FlexOS™ 386

Version 1.4

Beta Release Note 2

How this Release Note Differs from the Previous One

This FlexOS 386 Beta Release Note contains new information in Section 6, System Updates. Reading this section will bring you up to date on all that has been changed since the June 1987 release. If you have already read the prior release note, Section 6 is the only section that you need to read. If you have not read the June 1987 release note, read this release note in its entirety.

Contents

Section 1 describes the contents of the developer kits and the required hardware.

Section 2 describes how to use the bootscript, how to link FlexOS modules, and how to use the SASID386 debugger.

Section 3 describes shared memory and removable subdrivers.

Section 4 describes some new features of the FlexOS 386 development environment.

Section 5 contains a list of errors or omissions in the FlexOS documentation set.

Section 6 describes corrections and new features including new installation instructions, new driver service calls, and a list of DOS applications being certified.

July 1987 1079-1001-002

GENERAL SYSTEM INFORMATION

1.1 Beta Site Developer Kit Contents

FlexOS 386 beta software is shipped in two forms:

1. for those who already have the FlexOS 286 Developer Kit
2. for those who do not have the FlexOS 286 Developer Kit

Those who already have the FlexOS 286 Developer Kit receive the new material relevant to the 386 development environment; that is, the system disks, the FlexOS 386 Programmer's Utilities Guide and this release note. Those who do not have the 286 kit, receive the 386 information and the FlexOS 286 Developer Kit.

The FlexOS 286 Version 1.31 Developer Kit contains the FlexOS User's Guide, FlexOS Programmer's Guide, FlexOS System Guide, FlexOS Programmer's Utilities Guide, FlexOS Programmer's Utilities Guide Supplement.

Note: FlexOS 386 supports both 286-mode 16 bit addresses and 386-mode 32-bit addresses. However, you must use the MetaWare 286 High C Compiler to compile drivers. You must also use the 286 linker and debugger provided to generate and test system space code. See the FlexOS System Guide, Versions 1.3, for a full discussion of driver development.

FlexOS 386 is shipped on seven 5 1/4 inch quad density (1.12 Mb) disks organized into three product assemblies: Programmer's Toolkit, System Builder's Kit and the Debugger Disk.

1.1.1 Disk Contents

Generally, the files are grouped into subdirectories according to their purpose or use. The README file for each kit lists the files according to these groups and provide subdirectory definitions. We recommend printing out and reviewing the contents of the README.1 and README.2 files as soon as possible.

Each product assembly has a set of files appropriate to the tasks that purchasers will need to do. The Programmer's Toolkit supports creation of applications that use the FlexOS SVCs and run under FlexOS. This package does not contain FlexOS driver source code nor does it license further distribution of FlexOS.

The System Builder's Kit supports development of hardware drivers. The kit provides the FlexOS system object files, driver source, and link scripts. This package does not license further distribution of FlexOS.

The Debugger Disk contains SASID386, a stand-alone Symbolic Instruction Debugger, and a help file, DEBUG.DOC, that explains how to run it.

Be sure to backup all disks as soon as you open them to prevent against mishaps.

1.2 Required Hardware

Two executable versions of FlexOS are provided in the Programmer's Toolkit: one boots FlexOS from floppy disk, and the other boots FlexOS from a hard disk. These versions of FlexOS require the following hardware to run:

- Compaq® Deskpro 386™
- Minimum of 2Mb of RAM memory
- 20 Mb or greater hard disk with up to 4 partitions
- Quad density (1.2 Mb) 5 1/4 inch disk drive
- Serial port with the same adress as COM1 under MS-DOS™
- Centronics®-compatible port with the same address as PRN under MS-DOS
- IBM® Color Graphics Adapter (CGA), monochrome mode only, or
- IBM Enhanced Graphics Adapter (EGA), low resolution only
- Serial terminal for debugging and for developing multi-user applications (FlexOS was developed using Zenith® Z-29 terminals)

FlexOS™ 386 Beta release 3
15 September 1987

Dear FlexOS 386 Beta User,

The following information pertains to Beta release #3 of the FlexOS 386 Programmer's toolkit. Beta 3 is a refinement of the previous release, particularly in the FlexOS DOS Application Environment (DOS AE).

