FlexOS™ Supplement for Intel® iAPX™ 286-based Computers

First Edition: November 1986

Foreword

This supplement contains processor-dependent information and descriptions for the Intel® iAPX™ 286 (80286) version of the FlexOS™ operating system.

Section 1 supplements the FlexOS Programmer's Guide. Refer to this section for chip-specific information related to the use of Supervisor Calls (SVCs) and program development.

Section 2 supplements the FlexOS System Guide. Refer to this section for driver-related and system generation information.

Section 3 describes the FlexOS front end. Refer to this section for guidelines for running certified applications and for writing new applications to run under the FlexOS front end.

Section 4 describes the FlexOS Virtual Device Interface (VDI). Refer to this section for information on device support and VDI configuration and installation.

Contents

1 Supplement to the Programmer's Guide

1.1 Supervisor Call Distinctions 1-1
1.1.1 MALLOC 1-1
1.1.2 CONTROL 1-2
1.1.3 EXCEPTION Numbers 1-4
1.2 Assembly Language Programming Tools and Conventions 1-5
1.2.1 Reserved File Extensions 1-5
1.2.2 FlexOS Entry Mechanism 1-7
1.3 Program Load Formats 1-8
1.3.1 Code, Data, and Stack Group Descriptors 1-9
1.3.2 Fix-Up Information 1-10
1.3.3 Shared Run-Time Library Group Descriptor and Program Format 1-11
1.4 Memory Models 1-13

2 Supplement to the System Guide

2.1 Assembly Language Interface to Driver Entry Points 2-1
2.2 Exiting an Interrupt Service Routine 2-2
2.3 POLLEVENT Caveat 2-2
2.4 System Generation Notes 2-3
2.4.1 System Generation Utilities 2-3
2.4.2 FLEX286.SYS File Format 2-3
2.4.3 FLEX286.SYS Header Record Definition 2-4
2.4.4 FlexOS Data Header 2-5
2.5 Sample Boot Loaders 2-7
2.6 Console Driver I/O Functions 2-21

3 FlexOS Front End

3.1 Running PC DOS Applications 3-1
3.1.1 PC DOS Program Memory Allocation 3-2
3.1.2 Memory Allocation -- ADDMEM 3-3
3.2 PC DOS Emulation Under FlexOS 1.3 3-6
3.2.1 PC DOS BIOS Calls 3-6

3.2.2 Software Interrupts 3-8
3.2.3 DOS Function Calls 3-8
3.2.4 Guidelines for Application Writers 3-10
3.3 Building and Installing the PC DOS front end 3-14
3.4 Known Problems 3-15

4 VDI
4.1 Introduction 4-1
4.2 Device Support 4-1
4.3 FlexOS 286 VDI Configuration and Installation 4-2
4.3.1 VDI Binding Library 4-4
4.3.2 Specifying Device Numbers, Driver Files and Fonts . 4-4
4.3.3 Parameters to vs_color 4-6
4.4 FlexOS 286 VDI Application Notes 4-7
4.5 FlexOS 286 VDI Restrictions 4-8

Figures

1-1 Target Process ID Data Structure 1-2
1-2 286 File Header Format 1-9
1-3 Group Descriptor Format 1-9
1-4 SRTL Group Format 1-11
1-5 SRTL_ID Format 1-12
1-6 XSRTL Header 1-12
1-7 FlexOS Loader Memory Models 1-13
2-1 FLEX286.SYS File 2-4
2-2 FLEX286.SYS Header Record 2-4
2-3 FlexOS Data Header 2-6

Tables

1-1 FlexOS to 80286 Software Interrupts 1-4
1-2 Reserved File Extensions 1-6
1-3 Group Descriptor Numbers 1-10
1-4 SRTL_ID Fields 1-12
2-1 FLEX286.SYS Header Record Fields 2-5
2-2 Fields in FlexOS Data Header 2-7
4-1 Graphic Device Numbers 4-5
4-2 Parameters to vs_color 4-6

Listings

1-1 Sample _osif Routine 1-8
2-1 iAPX 8086/8088/80286 Assembler Interface Convention . . 2-1
2-2 Sample Boot Loader 2-8
2-3 Sample Boot Partition Record 2-16

Supplement to the Programmer's Guide

This section describes the following 80286-specific aspects of FlexOS:

- supervisor call distinctions
- assembly-language programming tools, conventions, and interface
- program load formats
- memory models

1.1 Supervisor Call Distinctions

This section contains supplemental information for three SVC descriptions in the FlexOS Programmer's Guide:

- MALLOC
- CONTROL
- EXCEPTION

1.1.1 MALLOC

MALLOC adds heap memory in two ways: it expands an existing heap or creates a new heap. On systems that are not memory-mapped, MALLOC works only if contiguous memory is available above the existing heap.

The architecture of the 80286 forces two restrictions on MALLOC use. This does not affect MALLOC's argument definitions, values, and parameter block. The restrictions are as follows:

- You cannot obtain a data segment larger than 64K bytes with one call.
- MALLOC allocates data segments up to the nearest multiple of 64K bytes.

It is still possible to expand a data segment beyond 64K bytes by using multiple MALLOC calls and specifying particular values for the min and max fields of the Memory Parameter Block (MPB). For example, suppose that a program has a 50K byte data segment and you want to expand this by 40K bytes. The program calls MALLOC twice, specifying option 0 both times: first to allocate 14K bytes and second to get the remaining 26K. The resulting 90K byte data segment is physically contiguous.

This example assumes that you knew the exact number of bytes before the next 64K boundary. In reality, this is not practical. The solution is to set the MPB min value to 0 and the max value to the total bytes desired in each MALLOC call. Between calls, subtract the number of bytes actually allocated from the total amount requested. (Recall that MALLOC returns the amount allocated in the MPB min field.) Use the difference as the max value in your next MALLOC call. Repeat this procedure until the data segment reaches the size required by your program.

Note: MFREE is not similarly affected by 64K limits. MFREE releases the entire amount of memory allocated to the data segment after the starting address, regardless of its size.

1.1.2 CONTROL

FlexOS uses the &tpid value in your CONTROL call as a pointer to the data structure shown in Figure 1-1. The address must be in user memory space. All values are then set by FlexOS when CONTROL option 1 (LOAD) is requested. All values are Read-Only. This information is used by the debugger to connect symbols to logical addresses.

```
         0       1       2       3
     +-------+-------+-------+-------+
   0 |              pid              |
     +-------+-------+-------+-------+
   4 |            bufsize            |
     +-------+-------+-------+-------+
   8 |      cnt      |    maxcnt     |
     +-------+-------+-------+-------+
  12 |                               |
     |            buffer             |
   n +-------+-------+-------+-------+
```

Figure 1-1. Target Process ID Data Structure

The data structure's tpid fields are defined as follows:

pid	The target process's process ID
bufsize	The byte length of the Fixup Item buffer
cnt	Number of Fixup Items returned in the buffer.
maxcnt	Number of Fixup Items that exist for this load.
buffer	Sequential array of Fixup Items; each item is organized as shown below (field descriptions follow):

```
       0       1       2       3
   +-------+-------+-------+-------+
 0 |            logaddr            |
   +-------+-------+-------+-------+
 4 |             offset            |
   +-------+-------+-------+-------+
 8 |      id       |
   +-------+-------+
```

logaddr: Logical address for the specified offset

offset: Byte offset from the group base

id: Group number for this fixup where:

0 = Code group
1 = Data group
2 = Heap group
3 = Stack group

Compare the cnt and maxcnt values to ensure that your buffer contains all of the program's fixup items. If maxcnt is larger than cnt, you have not made the buffer large enough.

1.1.3 EXCEPTION Numbers

Table 1-1 shows how the exception numbers for the EXCEPTION SVC correspond to the 80286 exception vectors.

Table 1-1. FlexOS to 80286 Software Interrupts

FlexOS EXCEPTION Number	Condition	80286 Vector Number
0	Non-existent memory	12
1	Memory boundary error	9
2	Illegal instruction	6
3	Divide by zero	0
4	Bound exception	5
5	Overflow error	4
6	Privilege violation	13
7	Trace	1
8	Breakpoint	3
9	Floating-point exception	16
10	Stack fault	12
11	Emulated instruction group 0	7
12 - 18	Not used	
19 - 255	Reserved	
256+n	TRAP vector number n	0 - 255
512 - 64K	Reserved	

1.2 Assembly Language Programming Tools and Conventions

The following tools are used with FlexOS to assemble, link, and debug 80286 assembly language programs.

RASM-86™	RASM-86 (relocatable assembler) processes an 8086, 80186, or 80286 assembly language source file, and produces a machine language object file.
LINK 86™	LINK 86 combines relocatable object files into a command file that is executable under FlexOS.
LIB-86™	LIB-86 creates library files of object modules and provides a means for appending, replacing, selecting, and deleting modules in an existing library file.
XREF-86™	XREF-86 uses the RASM-86 LST and SYM files to create a cross-reference file showing symbol use in the program. Provide in .EXE form to be run under the DOS front end.
SID-286™	SID-286 is a symbolic debugger.

