mapping of colors to gray levels for monochrome printers.

MONO	Use "setgray" function.

FONT Provides an alternative to supplying a PFI file for the PostScript font.

NAME	Font PostScript name
ID	GEM font identification number (for example, 2 = Swiss)
ATTR = M	font attribute: Normal
ATTR = B	font attribute: Bold
ATTR = I	font attribute: Italic
ATTR = BI	font attribute: Bold Italic
MAP = TEXT	character set re-encoding enabled
MAP = PI	character set re-encoding disabled (for symbol fonts)
RESFLAG = RES	font is resident

RESFLAG = DOWN:filename

Font **filename** must be downloaded.

ATM Files

Alpha Text Mapping (ATM) files are used by non-Postscript printer drivers to allow device-dependent and user-dependent mapping of characters above the standard ASCII set. ATM files are used only for alpha text and they are used only when the driver attempts to output characters in the range 128-255. These files allow the system to use a single driver for printers with the same graphics mode protocol but different alpha mode protocols or capabilities. For example, the Hewlett-Packard Laserjet in its simplest form cannot print the © or ® symbols, but if fitted with a "Legal" cartridge, it can access these characters with the appropriate escape sequence. The ATM files have a very simple format in which all characters are represented by their two-character hexadecimal representation. For example, decimal character 128 is written as 80. The following example, taken from EHI.ATM, illustrates the syntax of these files:

```
80 1B52015C1B5200
81 1B52027D1B5200
82 1B52017B1B5200
83 61085E
```

The first number on each line is the character requested by the application. The number sequence that follows identifies the set of characters that are sent to the printer. Any character without an entry is transmitted to the printer unchanged. In the example above, the GEM International character set character â—with decimal value 131 or hexadecimal value 83—is translated by the driver into the sequence `61 08 5e`: the character "a", a backspace, and the character "^".

Section 11
GEM Setup Text Files

The GEM Setup program uses two ASCII text files: GEMSETUP.MSG and GEMSETUP.TXT. This note describes the format of these two files so you can modify or translate the existing text files or create new ones.

GEMSETUP.MSG

GEMSETUP.MSG contains the messages, menus, and prompts the user sees while running the GEM Setup program. Here are two excerpts from this file:

```
@PROMPT_PTR
*** LINES: 4
{                               Welcome to GEM Setup!
    This program installs GEM/3 onto your computer.
    Do you want to install GEM/3 for the first time or change an existing
    GEM/3 installation?
}
.
.
.
@FLAB_PTR

The following strings are floppy disk labels.

*** LINES: 2

{GEM DESKTOP DISK
GEM STARTUP DISK
}
```

The file is made up of these elements:

- A pointer code that identifies how the following text will be used. These pointers are delimited by an at-sign (@) and are linked to the code in GEMSETUP.EXE. For that reason, they *must not be changed*. The pointer codes are described fully later in this note.
- A line count indicating how many lines of text are available at this point in the program. For example, four lines are available for the opening message and first prompt. You can change the content of these lines, but you must use the number of lines indicated. If you do not, all subsequent lines will be offset by a number of lines, and the wrong prompts and messages will appear on the screen.

- The prompt, message, or menu text that appears on the screen. This text is set off by braces ({}) and must occupy the number of lines specified in the line count.
- Optional comment text. This text is placed between the pointer code and the line count and is identified by the absence of any delimiter character.

Pointer Codes

These are the pointer codes used by GEMSETUP.MSG:

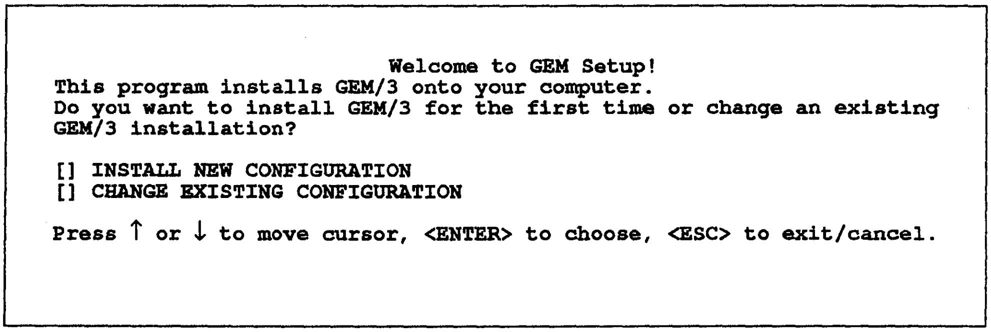
@PROMPT_PTR	Prompts, queries, and messages that form the bulk of the program's interaction with the user.
@CHOICE_PTR	Menus of options from which the user can choose. A box, check-mark, or other choice mechanism (it is system-dependent) appears to the left of each option.
@FOOTER_PTR	Footer lines that tell the user how to select options.
@HVOL_PTR	Disk volume labels. These labels are used by the code to identify the disk being used for a particular operation. The string *must* be eleven characters long. If the number of characters used is less than eleven, pad the inside of the string with blank spaces. The first example below is correct; the second example is incorrect. (The first line shows the character count.)

```
            12345678901
correct     GEM     SCRN
incorrect   GEM SCRN
```

@HLAB_PTR	Disk label strings. These strings are swapped into the text of **@PROMPT_PTR** to identify for the user the disks required for a given operation. The **XX** strings are placeholders for driver pack label strings, which come from GEMSETUP.TXT on the driver pack disk. Do not translate or alter the **XX** strings.
@KEYWORDS_PTR	Unique characters for the GEMSETUP.TXT label strings. Do not translate or alter.