You install this Beta system as previously described in Beta Release Note 2. If you have an existing CONFIG.BAT file that contains any changes made since the Beta 2 release, you should save it in a separate directory or give it the Read-Only attribute before installing this release.

Effective with this release, the following system files have been renamed as indicated:

Old name	New name
FLEX286.SYS	FLEXOS.SYS
BLOAD286.IMG	BOOTLOAD.IMG

If you want to boot FlexOS from the first partition on the hard disk, you must rename these files as indicated before executing the install procedure on the distribution diskette.

Note: If you have configured your system to boot from any other partition that the first, you must create a bootable floppy disk.

Hard Disk Directory Cache Problem

There is a problem in the FlexOS file system that prevents the directory cache for permanent media from being placed into an "empty" state. This problem primarily affects the FORMAT and SYS utilities, which do not properly install the system as documented. Currently, the workaround for this problem is to reboot the system after performing any logical or physical hard disk format operation.

The SET Utility

FlexOS contains a new utility named SET that allows you to specify DOS environment strings. This utility is equivalent to the DOS SET (Set Environment) command, but unlike DOS which holds environent strings in memory, FlexOS's SET places the environment strings in a file named DOS.ENV in your SYSTEM: directory. (The SET utility is found in the TOOLS directory on disk #3.)

Examples:

```
A>set path = c:\dosaps
```

This command sets the search path for an application that looks for its files in the specified directory DOSAPS.

```
A>set
```

The command SET with no parameters displays the contents of the file DOS.ENV.

Note: You should consult the DOS documentation for complete information about environment strings.

Serial Ports

When installing applications, be sure to specify COM1 as the serial port, although you should physically connect the mouse to the serial port designated as COM2. FlexOS uses COM2 as the primary serial port because it uses COM1 for debugging. The FlexOS DOS AE internally remaps COM1 to COM2 so the mouse works correctly.

Using a Mouse

To use a mouse with GEM applications, you must install the mouse driver, and use the following CONFIG command to set up the serial port with the parameters as shown:

```
C>config ser: 1200 8 n 1
```

Note: Only one DOS application can have control of the mouse at any given time.

When using the mouse, faster response can be gained by double-clicking and then immediately moving the mouse in any direction rather than simply double-clicking without movement.

Running GEMPREP

To use GEMPREP for GEM version 2.2, you must boot DOS; GEMPREP only works under DOS. Be sure DOS is installed in a partition less than 32 Mb.

COMMAND.COM

The FlexOS DOS AE does not support applications that use COMMAND.COM.

EGA Support

The FlexOS DOS AE does not support the EGA.

Specific DOS Applications Guidelines

The following notes apply to running the indicated DOS applications:

1 Dbase II v 2.41

ADDMEM setting: 384

2 Dbase III v1.0

ADDMEM setting: 384

3 Dbase III Plus v1.1

ADDMEM setting: 384

4 DataFlex v2.2b (Full DEMO Version)

ADDMEM setting: 128

Special Notes:

- Add a space before the "/R" in the first line of the INSTALL.BAT file. This is required otherwise the shell tries to find an executable called R in a directory called "PKXARC" that is a subdirectory of the current directory.

5 GEM DeskTop v2.2

ADDMEM setting: 512

6 GEM Draw Plus v2.0

ADDMEM setting: 512

7 GEM Paint v2.0

ADDMEM setting: 512

8 GEM WordChart v1.0

ADDMEM setting: 512

9 GEM First Word Plus v1.0 (UK Release)

ADDMEM setting: 512

10 GEM Write v1.0

ADDMEM setting: 512

11 Lattice C Compiler v3.10

ADDMEM setting: 256

Special Notes:

- Users are recommended to use LINK.EXE v3.0.