The FlexOS 286 Programmer's Utilities Guide describes these tools and the shared runtime library files. Note that all the tools except SID-286 are used for program development on computers with 8086, 8088, 80186, or 80286 microprocessors.

1.2.1 Reserved File Extensions

Table 1-2 lists the file extensions (also referred to as filetypes) reserved for system use under FlexOS.

Table 1-2. Reserved File Extensions

Extension	Meaning
A86	Source assembly language file
LST	Output assembly language listing with error messages
XRF	Cross-reference file showing symbols used throughout program.
SYM	Symbol file with user-defined symbols
OBJ	Object file with Intel 8086 and 80286 relocatable object code
L86	Indexed library file of commonly used object modules
INP	input file of file names and options
LIN	LINK 86 output file with line number symbols
MAP	LINK 86 output file with segment information about command file layout
286	Command file that runs directly under operating system
SRL	Shared runtime library file
BAT	Batch file
CMD	Reserved for future use
EXE	Reserved for future use
COM	Reserved for future use

1.2.2 FlexOS Entry Mechanism

Entry into FlexOS is made by application code using INT 220 as the entry point with two arguments passed in registers as shown:

Register	Contents
CX	SCV number
AX	Low order word of parameter
BX	High order word of parameter

The return code is passed back to the caller in registers as shown below:

Register	Contents
AX	Low order of return value
BX	High order of return value

The meaning and use of the arguments and the return value are described in the FlexOS Progr..mmer's Guide.

The FlexOS entry point can be encapsulated for use by C programs by the _osif function call. _osif is an assembly language routine that invokes INT 220 with the proper arguments placed by the caller on the stack. On return from INT 220, _osif in turn returns the return value in registers according to the convention used by the language processor used for the calling program. For MetaWare™ (High C, etc.) compiled programs, the default convention (changeable by "pragmas") is to pass the return value in DX:AX. Lattice uses AX:BX for returns. A sample of _osif is shown in Listing X_X. This sample can be used with C language programs compiled with the default MetaWare convention.

FlexOS can alternately be called via an SVC library of functions (such as that supplied with language processors available for FlexOS) that encapsulate not only the INT 220 instruction, but also the building of the parameter blocks required by FlexOS. Once again, these functions will return the return value in the processor's default return register convention.

Listing 1-1. Sample _osif Routine

```
;
;       FlexOS Interface for Applications
;
OSINT   EQU     220                     ; FlexOS entry point
                                        ; interrupt number
;
        cseg
        public  __OSIF
;
__OSIF:
        push    bp
        mov     bp,sp
        mov     cx,6[b]                 ; FlexOs function in CX
        mov     ax,8[bp]                ; Offset of pblk in AX
        mov     bx,10[bp]               ; Segment of pblk in BX
        int     OSINT                   ; Enter FlexOS
;
;   FlexOS returns the return code in BX:AX.
;   Place it correctly for the convention used by the caller.
;
        mov     dx,bx                   ; Return in DX:AX for High C
;       xchg    ax,bx                   ; Return in AX:BX for Lattice
        pop     bp
        retf
;
        end
;
```

1.3 Program Load Formats

FlexOS's program load format supports shared code, shared runtime libraries, and overlays. Executable command files and shared runtime libraries both consist of a file header followed by the program segments.

Executable command files are characterized by the file extension .286. The command file header is organized as illustrated in Figure 1-2.

```
OH     09H    12H    1BH    24H        48H      4AH         7DH    80H
+------+------+------+------+--        --+-------+--        --+------+
| GD1  | GD2  | GD3  | GD4  |Reserved|GDSRTL|Reserved|  FUI  |
+------+------+------+------+--        --+-------+--        --+------+
```

Figure 1-2. 286 File Header Format

The header record is always 128 bytes. The 9-byte group descriptor fields GD1 through GD4 describe the code, data, stack, and fix-up table portions of the program to be loaded. Bytes 24H to 48H are reserved and must be zero. GDSRTL is a different type of descriptor from the GDn type, whose only function is to indicate that a shared run-time library exists. Bytes 4AH to 7DH are reserved and must be zero. The FUI (fix-up information) is present only when there are fix-up items.

The code group follows immediately after the header unless the program uses a shared runtime library (SRTL). When there is a SRTL, the SRTL group follows the header.

All groups can use multiple segments.

Note: Overlays are a special class of command file that have only code and data groups. They must be created by the linker so that all code groups begin at zero paragraph addresses.

1.3.1 Code, Data, and Stack Group Descriptors

Figure 1-3 illustrates the group descriptor format.

```
OH          01H         03H         05H         07H         09H
+-----------+-----------+-----------+-----------+-----------+
|  G_TYPE   | G_LENGTH  | reserved  |   G_MIN   |   G_MAX   |
+-----------+-----------+-----------+-----------+-----------+
```

Figure 1-3. Group Descriptor Format

The **G_TYPE** field indicates the group descriptor type. Valid numbers for this field are listed in Table 1-3

Table 1-3. Group Descriptor Numbers

Number	Type
01H	Code Group
02H	Data Group
04H	Stack Group
08H	Fix-up Table
09H	Code Group (Shared Code)
FFH	Shareable run-time library (GDSRTL only)

G_LENGTH value indicates the number of paragraphs in the group. For example, a G_LENGTH of 080H indicates a size of 0800H bytes.

The **G_MIN** and **G_MAX** fields define the minimum and maximum size of the memory area to allocate to the group.

A command file must have one each of group types 1 or 9, 2, and 4. Note that FlexOS requires that programs have a stack group. In Intel xxx86 assembly language programs, declare a stack segment with an SSEG statement.

If your program does not include a stack group, use the following option and parameter when invoking LINK 86:

LINK86 testfil [stack [add [nnn]]]

where nnn is a hex number of paragraphs in memory.

Group types 8 and FFH are optional. The code group must contain relocatable 8086 or 80286 code. See Section 1.4 for a description of FlexOS's user memory models.

1.3.2 Fix-Up Information

A G_TYPE 8 descriptor is generated by the linker to describe the fix-up table. This is only present if fix-ups are required. If fix-ups are not required, there is no group type 8 and header bytes 7DH through 7FH are null.

The header does not have a fix-up group descriptor like the GDn type described above. Instead, the linker places the value 80H in byte 7FH of the header to indicate that fix-ups are needed and header bytes 7DH and 7EH are a word pointer to the fix-up table.

1.3.3 Shared Run-Time Library Group Descriptor and Program Format

FlexOS supports the use of shared run-time libraries (referred to as SRTLs). There are two components of this system:

- the shared file group descriptor in the calling program's file header, and the SRTL group itself immediately following the header
- the executable file with the SRTL code--these are referred to as XSRTLs below.

The calling program's SRTL descriptor and group values are inserted by the linker.

The indication that a shared run-time library file is used by the program is made in the GDSRTL-byte at offset 48H in the calling program's header. The value is always FFH when an XSRTL is necessary.

The SRTL group in the calling program is size-dependent upon the number of library files used and is formatted as shown in Figure 1-4.

```
0H              2H                         12H             14H
+-------+------+-------         -------+------+------+---
|   #SRTLS     |   SRTL_ID ...         |#SRTL fix-ups|  ...
+-------+------+-------         -------+------+------+---
                \                                    /
                 \                                  /
                      Repeated #SRTLS times
```

Figure 1-4. SRTL Group Format

The **#SRTLS** field indicates the number of SRTL_ID/#SRTL fix-up items to follow. (The SRTL group length is equal to (#SRTLS X 12H) + 2.)

The **#SRTL fix-ups** indicate the number of fix-ups in this run time library file.

Each **SRTL_ID** is 16 bytes long and formatted as shown in Figure 1-5.

```
0H              8H               0AH               0CH          0FH
+---        ---+-------+-------+-------+-------+----     ---+
| Name   ...   | Majorversion #| Minorversion #| Flags ...  |
+---        ---+-------+-------+-------+-------+----     ---+
```

Figure 1-5. SRTL_ID Format

Table 1-4 describes the SRTL_ID fields.

Table 1-4. SRTL_ID Fields

Field	Description
Name	The name of the file as it appears in the directory, zero-padded left to 8 bytes. Note that the XSRTL file extension is always SRL.
versions	Majorversion # and Minorversion # specify alternate SRTLs for use by the loader.
Flags	This is a reserved field; the linker uses the rightmost 4 bits and all others are zero.

The executable XSRTL file has a different program header than an executable command file. As shown in Figure 1-6, the header has group descriptors GD1 through GD4 and, at offset 60H, the file's SRTL_ID as shown in Figure 1-5 above.

```
0H    09H   12H   1BH   24H       60H       70H       7DH   80H
+-----+-----+-----+-----+--     --+-------+--     --+-----+
| GD1 | GD2 | GD3 | GD4 |   ...   |SRTL_ID|   ...   | FUI |
+-----+-----+-----+-----+--     --+-------+--     --+-----+
```

Figure 1-6. XSRTL Header

The code group follows immediately after the fix-up information.