@FLAB_PTR	Floppy disk labels. These strings are swapped into the text of **@PROMPT_PTR** to identify the disks created for a floppy disk installation.
@FVOL_PTR	Floppy disk volume labels.
@COPY_PTR **@TOO_MANY** **@D_SPACE**	Messages displayed by the GEM Setup program.

The example below shows the first GEM Setup screen and identifies the pointer codes for the types of text on the screen.



```
                          Welcome to GEM Setup!
This program installs GEM/3 onto your computer.
Do you want to install GEM/3 for the first time or change an existing
GEM/3 installation?

[] INSTALL NEW CONFIGURATION
[] CHANGE EXISTING CONFIGURATION

Press ↑ or ↓ to move cursor, <ENTER> to choose, <ESC> to exit/cancel.
```

GEMSETUP.TXT

The GEMSETUP.TXT file contains the strings that describe the various devices and their associated files. Here are two excerpts from GEM-SETUP.TXT:

```
@SCREEN
{
|DESCRIPTION|IBM Enhanced Card & 16-Color Display (640x350)
|SHORT DESCR|EGA HiRes 16
|FILENAME|SDEHF8.EGA
|SRC DISK|GEM SCREEN DISK #1
|FNT WILDCRD|*.EGA
|LONG DESCRP|
Choose this entry if your system is equipped with an IBM Enhanced
Graphics Adapter card, with at least 128K of graphics memory on
the card, and an IBM enhanced color display.  This 16-color
display offers a resolution of 640 horizontal by 350 vertical
pixels.
}
.
.
.
@PRINTER
{
|DESCRIPTION|Hewlett Packard Laserjet II Printer (300 x 300 Dots/Inch)
|SHORT DESCR|HP Laser II
|FILENAME|PDHPU8.B30
|SRC DISK|GEM PRINTER DISK #3
|FNT WILDCRD|*.B30
|LONG DESCRP|
Choose this entry if you are using a Hewlett Packard Laserjet
II printer.  This printer offers a print resolution of 300 x 300
dots per inch.
}
```

The descriptions are grouped according to the device:

```
@METAFILE
@SCREEN
@PRINTER
@PLOTTER
```

Within each category, the device descriptions are delimited by braces, as shown in the examples.

Each field name is delimited by a broken vertical bar (| |), which—with one exception—is followed immediately by the field content. The exception is the "long description" field, whose content starts in column 1 of the next line.

These are the fields used in GEMSETUP.TXT:

DESCRIPTION	A brief description of the device. This strings appears in the menus presented by GEM Setup. Maximum field length is 80 characters.
SHORT DESCR	An even more brief description of the device. This string appears below the device's icon in the GEM Output program. Maximum field length is 13 characters. See "Device Names" on at the end of this section.
FILENAME	The filename for the device driver. This field is required. Maximum field length is 13 characters.
SRC DISK	The disk on which the driver is found. This string should match one of the string names listed under **@HLAB_PTR** in GEMSETUP.MSG. Maximum field length is 40 characters.
FNT WILDCRD	A string in the form ***.EXT** that identifies the file extension used by the fonts associated with this device. Maximum field length is 40 characters.
AUX FILE	The filename (or names) of auxiliary files used by a printer. These are typically configuration or text mapping files. If this field lists more than one filename, the names are separated by a semi-colon. Maximum field length is 67 characters.
FONT DISK	The disk on which printer fonts are found. This string should match one of the string names listed under **@HLAB_PTR** in GEMSETUP.MSG. Maximum field length is 40 characters.

`MOUSE ID`	A unique one-byte code that identifies the mouse to the screen driver. The mouse ID occupies the second byte following the string **`zyxg`** in the driver file. (The first byte is the mouse port—`00` for COM1, `01` for COM2.) Reserved mouse ID's are:

`00`	No mouse
`01`	Mouse Systems™ PC Mouse™ / SummaMouse™ / Compatibles
`02`	Bus Mouse (Requires file MOUSE.COM)
`03`	Microsoft Serial Mouse (RS232)
`04`	SummaSketch™ 1201 Stylus-Type Tablet
`05`	SummaSketch 1201 Cursor-Type Tablet
`06`	SummaSketch 961 Stylus-Type Tablet
`07`	SummaSketch 961 Cursor-Type Tablet
`08`	Summagraphics™ MM1812 Stylus-Type Tablet
`09`	Summagraphics MM1812 Cursor-Type Tablet
`10 (hex 0A)`	IBM® Personal System/2™ Mouse

`LONG DESCRP`	A long description of the device. This text is displayed when the user asks for help in GEM Setup. Maximum field length is 80 characters per line, with a maximum of 20 lines.

Device Names

Ventura Publisher users can encounter an alert telling them that their printer and width table are incompatible, even though the width table is the correct one for the device.

This supposed incompatibility arises when the `SHORT DESCR` in the printer driver does not match the device identification in the width table file. (This identification is actually the `driver` field from the printer's DDF file and has been embedded in the width table by Fontware.) Ventura compares the two values and, if they do not match, returns the alert.

The alert is often more an annoyance than a sign of a true incompatibility. The user can ignore the alert if the incompatibility is simply a matter of inconsistencies in naming. For example, GEM printer drivers refer to the Hewlett-Packard LaserJet Series II as `HP Laser 300`, and Ventura drivers refer to it as `HP LJ+, 300dpi`.

To avoid this alert, make sure the `SHORT DESCR` and `driver` values are the same.

5149-2374-001