12 Lattice C Compiler v3.20

ADDMEM setting: 512

Special Notes:

- Users are recommended to use LINK.EXE v3.0.

13 Lotus 1-2-3 v1A

ADDMEM setting: 128

14 Lotus 1-2-3 v2.01

ADDMEM setting: 512

15 MASM v4.0

ADDMEM setting: 512

Special Notes:

- Users are recommended to use LINK.EXE v3.0.

16 MicroEmacs v30.5

ADDMEM setting: 128

17 MicroFocus Level II Cobol v2.6

ADDMEM setting: 128

18 Microsoft Fortran v4.0

ADDMEM setting: 512

Special Notes:

- Users are recommended to use LINK.EXE v3.0.

19 Turbo C v1.0

ADDMEM setting: 256

20 Turbo Prolog v1.1

ADDMEM setting: 256

21 Turbo Pascal v3.02A

ADDMEM setting: 256

22 Wordstar Professional v4.0

ADDMEM setting: 128

1.3 Booting FlexOS on the Target System

See Section 6.3 for detailed installation instructions. Refer to the FlexOS User's Guide for instructions on creating windows and running the FlexOS utilities.

1.4 FlexOS Hard Disk Installation

Section 6.3.3 describes how to install FlexOS so that it will boot from a hard disk.

WARNING

If you create partitions larger than 32K, under no circumstances should you run MS/PC-DOS. Writing a file in that environment will destroy the FAT tables.

The FlexOS **FDISK** and **FORMAT** utilities give unpredictable results with hard disks above 64 Mb.

If you are going to change the partitioning on the hard disk before you install FlexOS, you **MUST** do a physical format of the hard disk first. If you do not, you may obtain unpredictable results.

End of Section 1

SYSTEM DEVELOPMENT

2.1 System Configuration

Certain portions of FlexOS can be defined during system initialization in the bootscript. For example, systems with sufficient memory can install a RAM disk from the script or, if the hardware is present, add serial ports. Refer to listing 3.1 on page 3-9 of the FlexOS System Guide, Version 1.3, for an example of a bootscript.

The basic bootscript is a user-modifiable file named CONFIG.BAT. The distribution bootscript is on the **Programmer's Toolkit #1** disk.

Other portions of FlexOS are defined in CONFIG.OBJ. This module specifies, for example, the resource managers and drivers to be loaded with the system. User changes are generally made in CONFIG.H. The source of CONFIG.OBJ is the C language file, CONFIG.C.

Refer to the FlexOS User's Guide to modify CONFIG.BAT, and to the FlexOS System Guide, section 3.3, to modify the CONFIG.C file.

2.2 FlexOS Library Modules

FlexOS is built primarily from library modules. There are three main libraries:

- COMLIB.L86 contains nonconfigurable modules
- ATLIB.L86 contains configurable driver modules
- FILESYS.L86 contains the file system manager

The disks in the System Builder's Kit provide the object sources and sample input files for building all these libraries. Refer to the README files for for descriptions of the input and object files.

2.2.1 Linking FlexOS Modules

The files DISP386.OBJ, CONFIG.OBJ, ACONF386.OBJ, CLOCK.OBJ, and CLOCKAT.OBJ are linked together with the modules in the ATLIB.L86, FILESYS.L86, and COMLIB.L86 libraries to form FlexOS.

The System Builder's Kit provides linker input files for creating bootable and non-bootable versions of FlexOS. Link the system using DEBUG.INP to create a non-bootable system for debugging drivers.

Link the system using BOOT386.INP to create a bootable system. A complete FlexOS system can be generated with this file.

After the link is complete, process the output file with the FIX.286 utility, then write the resulting file to the boot disk using the SYS.286 utility or FORMAT.286 under FlexOS. FIX.286 is explained in section 5.3 in this release note.