1.4 Memory Models

The FlexOS loader creates one of two user memory models depending on whether the system is memory mapped or non-memory mapped. Figure 1-7 illustrates the two models. Note that on many memory mapped systems, the stack is automatically extended upon overflow; however, this is not true for non-memory mapped systems. Both systems allow you to expand the heap with MALLOC when contiguous memory is available above the existing heap.

```
 High
 Memory
     +--------------------+
     | Stack: preallocated|                 /\
     | for minimum        |                 ||
     +--------------------+     +---------------------+
               ||               | Heap: preallocated  |
               \/               | for maximum         |
                                +---------------------+
               /\               | Stack: preallocated |
               ||               | for maximum         |
     +--------------------+     +---------------------+
     | Heap: preallocated |     | Data: fixed at load |
     | for minimum        |     +---------------------+
     +--------------------+
                                +---------------------+
     +--------------------+     | Code: fixed at load |
     |Data: fixed at load |     +---------------------+
     +--------------------+

     +--------------------+
     |Code: fixed at load |
Low  +--------------------+
Memory

      Memory Mapped System        Non-memory Mapped System
```

Figure 1-7. FlexOS Loader Memory Models

The FlexOS loader allows programs to share code. Processes share code groups when the code is loaded from the same command file.

FlexOS supports the 80286 small, medium, compact, and large memory models. The models are defined as follows:

- Small: Separate code and data groups, each a maximum of 64K bytes long. Stack and heap are within the data group.
- Medium: Code group consists of multiple segments, but the data group is a maximum of 64K bytes long. Stack and heap are within the data group.
- Compact: The maximum size of the code group is 64K bytes, the data group consists of multiple segments, and the stack is separate, with a 64K byte limit.
- Large: Code and data groups consist of multiple segments. Stack and heap are within the data group.

End of Section 1

Supplement to the System Guide

This section describes the following:

- assembly-language interface to driver entry points
- exiting an Interrupt Service Routine (ISR)
- caveat on calling POLLEVENT driver service
- video_init Routine Explanation
- system generation notes
- sample loader code
- Console Driver I/O functions

2.1 Assembly Language Interface to Driver Entry Points

Listing 2-1 presents the iAPX 8088/80286 assembly language interface to FlexOS driver installation and I/O function entry points. The C interface is defined in Section 4.3, "Entry Point Parameter Interface," of the FlexOS System Guide.

Listing 2-1. iAPX 8086/8088/80286 Assembler Interface Convention

```
Calling Sequence:
        push    HI_WORD
        push    LO_WORD
        callf   function
        add     sp,4
        mov     LO_RET_CODE,ax
        mov     HI_RET_CODE,bx

Function Interface:
```

Listing 2-1. (continued)

```
Function:
        push    bp
        mov     bp,sp
        mov     HI_ARG,8[bp]
        mov     LO_ARG,6[bp]
        . . .
        mov     ax,LO_RET_CODE
        mov     bx,HI_RET_CODE
        pop     bp
        retf
```

2.2 Exiting an Interrupt Service Routine

A FlexOS system based on an 80286 requires that you exit an Interrupt Service Routine (ISR) by executing a far return with the appropriate true or false value contained in register AX.

ISRs must establish their own data segment registers. FlexOS sets the Code and Stack segments to be the ISR code and Global Interrupt Stack. FlexOS also preserves register contents at the time of the interrupt and restores them to these values when the ISR makes the retfar.

Detailed information on ISRs is provided in Section 5, "Driver Services," of the FlexOS System Guide.

2.3 POLLEVENT Caveat

You must preserve the register DS when you call the POLLEVENT driver service. Poll routines are described in Section 5.3, "Device Polling," of the FlexOS System Guide.

2.4 System Generation Notes

This section supplements Section 3, "System Configuration", and Section 12, "System Boot", of the FlexOS System Guide and contains information about generating and cold booting a FlexOS system.

2.4.1 System Generation Utilities

The following utilities are used to generate a FlexOS system on machines based on the 8086, 8088, and 80286 microprocessors:

- RASM-86 - Relocatable assembler
- C Compiler - Lattice™ C
- LINK 86 - Linker
- FIX-286™ - Generates an output file containing a relocated operating system image from a relocatable operating system file. Also creates the Global Descriptor Table (GDT) and Interrupt Descriptor Table (IDT) and appends them to the data segment. If you are generating a Real Mode system (indicated by the /r parameter on the FIX-286 command line), FIX-286 does not create the GDT and IDT. FIX-286 expects the OS Data Header, described in Section 2.4.4 below, to be the first item in the data segment.

The system generation utilities are fully described in the Programmer's Utilities Guide for FlexOS.

2.4.2 FLEX286.SYS File Format

Figure 2-1 shows the layout of the FLEX286.SYS file generated by FIX-286.

```
 ____________________________________
|                                    |
|           Header Record            |
|____________________________________|
|                                    |
|       Code Portion of the OS       |
|                                    |
|____________________________________|
|                                    |
|   Initialized Data Portion         |
|          of the OS                 |
|____________________________________|
```

Figure 2-1. FLEX286.SYS File

2.4.3 FLEX286.SYS Header Record Definition

The format of the FLEX286.SYS header record generated by FIX-286 is shown in Figure 2-2.

```
          0        1        2        3
      +--------+--------+--------+--------+
00H   |      Code Load Base Address       |
      +--------+--------+--------+--------+
04H   |            Code Length            |
      +--------+--------+--------+--------+
08H   |      Data Load Base Address       |
      +--------+--------+--------+--------+
0CH   |            Data Length            |
      +--------+--------+--------+--------+
      |                                   |
      .             Reserved              .
      .            (56 Words)             .
      .                                   .
80H   |                                   |
      +--------+--------+--------+--------+
```

Figure 2-2. FLEX286.SYS Header Record

Table 2-1 defines the FLEX286.SYS header record fields.

Table 2-1. FLEX286.SYS Header Record Fields

Field	Description
Code Load Base	Segment and offset into which the operating system code is to be loaded. The offset field of the address is zero.
Code Length	Length, in bytes, of the code segment.
Data Load Base	Segment and offset into which the operating system's initialized data is to be loaded. The offset field of the address is always zero.
Data Length	Length, in bytes, of the initialized data.

2.4.4 FlexOS Data Header

Figure 2-3 illustrates the FlexOS Data Header. This structure is at offset 00 in the data segment.

```
          0        1        2        3
      +--------+--------+--------+--------+
  00H |       GDT       |       GDT       | \
      |      Limit      |  Base Address   |  \  GDT
      +--------+--------+--------+--------+   > Descriptor
  04H |GDT Base|  GDT   |      Zero       |  /
      | Address| Access |                 | /
      +--------+--------+--------+--------+
  08H |    First GDT    |
      |      Entry      |
      +--------+--------+--------+--------+
  0AH |       IDT       |       IDT       | \
      |      Limit      |  Base Address   |  \  IDT
      +--------+--------+--------+--------+   > Descriptor
  0EH |IDT Base|  IDT   |      Zero       |  /
      | Address| Access |                 | /
      +--------+--------+--------+--------+
  12H |      Code Load Base Address       |
      +--------+--------+--------+--------+
  16H |            Code Length            |
      +--------+--------+--------+--------+
  1AH |      Data Load Base Address       |
      +--------+--------+--------+--------+
  1EH |            Data Length            |
      +--------+--------+--------+--------+
```

Figure 2-3. FlexOS Data Header

Table 2-2 defines the data header fields.

Table 2-2. Fields in FlexOS Data Header

Field	Description
GDT Limit	The length of the Global Descriptor Table (GDT) minus one word. FIX-286 allocates space for the GDT based on this field.
GDT Base	The linear base address of the GDT. This field is initialized by FIX-286.
GDT Access Byte	Controls access to the GDT. Initialized by FIX-286.
First GDT Entry	Offset of the next available GDT entry. This field is initialized by FIX-286.
IDT Limit	The length of the Interrupt Descriptor Table (IDT) minus one word. FIX-286 allocates space for the IDT based on this field.
IDT Base	The linear base address of the IDT. This field is initialized by FIX-286.
IDT Access Byte	Controls access to the IDT table. This field is initialized by FIX-286.

Code and data base addresses and lengths are defined in Table 2-1.

2.5 Sample Boot Loaders

Listing 2-2 is a sample loader for a PC DOS double-sided, 9 sector disk. Listing 2-3 is a sample loader for a hard disk partition record. Both loaders are for systems running FlexOS on 8086, 8088, and 80286 microprocessors.

Listing 2-2. Sample Boot Loader

```
;*============================================================*
;*  VERSION 1.2  BOOT.A86 floppy or hard disk boot code      *
;*************************************************************
;*  Sample Boot Loader                                       *
;*                                                           *
;*  This program loads the OS from a PCDOS double sided, 9  *
;*  sector disk.  The BPB variables (and extensions) are     *
;*  filled in by FORMAT.  The CLBASE, DLBASE, CODELEN, and  *
;*  DATALEN variables are filled in by the SYS program.      *
;*  This program must be in 8080 model.  The generation      *
;*  commands are:                                            *
;*                                                           *
;*      RASM86 BOOT                                          *
;*      LINK86 SYS,BOOT [CODE[ORIGIN[0]],DATA[ORIGIN[0]]     *
;*                                                           *
;*************************************************************

LJMP_OFF        equ     7c00h

LCODE       cseg para
            public  lbase
            public  clbase
            public  sector_size
lbase:
                                        ;*
                                        ;* Standard BPB
                                        ;*
        jmp startldr

            db     'OEMNAME1'
sector_size dw     512  ; Bytes per Sector
            db     2    ; Sectors per Cluster
res_sects   dw     1    ; Reserved Sectors
fats        db     2    ; Number of FATs
root_dir    dw     112  ; Root Directory Entries
            dw     720  ; Disk Size (if 0, see DRI extension below)
media_desc  db     0FDH ; Media Descriptor
fat_size    dw     2    ; FAT Size
trk_size    dw     9    ; Track Size
```

Listing 2-2. (continued)

```
num_heads    dw    2    ; Number of Heads
             dw    0    ; Hidden Sectors
                                  ;*
                                  ;* DRI extensions to BPB
                                  ;*
             dw    0    ; DRI extension to Hidden Sectors
             rd    1    ; If (WORD)disk size above is 0, this is
                        ; full (LONG).

first_data_sector    dw 12      ; 1st logical data sector filled
                                : in by FORMAT
                     dw 0       ; For large disks (not supported!)
                                ;*
                                ;* Code and Data LOAD info
                                ;*

clbase       rd    1    ; Code Load Base Address
codelen      dw    0,0  ; Code Length in bytes
dlbase       rd    1    ; Data Load Base Address
datalen      dw    0,0  ; Data length in bytes

;
;  start of loader code
;
startldr:
        cld
;
;  relocate the loader to Top of Memory from 07C0:0000
;
        int 12h          ;get amount of memory in K
        sub ax,3h        ;calculate base of where to move loader
        mov cl,6
        shl ax,cl
        mov es,ax

        xor di,di
        mov si,LJMP_OFF
        mov cx,sector_size
        shr cx,1         ;

        cli              ;disable ints to handle old 8088 bug
        mov ss,ax
        mov sp,2048
        rep movs es:ax,cs:ax   ;copy loader to top of memory
        sti
```

Listing 2-2. (continued)

```
        mov ax,offset beginload
        push es
        push ax
        retf            ;return to beginload
;
;  begin loading the OS into memory
;
beginload:
        push cs
        pop ds
        mov cl,4
        mov ax,sector_size
        shr ax,cl
        mov para_sec_size,ax
;
;  initialize the disk driver
;
        call init           ;initialize the disk driver
;
;  read the code portion of the OS in to memory
;
        mov ax,codelen      ;get low order part of the length
        mov dx,codelen+2    ;get high order part of the length
        mov si,offset no_os_mes
        mov cx,15
        mov bx,ax
        or bx,dx            ;test for zero code len
        jnz os_present
          jmp print_loop
os_present:
        mov bx,sector_size
        div bx
        or dx,dx ! jz noremainder
          inc ax
noremainder:
        les di,clbase       ;get the Code Load Base address
        mov dx,first_data_sector
        call read_block     ;parameters in ax,dx and es:di
                            ;returns the next sector to read in
                            ;ax and es:di point to the last sector
                            ;read in memory
        push ax             ;save next sector
```

Listing 2-2. (continued)

```
;
;  read the data portion of the OS in to memory
;
        mov ax,datalen     ;get low order part of the length
        mov dx,datalen+2   ;get high order part of the length
        mov bx,sector_size
        div bx
        or dx,dx ! jz noremainder2
          inc ax
noremainder2:
        les di,dlbase      ;get the Data Load Base address
        pop dx             ;restore next sector to read
        call read_block    ;parameters in ax,dx and es:di
                           ;returns the next sector to read
;
;  Start the loaded operating system
;

        mov ds,word ptr dlbase+2
        jmpf dword ptr clbase    ;jump to the OS

;************************************************************
;*    Init - Initialize any hardware necessary              *
;************************************************************
init:
        push ax
        push dx          ;hd support
        xor dx,dx        ;hd support: assume flop disk dl->0
        cmp media_desc,0f8h    ;hd support: hard or flop disk?
        jnz init1        ;hd support
        or dl,80h        ;hd support: hard disk
init1:  xor ax,ax
        int 13H          ;initialize the disk driver
        pop dx           ;hd support
        pop ax
        ret
```

Listing 2-2. (continued)

```
;***************************************************************
;*    Read Block - read consecutive sectors into memory        *
;*                                                             *
;*        input:   ax - number of sectors to read              *
;*                 dx - starting sector                        *
;*                 es:di - dma address                         *
;*        output:  ax - next sector to read                    *
;*                 es:di - dma address of last sector read     *
;*                                                             *
;*    NOTE: This boot loader uses one byte to describe the     *
;*    hard disk cylinder.  This means you cannot place the     *
;*    first record of the system image beyond cylinder 255.    *
;*    This is not a problem if you place FlexOS in the     *
;*    first partition.                                         *
;***************************************************************
read_block:
        mov num_sects,ax            ;save input parameters
        mov start_lsec,dx
        mov ax,es                   ;normalize address to nnnn:0000
        mov cl,4                    ;to nnnn:0000 for easy physical
        shr di,cl                   ;segment overflow checking
        add ax,di
        mov es,ax
        mov ax,dx                   ;compute starting head,
        xor dx,dx                   ;cylinder and sector
        div trk_size
        mov cl,dl                   ;initialize sector to read
        inc cl
        div byte ptr num_heads
        mov ch,al                   ;initialize cylinder to read
        mov dh,ah                   ;initialize head to read
        mov ax,es
read_blocka:
        test num_sects,0FFFFH
        jz read_done
          mov bx,trk_size
          sub bl,cl
          inc bl                    ;number of sectors to read
          cmp num_sects,bx
     ja dont_adjust
            mov bx,num_sects
```

Listing 2-2. (continued)

```
dont_adjust:
        push ax
        mov ax,para_sec_size
        mul bl
        mov read_paras,ax
        pop ax
        and ah,0FH              ;test for physical segment
        add ax,read_paras       ;overflow
        test ah,0F0H
        jz not_local_read
          mov ax,para_sec_size
          mov read_paras,ax
          mov bx,offset buffer_base
          push es
          push ds ! pop es
          mov al,1
          call read
          pop es              ;ES must always point to curdma
          xor di,di
          mov si,bx
          push cx
          mov cx,sector_size
          rep movsb
          pop cx
          jmps read_a_sector
not_local_read:
        mov al,bl
        xor bx,bx
        call read
read_a_sector:
        add start_lsec,ax
        sub num_sects,ax
        mov ax,es
        add ax,read_paras
        mov es,ax             ;bump the dma address by 1 sector
        jmps read_blocka
read_done:
       mov ax,es
       sub ax,para_sec_size
       mov es,ax
       mov di,bx
       mov ax,start_lsec
       ret
```

Listing 2-2. (continued)

```
;*************************************************************
;*    Read - the required sectors and perform retries as     *
;*           necessary.                                      *
;*    Return - AX = number of sectors read.                  *
;*************************************************************
read:
        mov dl,5                ;5 retries if error
read_loop:
        push ax
        push dx
        mov dl,0                ;hd support: assume flop disk
        cmp media_desc,0f8h     ;hd support: hard or flop disk?
        jnz read_1              ;hd support
        or dl,80h               ;hd support: hard disk
read_1: mov ah,02H              ;read n sector(s)
        int 13h
        pop dx
        pop ax                  ;restore sector count
        jnc no_read_error
          dec dl
          jz read_error
            call init           ;force recal for retry
            jmps read_loop

no_read_error:
        xor ah,ah
        add cl,al
        cmp cl,byte ptr trk_size
        jbe read_exit           ;no track overflow
          mov cl,1
          inc dh
          cmp dh,byte ptr num_heads
          jb read_exit          ;no head overflow
            mov dh,0
            inc ch
read_exit:
        ret

read_error:
        mov si,offset error_string
        mov cx,14
```

Listing 2-2. (continued)

```
print_loop:
        lodsb
        mov ah,14
        int 10H
        loop print_loop
hlt_loop:
        hlt
        jmps hlt_loop

codeend      rb 0
        org  (((offset codeend - offset lbase) + 0fh) and 0fff0h)
dbase        rb 0

error_string db      'Disk I/O Error'
no_os_mes    db      'Non-System disk'

filler       db 0       ;place signature at sector end - 2

signature    dw 0AA55h  ;generic IBM signature

data_end                rw      0

para_sec_size           equ     data_end+0
read_paras              equ     data_end+2
start_lsec              equ     data_end+4
num_sects               equ     data_end+6

buffer_base             equ     data_end+8

        end
```