2.3 Building FLEX386.CMD, a Debuggable System

First, link the system with DEBUG.INP. This creates FLEX386.CMD, a debuggable version of FlexOS, and FLEX386.SYM, its corresponding symbol table. Boot the disk labelled **DEBUG Disk #1** which contains the debugger. This automatically loads the debugger SASID386.CMD, which in turn loads FLEX386.CMD and FLEX386.SYM for execution. The debugger executes from the attached serial terminal; FlexOS runs on the primary console.

Note: Since the debuggable system runs the debugger on the serial terminal, it is necessary to have that terminal connected.

The version of SASID386 provided runs under the "FlexOS Debug System" on a Compaq Deskpro 386 equipped with one quad density floppy disk drive.

2.4 Using SASID386

The SASID386 debugger is similar to the SID 286 debugger, and you will recognize many of the same commands from that utility. First, backup the disk containing SASID386 and put the copy in drive A of your machine. Invoke the program by first booting up the FlexOS Debug System. After a while, you will see the SASID386 prompt on the serial terminal screen.

The SASID386 loader looks for the FLEX386.CMD and FLEX386.SYM files, loads SASID386 at selector 0A00, then loads FlexOS 386 at selector 0A30. For more information on running SASID386, and changing the debugging file, see the text file called DEBUG.DOC included on **DEBUG Disk #1.**

Here is a list of the SASID386 commands:

Display Memory

b<address>,<length>,<address>	Compare memory
d<address>,<address>	Display bytes
dw<address>,<address>	Display words
dlw<address>,<offset>,<size>	Display linked list by words
l<address,address>	Disassemble code
sr<address>,<length>,<value>	Search for value

Examine Memory

s<address>	Display and set bytes
sw<address>	Display and set words
f<address>,<address>	Fill memory bytes
fw<address>,<address>	Fill memory words
m<address>,<length>,<value>	Move memory block

Execute

g<address>,<address>,<address>	Go at address until break
p<address>,<count>	Set passpoint
t<count>	Trace instructions
tw<count>	Trace without calls
u<count>	Trace without display
x	Display registers
c<address>,parm,parm...	Call a function

Miscellaneous

h	Display symbols
h.symbol	Display symbol offset
h<value>	Hex-decimal conversion
h<value1>,<value2>	Hex arithmetic
n<name>,<address>	Add name to symbols
qi<port>	Input byte from port
qiw<port>	Input word from port
qo<port>	Output byte to port
qow<port>	Output word to port

Note: A symbol or register can be used as an <address>.

2.5 Adding Drivers

Digital Research provides driver source code in the System Builder's Kit for the target system:

- Floppy Disk Driver
- Hard Disk Driver
- Printer Driver
- Serial Driver
- RAMdisk Driver
- Mouse Driver
- Console Driver

Note: This code is subject to ongoing revisions and optimizations. It is provided only as an example of how the driver code interfaces with FlexOS.

Drivers in this release are linked into the system by including the driver object files in the system build .INP files.

Alternatively, you can load drivers in the bootscript or load them interactively with the DVRLOAD command. To load them interactively, you must be a superuser.

When you make changes to files and recompile them, be sure the object files you create do not write over object files of the same name provided with this release.

All the C modules for FlexOS 386 Version 1.4 were compiled using MetaWare High C Version 1.3. Because of parameter passing conventions and other differences, you must also use High C to compile your driver code.

End of Section 2

SECTION 3

Shared Memory Considerations

3.1 Shared Memory

This feature lets multiple processes share common memory regions. Processes can also access specific physical memory locations, for dual ported RAM or system ROMs. (The following information is substantially the same as that provided with the Version 1.31 release of FlexOS 286.)