Listing 2-3. Sample Boot Partition Record

```
;*============================================================*
;*  VERSION 1.3 HDBOOT.A86  Master boot partition record     *
;*************************************************************
;*  Sample Boot Partition Record                              *
;*                                                            *
;*  This FlexOS master boot strap loads and passes            *
;*  control to any bootable DOS partition.                    *
;*                                                            *
;*  The master boot strap searches its internal partition     *
;*  table for a bootable partition entry.  If it locates      *
;*  a single bootable entry, the referenced sector is read    *
;*  into memory at 0000:7C00H.  The loaded sector must        *
;*  contain a valid signature (AA55h) at offset address       *
;*  1FEh from the load address.  Program control is passed    *
;*  to the loaded sector with a pointer (DS:SI) to the        *
;*  original partition entry from which it was loaded.        *
;*                                                            *
;*  The generation command is:                                *
;*                                                            *
;*       RASM86 HDBOOT                                        *
;*                                                            *
;*************************************************************
        public hdboot
;       ROM BIOS CONSTANTS
FDSK_INT        equ     13h             ;rom disk I/O entry
BASIC_INT       equ     18h             ;rom basic entry
VIDEO_INT       equ     10h             ;

DSK_RESETF      equ     0               ;disk reset command function
DSK_READF       equ     2               ;disk read command function
VID_COUTF       equ     14              ;teletype char out function

;       PARTITION TABLE INDEXES
PT_BID_HD       equ     0               ;boot indicator / head
PT_SEC_CYL      equ     2               ;sector / cylinder
```

Listing 2-3. (continued)

```
;       MISC. PROGRAM CONSTANTS
LOAD_OFF        equ     00h             ;MBL offset address
LOAD_SEG        equ     7c0h            ;MBL segment address
LJMP_OFF        equ     7c00h           ;MBL jump offset
LJMP_SEG        equ     00h             ;MBL jump segment
RUN_OFF         equ     00h             ;MBL run offset address
RUN_SEG         equ     60h             ;MBL run segment address

eject
; Loaded by the ROM boot strap to address 0000:7C00h
;
HDBCODE         CSEG    para
;
hdboot:
        cli
        mov     ax,LOAD_SEG
        mov     ss,ax
        mov     ax,LOAD_OFF
        mov     sp,ax
        sti

; Move master boot so that the DOS boot sector can be read
: to 0000:7C00h
;
        cld
        mov     ax,LOAD_SEG
        mov     ds,ax
        mov     si,LOAD_OFF
        mov     ax,RUN_SEG
        mov     es,ax
        mov     di,RUN_OFF
        mov     cx,200h/2
        rep     movsw
                                        ;jmpf  msboot_exec  ;ms_boot:
        db      0eah
        dw      offset ms_boot
        dw      RUN_SEG
```

Listing 2-3. (continued)

```
; Find a valid boot partition
;
ms_boot:
        mov     ax,cs
        mov     ds,ax                   ;NOTE: DS = CS !!!
        mov     cx,4                    ;# of partitions to search
        mov     di,offset st_part_tbl    ;partition 1

        find_boot_part:
        mov     dx,PT_BID_HD[di]        ;DL=boot indicator DH=DOS boot HEAD#
        cmp     dl,80h                  ;boot this partition?
        jz      found_boot_part         ;if yes
        cmp     dl,0                    ;valid partition if 0
        jnz     invalid_part_err        ;else partition table garbaged
        add     di,16
        loop    find_boot_part          ;check next partition

        int     BASIC_INT               ;ERROR no bootable partitions found

; Make sure all partitions are valid
;
found_boot_part:
        mov     si,di                   ;save pointer to bootable partition
        jmps    chk_next_part

chk_boot_part:
        add     di,16
        mov     ax,PT_BID_HD[di]
        cmp     al,80h
        jz      invalid_part_err        ;error if >1 bootable partition
        cmp     al,0
        jnz     invalid_part_err        ;error if garbage

chk_next_part:
        loop    chk_boot_part


; Read DOS boot strap
; (DX = head,drive of DOS boot SI -> boot partition)
;
        mov     cx,5                    ;# of retries on a error
```

Listing 2-3. (continued)

```
read_dos_boot1:
        mov     ax,LOAD_SEG
        mov     es,ax
        mov     bx,LOAD_OFF             ;DMA
        mov     ah,DSK_READF            ;command
        mov     al,1                    ;always 1 sector
        push    cx
        mov     cx,PT_SEC_CYL[si]       ;CH=l.s. 8 bits of cylinder
        int     FDSK_INT                ;CL=m.s. 2 bits of cyl. 6 bits
        pop     cx                      :sector
        jnc     goto_dos_boot           ;no error

        mov     ah,DSK_RESETF           ;reset
        int     FDSK_INT
        loop    read_dos_boot1          ;retry read

        mov     si,offset rd_err_msg ;ERROR in loading operating system
        jmps    msboot_error

                                        ;jmpf     dos_boot
; Pass control to DOS boot strap if found valid
; (SI -> boot partition table)
;
goto_dos_boot:
;       cmp     es:signature[bx],0AA55h ;valid signature
;       jnz     no_dos_err

        db      0eah
        dw      LJMP_OFF
        dw      LJMP_SEG

no_dos_err:
        mov     si,offset no_dos_msg ;ERROR no OS boot
        jmps    msboot_error

invalid_part_err:                       ;ERROR invalid partition
        mov     si,offset inval_part_msg
```

Listing 2-3. (continued)

```
; Print error message and loop
; (DS:SI -> error msg '$'delimiter)
;
msboot_error:
        lodsb
        cmp     al,'$'

msboot_err1:
        jz      msboot_err1             ;loop until system reset

        push    si
        mov     bx,7
        mov     ah,VID_COUTF
        int     VIDEO_INT               ;teletype console output
        pop     si
        jmps    msboot_error

bootend         rb 0
eject
;
; ERROR MESSAGES
;
mstart          rb      0
inval_part_msg  db      'Invalid partition table$'
no_dos_msg      db      'Missing operating system$'
rd_err_msg      db      'Error loading operating system$'
msend           rb      0
;
; PARTITION TABLE MUST START AT OFFSET ADDRESS 01BEh
;
                org   01beh
;
;                                                       Partition
st_part_tbl     db   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ;    1
                db   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ;    2
                db   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ;    3
                db   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ;    4

signature       dw   0AA55h         ;generic IBM signature
;
                end
```

2.6 Console Driver I/O Functions

This information supplements the description of the Console Driver I/O functions in Section 7 of the FlexOS System Guide.

The video_init routine in the model console driver initializes video displays. It is used in three places to make changes to the current display mode.

- It is called by the c_init function to initialize the video controller to the default display mode.
- It is called by Console driver's SPECIAL function 4 to reinitialize the video controller when a different display mode is to take the top position on the screen.
- It is called by the Console Resource Manager when VFRAMEs are reordered and the VFRAME moving to the top does not match the previous.

The video modes are tracked by the driver as a BYTE code. The table below lists the assignment for values 0 through 7.

You may add new modes to video2up(_)init to enhance displays (a 1024 x 1024 color graphics mode, for example). However, mode values 0 through 20 are reserved. The range of new mode values is 21 through 255.

Value	Description
0	40 x 25 B&W Character
1	40 x 25 Color Character
2	80 x 25 B&W Character
3	80 x 25 Color Character
4	320 x 200 Color Graphics
5	320 x 200 B&W Graphics
6	640 x 200 B&W Graphics
7	80 x 25 B&W Monochrome Card
8	720 x 348 B&W Hercules Card
9-13	Reserved
14	640 x 200 16 Color EGA Card
15-128	Reserved

End of Section 2

FlexOS Front End

The FlexOS front end allows certified PC DOS version 1.0 and 2.x applications to run in the protected environment of a multitasking operating system. This section provides guidelines for running the certified applications and for writing new applications using PC DOS functions to run under the FlexOS 1.3 front end.

Important: The PC DOS front end works only with the E2 version of the iAPX 286.

3.1 Running PC DOS Applications

The indicated versions of the following PC DOS applications have been verified by Digital Research® to run under FlexOS's front end.

Application	Version	Manufacturer
1-2-3™	1A	Lotus™ Development Corp.
MultiMate™	3.30, 3.31	MultiMate™ International
Level II Cobol™	2.1	Micro Focus™ Inc.
R:base 5000™		Microrim™, Inc.
dBase III™	1.0*	Ashton-Tate

*The tested version of dBase III was not copy protected. Because unprotected versions are not generally available from a retail store, contact Ashton-Tate to acquire it.

The PC DOS front end does not change or replace FlexOS's native interface. The command line and commands operate exactly as described in the FlexOS User's Guide. Although many FlexOS and PC DOS utilities have the same names, you must use the utilities provided with FlexOS.

Refer to the documentation that accompanies the application for instructions on its use. Only FlexOS-specific notes are provided here.

3.1.1 PC DOS Program Memory Allocation

PC DOS is a single-tasking operating system. Because only one task is intended to run at a time, DOS allows a single program to occupy all of user memory. There is no need to share memory because there is never another program with which to share it.

FlexOS is a multitasking operating system in which several programs are expected to coexist in the system. Therefore, memory must be carefully allocated so that as many programs as possible can run at the same time (see the description of the ADDMEM feature, below).

There are two types of executable PC DOS files, those with an extension of EXE and those with an extension of COM. EXE files provide FlexOS some loading information in a file header. This information indicates the size of the EXE file, the size of the header, the minimum amount of memory space the programmer wanted this file to own in addition to the code size, and the maximum additional amount to allocate. COM files have no header and provide no information to FlexOS regarding the program's memory requirements.