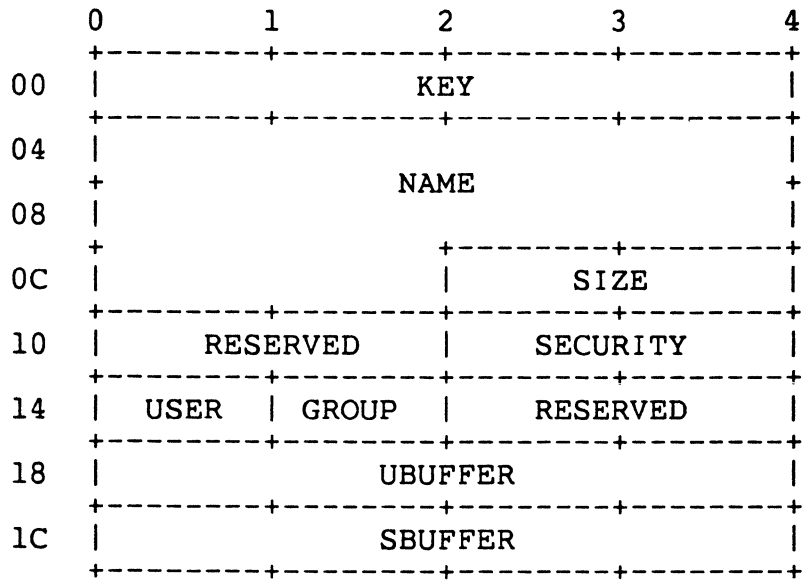
The processes can share data regions with drivers for fast communications in both protected and unprotected FlexOS environments, and multiple user processes can share data regions with each other. FlexOS grants access to shared memory only to those user processes with access rights established during system implementation.

There are two ways to access shared memory; through shared memory files, which work like pipes, and through the new driver services SHMEM and UN_SHMEM.

With the SHMEM table (illustrated below), a driver or process can create a shared memory file specifying a name for the memory allocation, a security word, and the size of the memory. Shared memory files have the "sm:" device name. You create a shared memory file with the CREATE SVC. See the description of CREATE in the FlexOS Programmer's Guide on page 7-26.

A subsequent OPEN SVC provides and verifies access to this file. The GET SVC returns a valid address for the shared memory region, the CLOSE SVC disables access via this address, and DELETE releases the region. Each shared data file also contains a semaphore, so drivers and processes can synchronize usage through the READ and WRITE SVCs.

The Pipe Resource Manager disallows an open request of "sm:" devices by any process with an rnid <> 0. This prevents a process on a remote node of a network from gaining access to shared data. Note that pipes are different in this respect: processes on one node can access pipes on remote nodes.



20H - Maximum Size of SHMEM Table

KEY	Unique ID
NAME	Name
RESERVED	Must be 0
SIZE	Size of memory area in bytes
SECURITY	Security word
USER	User ID of creator
GROUP	Group ID of creator
UBUFFER	User address of shared memory. This value is zero if the table was obtained through the LOOKUP SVC. You must use GET to obtain the address.

SBUFFER	System address of shared memory. This value is used by drivers and system processes independent of process context.
Device Type	0x11
Device Name	"sm:"
Table Number	0x11
SVC's supported	CREATE, OPEN, READ, WRITE, CLOSE, DELETE, GET and LOOKUP.

3.1.1 Shared Memory Driver Services

A shared memory driver service must be used for a new process to gain access to shared memory. The driver itself obtains system memory by utilizing the MAPPHYS() or SALLOC() driver services. It then allows a user process access to memory through a SHMEM() driver service. The region is released with the UN_SHMEM driver service. This gives a user process direct control of memory related devices. The OEM must write a shared memory driver to support an application's use of shared memory, but FlexOS 386 provides a set of subroutines that the driver can call.

SHare MEMory

```
BYTE    *usr_addr, *sys_addr;
UWORD   flags;

usr_addr = shmem(sys_addr, flags);
```

Parameters:

flags

bit 0: 0 = Read/Write buffer.
1 = Read Only buffer.

bits 1-15 are reserved.

sys_addr

System address obtained through SALLOC() or MAPPHYS().