FlexOS's loader uses the following characteristics to allocate memory to EXE-type programs:

```
  code size from the header
+ minimum extra from the header
+ 16K automatic allocation by FlexOS
+ user-defined ADDMEM
= EXE load module size
```

FlexOS's loader uses the following characteristics to allocate memory to COM-type programs:

```
  size of .COM file
+ 20K automatic allocation FlexOS
+ user-defined ADDMEM
= COM load module size
```

3.1.2 Memory Allocation -- ADDMEM

The ADDMEM feature of FlexOS's DEFINE command allows you to control the amount of memory allocated to each PC DOS program when the program is loaded. FlexOS applies the memory allocation on a process family (FID) basis. An ADDMEM memory allocation applies only to programs loaded under the virtual console on which the DEFINE command was invoked. Regular users of PC DOS applications should include a DEFINE ADDMEM command in their AUTOEXEC.BAT file.

The DEFINE command used to add memory has the following form:

```
DEFINE ADDMEM = n
```

where n specifies the amount of memory in kilobytes.

Programs you expect to use with large amounts of data (such as large spreadsheets) will need extra memory space for the data. Programs that load and execute other programs will also need ample space in excess of their own code size. In both cases, define a large ADDMEM value.

The applications described below (under "Application Guidelines") as able to run multiple copies do so on systems supplied with sufficient RAM in which to load those copies. FlexOS displays:

```
SHELL: Load Error
```

to indicate that the available RAM in your system is less than the load module size as computed above. If a large ADDMEM value has been defined, decrease it. The following message indicates the program was not allocated enough memory in which to load:

```
SHELL:  Memory bound exception
```

To solve the error in this case, increase the ADDMEM value.

Logical Drive B

On systems with one floppy drive, PC DOS and several PC DOS applications assume that there is a second, logical drive, drive B. On these systems, you should use FlexOS's ASSIGN command to direct requests for drive B to drive A. The ASSIGN command line used to direct requests for drive B to drive A is:

```
A>ASSIGN B=A
```

See the FlexOS User's Guide for a complete description of the ASSIGN command.

Applications Guidelines

Use the following guidelines when running the tested applications.

1-2-3 Version 1A

Recommended ADDMEM setting:	128 for LOTUS.COM none required for 123.EXE
Run multiple copies?	yes

Special Notes:

- When starting 1-2-3 from a hard disk, the 1-2-3 floppy disk must be in drive A. This is a requirement of 1-2-3.
- Although the Lotus Access Manager will run without ADDMEM set, it needs extra memory to load 1-2-3. Be sure to set ADDMEM.
- To load a large spreadsheet, increase the ADDMEM setting as needed.
- Graphics are supported on an IBM Color Graphics Adapter, allowing graphs to be generated and displayed.
- PrintGraph may be used on either a Monochrome or Color Card, but may not be run from a serial terminal.
- If you run 1-2-3 on a single-floppy drive system, remember to ASSIGN all requests for drive B to drive A.
- The Lotus disk selection menus display consecutive drive letters up to the alphabetically greatest drive name defined for the system. For example, on a system with floppy drive A and hard disk drive D, the menus display choices for drives A, B, C, and D.
- 1-2-3 version 2 does not run on FlexOS.

RBase: 5000

Recommended ADDMEM setting:	64
Run multiple copies?	no

Special Notes:

- FlexOS looks in the current directory for the file RBASE.DAT when RBASE is initialized. If the file is not in that directory, it looks on the system: disk. If system: is defined as a floppy, be sure you have a disk in the drive.
- You cannot invoke applications from the RB5000 menu. Invoke the applications from the command-line prompt rather than from the menu.

Level II Cobol Version 2.1

Recommended ADDMEM setting:	64
Run multiple copies?	yes

Special Note: Multiple copies of the Cobol compiler cannot access the same source program simultaneously.

MultiMate Version 3.30 and 3.31

Recommended ADDMEM setting:	64
Run multiple copies?	yes

Special Note: You need a minimum of 1152K RAM to run two copies of MultiMate.

dBase III Version 1.0

Recommended ADDMEM setting:	64
Run multiple copies?	yes

Special Notes:

- FlexOS looks in the current directory for the file CONFIG.DB when dBase III is initialized. If the file is not in that directory, it looks on the system: disk. If system: is defined as a floppy, be sure you have a disk in the drive.
- Multiple copies of dBase III cannot edit the same database.
- The copy protected version of dBase III does not run.
- The dBase III Command Assistant directory menus display consecutive drive letters up to the alphabetically greatest drive name defined for the system. For example, on a system with floppy drive A and hard disk drive D, the menus display choices for drives A, B, C, and D.

3.2 PC DOS Emulation Under FlexOS 1.3

This section lists the PC DOS BIOS calls and software interrupts supported by FlexOS. Digital Research has validated FlexOS's support of these functions by testing the certified applications. However, no claim is made about the exact equivalence of each function and its PC DOS counterpart, including those functions described as "supported."

Note: You cannot mix PC DOS with native FlexOS calls (SVCs) in the same program.

3.2.1 PC DOS BIOS Calls

Int 10H Subfunctions

0H:	Supported. Can also set mode by writing to CRT controller.
1H:	Supported. Sets no cursor if no cursor or a bad cursor value specified, else sets blinking line cursor.
2H:	Supported. Display page is ignored.
3H:	Supported. Display page is ignored.
4H:	Not supported.
5H:	Not supported.
6H:	Supported.
7H:	Supported.

8H:	Supported. Display page is ignored.
9H:	Supported. Display page is ignored.
0AH:	Supported. Display page is ignored.
0BH:	Not supported.
0CH:	Not supported.
0DH:	Not supported.
0EH:	Not supported.
0FH:	Supported. Active display page is always zero.
13H:	(AT only) Not supported.

Int 11H: Supported. Returns 287 presence and initial video mode.
Returns following static values:
ipl present
one floppy drive
high byte always 0.

Int 12H: Supported. Returns memory allocated to current process.

Int 13H Subfunctions

00H:	Supported.
01H:	Supported.
02H:	Supported.
03H:	Supported with restrictions to maintain system integrity.
04H:	Supported.
05H:	Not supported.

Int 14H: Not supported.

Int 15H: Not supported.

Int 16H Subfunctions

00H:	Supported.
01H:	Supported.
02H:	Returns zero.

Int 17H: Supported, but not fully tested.

3.2.2 Software Interrupts

Int 20H:	Supported.
Int 22H:	Supported.
Int 23H:	Supported.
Int 24H:	Supported.
Int 25H:	Supported.
Int 26H:	Supported with restrictions to maintain system integrity
Int 27H:	Not supported

3.2.3 DOS Function Calls

Int 21H Subfunctions

00H:	Supported.
01H:	Supported.
02H:	Supported.
03H:	Supported.
04H:	Supported.
05H:	Supported.
06H:	Supported.
07H:	Supported.
08H:	Supported.
09H:	Supported.
0AH:	Supported.
0BH:	Supported.
0CH:	Supported.
0DH:	Not supported.
0EH:	Supported.
0FH:	Supported.
10H:	Supported.
11H:	Supported.
12H:	Supported.
13H:	Supported.
14H:	Supported.
15H:	Supported.
16H:	Supported, cannot create read-only files
17H:	Supported.
18H:	Returns zero.
19H:	Supported.

1AH:	Supported.
1BH:	Supported.
1CH:	Supported.
1DH:	Returns zero.
1EH:	Returns zero.
1FH:	Not supported.
20H:	Returns zero.
21H:	Supported.
22H:	Supported.
23H:	Supported.
24H:	Supported.
25H:	Supported.
26H:	Supported.
27H:	Supported.
28H:	Supported.
29H:	Supported.
2AH:	Supported.
2BH:	Supported.
2CH:	Supported.
2DH:	Supported.
2EH:	Not supported.
2FH:	Supported.
30H:	Supported.
31H:	Not supported.
32H:	Not supported.
33H:	Set has no effect; Get returns TRUE.
34H:	Not supported.
35H:	Supported.
36H:	Supported.
37H:	Set has no effect; Get returns constant values.
38H:	Not supported.
39H:	Supported.
3AH:	Supported.
3BH:	Supported, defines default: at the process level.
3CH:	Supported, cannot create read-only files.
3DH:	Supported.
3EH:	Supported.
3FH:	Supported.
40H:	Supported.
41H:	Supported.

42H:	Supported.
43H:	Supported.
44H:	Subfunctions 0, 1, 6, 7 supported.
45H:	Supported.
46H:	Supported.
47H:	Supported.
48H:	Effective if function 4Ah is first called to shrink memory allocation by the size desired.
49H:	Supported.
4AH:	Supported.
4BH:	Not supported.
4CH:	Supported.
4DH:	Returns zero. Processes continue asynchronously.
4EH:	Supported.
4FH:	Supported.
50H:	Supported.
51H:	Supported.
52H:	Not supported.
53H:	Not supported.
54H:	Returns zero.
55H:	Supported.
56H:	Supported.
57H:	Supported.

3.2.4 Guidelines for Application Writers

The PC DOS emulator executes a PC DOS application in the 80286 protected mode. In this mode, an application is restricted from the execution of some machine instructions and is restricted from accessing any memory that is not assigned to it initially. Violation of these restrictions results in a protection exception. In addition, there are some minor instruction differences between the 8088/8086 and the 80286 that may be important to some programs.