Return Code:

usr_addr

User buffer address. 0 Indicates failure.

The SHMEM driver service lets a user process address system memory while running in user space.

UN_SHare MEMory

```
LONG    ret;
BYTE    *usr_addr;

ret = un_shmem(usr_addr);
```

Parameters:

usr_addr

User buffer address obtained through shmem().

Return Code:

ret0 indicated success; error code indicates bad usr_addr.

The UN_SHMEM() driver service reverses the function of a previous SHMEM() call. After this call, the user process gets an exception if it tries to access shared memory. If the user process passes an address to UN_SHMEM() that was not previously obtained through an SHMEM() call, it receives an error.

3.1.2 How to Use Shared Memory Files

A user process gains access to shared memory regions through shared a memory file, which is managed by the Pipe Resource Manager and accessed through the device name "sm:".

To create a shared memory region a user process performs the following calls:

```
fnum = s_create(0, flags, "sm:name", 0, security, size);
s_get(T_SHMEM, fnum, &shmem, sizeof(shmem))
buff_ptr = shmem.ubuffer;
```

BUFF_PTR now points to the shared memory.

If another user process wants to use the above shared memory file it performs the following calls:

```
fnum = s_open(flags, "sm:name");
s_get(T_SHMEM, fnum, &shmem, sizeof(shmem));
buff_ptr1 = shmem.ubuffer;
```

All references to *BUFF_PTR1 will access the named shared memory region.

A driver could give a user process access to a ROM of length LENGTH at address PHYS_ADDR by using the following calls:

```
struct
{
        LONG    link, pstart, plength;
} phys_mem = { 01, PHYS_ADDR, LENGTH };

sys_addr = (BYTE *)mapphys(&phys_mem, 1);
usr_addr = shmem(sys_addr, read_only_flag);
```

The user process would then use a SPECIAL() or GET() call to receive the user buffer address from the driver.

If two user processes need to synchronize access to a shared memory file they could each make the following calls:

```
s_read(0, fnum, "", 01, 01);     /* Get exclusive access  */
critical_code();                 /* Perform critical code */
s_write(0, fnum, "", 01, 01);    /* Release semaphore */
```

FNUM is the file number of the shared memory file obtained through the CREATE or OPEN calls.

When it no longer needs access to the shared memory file, the user process makes the call:

```
s_close(0, fnum);
```

FNUM is the file number that was attained by the create or open calls.

If the driver wants to remove user access to the shared memory it created it makes the call:

```
un_shmem(usr_addr);
```

usr_addr is the address obtained by the SHMEM() call.

3.2 Removable Subdrivers

FlexOS has the ability to remove subdrivers. This feature is implemented through the standard user DVRUNLK command and supervisor INSTALL function. For example, the user enters the subdriver device name in the DVRUNLK command to remove the subdriver from a driver. Similarly, the programmer uses the INSTALL SVC with the option field set to 0 and the devname field set to the subdriver name address to remove a driver.

Subdrivers like drivers are set as removable or permanent in INSTALL flag bit 5. When the bit is set the subdriver is marked as removable; otherwise it should not be removable. Permanent versus removable install status is reflected in the DEVICE Table's INSTAT field. For subdrivers, the fields are defined as follows:

0x00 - Not installed
Qx01 - Requires subdriver
0x02 - Owned by Miscellaneous Resource Manager
0x03 - Owned by another driver
0x04 - Optional subdriver

Drivers are informed to remove a subdrive through the SUBDRIVE function entry point. This entry point is now used both to associate and disassociate a subdriver. To indicate which operation to perform, bit 10 in the Access field is set as follows:

Bit 10: 0 = Install subdriver
1 = Uninstall subdriver

The remainder of the Access flags remain as defined in Table 4-4, INSTALL Flags in the FlexOS System Guide.