To run under FlexOS, PC DOS applications must behave according to the following rules.

The program must:

- Use PC DOS function calls to perform all system functions and data I/O.
- Use the PC DOS system call to obtain additional memory.
- Limit direct BIOS calls (interrupt request to the BIOS) to the following functions:

  10H
  11H
  12H
  13H
  16H
  17H

The program must not:

- Use instructions that are restricted to 80286 privileged mode. These instructions are:
  - IN, INS, OUT, and OUTS except to the CRT controller (the emulator ignores all other of these instructions)
  - CLI and STI (the emulator ignores these instructions)
  - LOCK (causes program termination)
  - HLT (causes program termination)
  - all of the 80286 unique protection control instructions (cause program termination)
- Jump to or call BIOS routines directly.
- Address PC DOS flags, data buffers, tables, work areas, etc.
- Address any memory that has not been assigned to the application at load time.

- Set the video display mode any way other than with BIOS Int 10H, subfunction 0 or by accessing the CRT controller with an OUT instruction.
- Defile the PC DOS Reserved areas of the PSP and FCB.

Generally, you should not access absolute addresses in memory directly. The only exceptions are the following virtual addresses:

- screen region buffers B0000-B3FFF (mono) and B8000-BBFFF (color).
- zero page 0-5FF; however, do not depend on the values in 400-5FF.

For better program performance, avoid instructions that load segment registers. These include the following instructions:

```
LDS
LES
POP sr
MOV sr, memreg
CALLF
RETF
JMPF
```

In addition, small model programs perform better than programs written in the other models.

There are several operating differences between the 8088/8086 and the 80286. Some of these differences are handled by FlexOS; others are not. For more information on the differences between the 8088/8086 and the 80286, see the Intel iAPX 286 Operating System Writer's Guide and the Intel iAPX 286 Programmer's Reference Manual. The following differences may affect program performance:

- Most instructions take fewer clock cycles on the 80286 than they do on the 8088/8086.
- PUSH SP in the 80286 puts the value of SP from before the instruction was executed onto the stack. POP SP works accordingly.
- 80286 masks all shift/rotate counts to the low 5 bits (maximum 31 bits).
- Do not duplicate prefixes. (80286 sets an instruction length limit to 10 bytes.)
- Do not rely on IDIV exceptions for quotients of 80H or 8000H.
- Instructions or data items may not wrap around a segment.
- Do not attempt to change the sense of any reserved or unused bits in the flag word via IRET.
- Floating point exceptions appear as interrupt 2.
- Divide exceptions point at the DIV instruction.

3.3 Building and Installing the PC DOS front end

The PC DOS front end is linked into the bootable systems that have been provided. The following front end files are included in the Developer Kit Supplement:

```
BOOTFE  .INP    ATFELNK .INP    FELIB   .L86
```

The following files are included in the OEM Redistribution Kit:

```
MKFELIB .INP    COMFE   .OBJ    COMFEU  .OBJ
CRMB000 .OBJ    DOSFE1  .OBJ    DOSFE2  .OBJ
DOSFEINT.OBJ    DOSINIT .OBJ    DOSMAIN .OBJ
DOSMAN  .OBJ    EM86    .OBJ    FEIO    .OBJ
FEU     .OBJ    GP      .OBJ    LEXECOM .OBJ
```

Perform the following procedures to generate a FlexOS system with the PC DOS front end:

1. Edit the ICONF286.C file and make sure that there are no comment delimiters around the statement:

   ```
   #define DOSFE
   ```

   This statement is at the beginning of the file.

2. Compile the file.

3. Link the system. To create a non-bootable system, use the ATFELNK.INP input file. To create a bootable system, use the BOOTFE.INP input file.

3.4 Known Problems

This section contains a list of known problems in the PC DOS Front End as of the release of this document. The list is arranged in the order of severity where the severity levels are defined as follows:

- Severity 1: Causes the system to halt indefinitely and/or the loss of data integrity.
- Severity 2: Can cause system failure, but the impact is less severe and can generally be avoided.
- Severity 3: Does not cause system failure in most cases and has less impact than severity 2 problems.

There are no known Severity 1 problems in this release of the PC DOS front end.

<u>Severity 2 Problem</u>

MultiMate	Backspace key is not operable on Document: line.

<u>Severity 3 Problems</u>

MultiMate	Lower righthand arrows indicating Scroll and Numlock status always point down. Automatic underline indicator in same location is absent.
R:base 5000	RCOMPILE menu border and one GATEWAY screen draw incorrectly. The RBEDIT status line occasionally wraps around to second line. The first character of the RBEDIT screen display is blanked when the cursor returns to it. One additional scroll occurs per screen of directory display -- this causes one line of the display to be lost.

General Emulation Problems

The following list of general DOS emulator problems is provided for those who wish to enhance FlexOS's present level of DOS 2.1 emulation. This list is included as an aid to product planning. It is not intended to be a complete list.

- An application that performs two null segment loads with an access, but no non-null segment load in between, is terminated. This problem affects 1-2-3 (Version 2.0) and Personal Editor. It may also effect other applications that have not be tested by Digital Research.
- Applications that write directly to the video map are not supported on a serial terminal.

- Environmental strings are not supported.
- FlexOS does not support the Load or Execute a Program (4BH) function.
- Hardware-specific coding could provide a higher level of Interrupt 11H (Equipment Determination) implementation. See Section 3.2.1, "PC DOS BIOS Calls".

End of Section 3

VDI

4.1 Introduction

The FlexOS 286 Virtual Device Interface (VDI) provides a standardized, device-independent graphics extension to the FlexOS 286 Supervisor Calls (SVCs). The implementation and use of the VDI is documented in the GEM™ Virtual Device Interface Reference Guide. This section identifies the differences between the VDI documentation in that guide and the VDI available on FlexOS 286.

4.2 Device Support

The FlexOS VDI supports the following VDI devices:

Display	IBM PC Color Display or compatible monitor with an IBM Color Graphics Adapter in Black and White 640 x 200 pixel resolution. The screen driver CGASCR1.SYS is included in the VDIDRVR subdirectory in the FlexOS 286 system disk set.
	IBM PC Color Display or compatible monitor with an IBM Enhanced Graphics Adapter or compatible with 16 colors and 640 x 200 pixel resolution. The screen driver EGASCR1.SYS is included in the VDIDRVR subdirectory in the FlexOS 286 system disk set.
Mouse	The Mouse Systems™ mouse and SummaGraphics® SummaMouse™ are supported via the FlexOS 286 Console and Port drivers that are always included with the FlexOS 286 system.
Printer	IBM Graphics Printer MX-80 or compatible. Two drivers, MX80PRL1.SYS (low-resolution mode) and MX80PRH1.SYS (high-resolution mode), are included in the VDIDRVR subdirectory in the FlexOS 286 system disk set.

Metafile	To support this logical device, the METAFIL1.SYS is included in the VDIDRVR subdirectory in the FlexOS 286 system disk set.

4.3 FlexOS 286 VDI Configuration and Installation

This section lists the FlexOS 286 VDI Components and describes the VDI installation procedures.

Note: You must have release 1.3 of the system to use the VDI described in this book.

The following FlexOS 286 VDI software is included in the FlexOS 286 system disk set.

```
IBMLSS10.FNT     IBMLSS14.FNT     IBMLSS18.FNT
IBMLSS36.FNT     IBMLTR10.FNT     IBMLTR14.FNT
IBMLTR18.FNT     IBMLTR36.FNT     IBMHSS07.FNT
IBMHSS10.FNT     IBMHSS14.FNT     IBMHSS18.FNT
IBMHSS36.FNT     IBMHTR07.FNT     IBMHTR10.FNT
IBMHTS14.FNT     IBMHTR18.FNT     IBMHTR36.FNT
EPSLSS10.FNT     EPSLSS14.FNT     EPSLSS20.FNT
EPSLSS28.FNT     EPSLSS36.FNT     EPSLTR10.FNT
EPSLTR14.FNT     EPSLTR20.FNT     EPSLTR28.FNT
EPSLTR36.FNT     EPSHSS07.FNT     EPSHSS10.FNT
EPSHSS14.FNT     EPSHSS20.FNT     EPSHSS28.FNT
EPSHSS36.FNT     EPSHTR07.FNT     EPSHTR10.FNT
EPSHTR14.FNT     EPSHTR20.FNT     EPSHTR28.FNT
EPSHTR36.FNT     CGASCR1 .SYS     EGASCR1 .SYS
MX80PRL1.SYS     MX80PRH1.SYS     METAFIL1.SYS
ASSIGN  .SYS
```

Some other files needed in the system are:

```
BOOTGIF .INP     MKGIFLIB.INP     ATGIFLNK.INP
VDIBINDB.L86     GIFLIB  .L86     VDIBIND .H
VDILIB  .H       CRMGSX  .C       MISGSX  .C
GMAN    .C       GMANL   .C       VDI     .H
GMAN    .H
```

all sources for the VDI drivers

Perform the following procedures to generate a FlexOS 286 system that includes the VDI:

1. Edit the CONFIG.H file to ensure that the graphics resource manager is included by the statement
2. #define GIF
3. Compile CONFIG.C.
4. Edit IPDRV.H to set port 0's default baud rate to 1200 baud for the mouse.
5. The (#define PT0_BAUD 7) sets the baud rate for port 0 to 1200.
6. Compile IPDRV.C.
7. Edit ICDRV.H to set console 0's default configuration to own a mouse device.
8.
```
#define CT0W_FLGS  AMOUSE + NUMPD   A mouse + numberpad are supported.
#define CT0_BUTS   3               The mouse has three buttons.
#define CT0_MR     2               2 Mickeys/per pixel on rows.
#define CT0_MC     1               1 Mickeys/per pixel on cols.
```
9. Compile ICDRV.C.
10. Make the ATLIB.L86 with the input file MKATLIB.INP.
11. Make sure a GIFLIB.L86 is in the current directory; if not create it with the input file MKGIFLIB.INP.
12. Link the operating system using the input file ATGIFLNK.INP (for a debuggable system) or BOOTGIF.INP (for a bootable system).
13. Create a directory VDIDRVR on the SYSTEM: device. Place the VDI drivers (CGASCR1.SYS, EGASCR1.SYS, METAFIL1.SYS, MX80PRL1.SYS, and MX80PRH1.SYS), font files (*.FNT), and ASSIGN.SYS in the directory VDIDRVR.
14. Define VDISYS: = SYSTEM:VDIDRVR/ to let the system know where to find ASSIGN.SYS, the VDI drivers, and font files.
15. Edit CONFIG.BAT to add the following commands to load the drivers at boot time. Note that REM is used as a comment.

```
rem     Load the EGA VDI screen driver
dvrload vdi01: system:/vdidrvr/egascrl.sys n
rem
rem     Load the CGA VDI screen driver
rem dvrload vdi01: system:/vdidrvr/cgascrl.sys n
rem
rem     Load the EPSON VDI Lo Res printer driver
dvrload vdi21: system:/vdidrvr/mx80prll.sys n
rem
rem     Load the EPSON VDI Hi Res printer driver
rem dvrload vdi21: system:/vdidrvr/mx80prhl.sys n
rem
rem     Load the VDI metafile driver
dvrload vdi31: system:/vdidrvr/metafill.sys n
rem
```

16. Boot FlexOS 286 watching the CONFIG.BAT file execution to ensure that the driver are loaded correctly. Then try executing some one of the demo programs provided.

4.3.1 VDI Binding Library

The bindings library VDIBINDB.L86 is provided with the system for FlexOS 286 applications making VDI calls.

You may now use DOLNK.BAT to link an application with the VDI library.

4.3.2 Specifying Device Numbers, Driver Files and Fonts

A FlexOS 286 system configured with VDI reads the ASCII file ASSIGN.SYS upon receiving an Open Workstation call from an application program. It searches for a device number which matches the requested device from the Open Workstation call. Once found, the corresponding driver name and font names are read into a table for future use by the application. You can edit the ASSIGN.SYS file to change the names of the drivers and fonts. Use a semicolon (;) to delimit comments.

The following table lists the ranges of device numbers that are associated with different types of VDI devices in ASSIGN.SYS.

Table 4-1. Graphic Device Numbers

Device	Numbers
Screen	01-10
Plotter	11-20
Printer	21-30
Metafile	31-40
Camera	41-50
Tablet	51-60

The ASSIGN.SYS file contains the following lines to assign a device number and fonts to the FlexOS 286 VDI drivers.

```
01 EGASCR1.SYS  ; IBM Enhanced Card / Color Display (640x200) 16 color
IBMLSS10.FNT    ; IBM 640 x 200 Swiss 10 Point
IBMLSS14.FNT    ; IBM 640 x 200 Swiss 14 Point
IBMLSS18.FNT    ; IBM 640 x 200 Swiss 18 Point
IBMLSS36.FNT    ; IBM 640 x 200 Swiss 36 Point
IBMLTR10.FNT    ; IBM 640 x 200 Dutch 10 Point
IBMLTS14.FNT    ; IBM 640 x 200 Dutch 14 Point
IBMLTR18.FNT    ; IBM 640 x 200 Dutch 18 Point
IBMLTR36.FNT    ; IBM 640 x 200 Dutch 36 Point
21 MX80PRL1.SYS ;IBM/EPSON Graphics Pinters Lo Res mode (60x72 dots/in.)
EPSLSS07.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 07 Point
EPSLSS10.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 10 Point
EPSLSS14.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 14 Point
EPSLSS20.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 20 Point
EPSLSS28.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 28 Point
EPSLSS36.FNT    ; EPSON Lo Res (60x72 dots/inch) Swiss 36 Point
EPSLTR07.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 07 Point
EPSLTR10.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 10 Point
EPSLTR14.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 14 Point
EPSLTR20.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 20 Point
EPSLTR28.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 28 Point
EPSLTR36.FNT    ; EPSON Lo Res (60x72 dots/inch) Dutch 36 Point
31 METAFIL1.SYS ; Metafile driver.
```

4.3.3 Parameters to vs_color

Note that when "500" is used below, it refers to any value greater than zero and less than or equal to 500. Similarly, "1000" means any number greater than 500.

Table 4-2. Parameters to vs_color

R	G	B	Color
0	0	0	Black
0	0	500	Dark Blue
0	0	1000	Light Blue
0	500	0	Dark Green
0	500	500	Dark Cyan
0	500	1000	Light Blue
0	1000	0	Light Green
0	1000	500	Light Green
0	1000	1000	Light Cyan
500	0	0	Dark Red
500	0	500	Dark Magenta
500	0	1000	Dark Magenta
500	500	0	Dark Yellow
500	500	500	Gray
500	500	1000	Dark White
500	1000	0	Dark Yellow
500	1000	500	Dark White
500	1000	1000	Light Cyan
1000	0	0	Light Red
1000	0	500	Light Red
1000	0	1000	Light Magenta
1000	500	0	Dark Yellow
1000	500	500	Dark White
1000	500	1000	Light Magenta
1000	1000	0	Light Yellow
1000	1000	500	Light Yellow
1000	1000	1000	White

4.4 FlexOS 286 VDI Application Notes

The GEM Virtual Device Reference Guide should be consulted in order to find out how to write VDI applications. FlexOS 286 VDI applications are almost identical to those written to run on PC DOS. There are three differences:

1. On FlexOS 286, a call to the s_open FlexOS SVC should be added just prior to the Open Workstation call as follows:

   ```
   gdvr = s_open(0x50,"vdi01:");
   ```

   ```
   where: gdvr   - LONG driver number returned by s_open
          0x50   - requested permissions for s_open
          vdi01: - device to be opened (typically loaded by a
                   DVRLOAD command in CONFIG.BAT) in this case
                   a screen driver.
   ```

2. Similarly, After the Close Workstation call, the following calls to FlexOS SVCs should be added:

   ```
   s_close(0,gdvr);
   s_exit(0L);
   ```

   This closes the device opened earlier and terminates the program normally.

3. The application is compiled, then linked with the VDI binding library, VDIBINDB.L86, and the standard C run-time library provided with your C compiler.

In porting VDI applications written for PC DOS, you must, in addition to the steps above, convert the PC DOS calls into appropriate FlexOS SVC calls provided through the standard C run-time library.

Programs access VDI drivers through the C language bindings. The bindings transform the VDI calls into the FlexOS 286 SVC, S_GSX, which is the entry into FlexOS 286 Graphics Interface.

A VDI application first makes the s_open call to open the connection to the requested VDI driver (VDI01:). The application then makes the VDI Open Workstation (v_opnwk). This executes the v_opnwk routine in the VDI bindings library. v_opnwk then calls the GSX™SVC so that the application can access the Graphics Interface.

The Graphics Interface searches the ASSIGN.SYS file for the requested driver number. It then loads the associated font names into the font table. For a screen VDI, the Graphics Interface calls the Console Driver SPECIAL function 4 (c_special) to initialize the screen to Graphics Mode and creates a graphics virtual console. The VDI driver entry point is then called with the required VDI parameters and the workstation is initialized before returning to the application program.

All further VDI calls are handled in the same manner, using the device handle returned by v_opnwk.

When the application makes the VDI Close Workstation call (v_clswk), the Graphics Interface calls the VDI driver to clean up, and initializes the screen to Character Mode. Upon returning, the application performs the CLOSE SVC, severing the connection to the VDI driver, VDI01:.

4.5 FlexOS 286 VDI Restrictions

The following list summarizes the restrictions of the FlexOS 286 VDI:

- FlexOS 286 permits only full-screen graphics windows. Character windows and graphic windows cannot be simultaneously displayed on the same screen.
- Text rotation is not supported.
- Exchange vector calls are not supported.

- vrq_string
  vsm_string

  These functions do not support the echo mode. Also FlexOS screen VDIs do not support the VDI standard character set in Appendix D of the GEM VDI Reference Guide, but rather the FlexOS 16-bit character set.

- Plotter and Camera device drivers have not been implemented.

End of Section 4