The driver should then do what's necessary to remove the subdriver. Note, however, that the driver can ignore the request, for example, if the subdriver is currently in use. The following sample code illustrates a SUBDRIVE routine that handles both installation and removal of the subdriver.

```
LONG    s_subdrvr(pb)
DPB     *pb;
{
    PHYSBLK     *d;

    if(pb->dp_flags & 0x400){
        sfree(sdev[pb->dp_option]);
        sdev[pb->dp_option] = 0;
        return((((LONG)DVR_PORT << 16) | (LONG)DVR_SER));
        }

    ser_unit[pb->dp_option] = pb->dp_unitno;
    pt_hdr[pb->dp_unitno] = (DH *) pb->dp_swi;
    pt_unit[pb->dp_unitno] = pb->dp_option;

    d = sdev[pb->dp_option] =
        (PHYSBLK *) salloc( (LONG)sizeof(PHYSBLK));
    d->Qrear = d->Qfront = d->evpend =
        d->xoffed = d->Qlen = 0;

    return(E_SUCCESS);
}
```

The return code from the SUBDRIVE function should indicate the type of subdriver required or 0, if no subdriver is required.

Typically, the SUBDRIVE routine is not the only portion of the driver involved in the subdrive interface. For example, you should also free the resources (for example, flags, pipes and memory for data structures) used to enable device I/O when the remove command is received. A general rule of thumb regarding subdriver removal is: Everything done in INIT and SELECT to support device I/O should be undone in UNINIT and FLUSH, respectively.

End of Section 3

386 Development Environment

4.1 Overall Considerations

FlexOS 386 uses the paging portion of the 386 chip for efficient memory management. It also supports a flat 32-bit program load format. Also, there is no longer a 64K limit on memory allocations when the S_MALLOC SVC is called by a 386 program.

There is a change in the exception numbers for the FlexOS EXCEPTION SVC as they correspond to the 80286 exception vectors. FlexOS EXCEPTION numbers 11 - 18 (for the condition "emulated instruction group 0") now correspond to 80286 vector number 7.

FlexOS 386 runs all 286 drivers and applications. In fact, FlexOS 386 Release 1.4 supports 286 mode drivers only.

4.1.1 Program Development Tools

In addition to the 286 Assembly Language Programming tools provided with FlexOS, there are new 386 model program development tools provided with this release that are based on the Common Object File Format (COFF). They are:

Tool	Description
CASM	386 Assembler
CLINK	COFF Linkage Editor
CLIB	COFF Librarian
CSID	386 Model Program Debugger
CTO	COFF to OBJ conversion utility

4.1.2 Reserved File Extensions

In addition to the reserved file extensions listed in the FlexOS Supplement for Intel iAPX 286-based Computers, the following extensions have been reserved for the 386 development environment.

Extension	Description
A	CASM Assembly language file
O	COFF 80386 object file
386	Command file that runs directly under the operating system in 32-bit mode

4.1.3 Entry Mechanism

Entry into FlexOS 386 by 386 programs is made by application code using INT 221 as the entry point with the parameter and return value as shown below.

Register	Contents
ECX	SVC number
EAX	Parameter block address
EAX (return)	Return Value

For a sample program illustrating the entry mechanism, see the COFF Programming Utilities Guide. Also described in that manual is the 386 application program memory model.

4.1.4 System Generation Utilities

FlexOS itself is compiled into OBJ format object files using the 16-bit addressing mode. Therefore, driver code must be compiled into the same format object files so that they can be linked with the system object files. The following 16-bit utilities are used to generate a FlexOS 386 system, to construct drivers, and to develop FlexOS 286-compatible applications. They are described in full in the FlexOS Programmer's Utilities Guide.

Utility	Description
RASM86.286	Relocatable Assembler
LINK86.286	Linker
LIB86.286	Librarian
SID86.286	Symbolic instruction debugger
FIX.286	Generates a relocated operating system image

End of Section 4

DOCUMENTATION ERRATA

5.1 FlexOS User's Guide

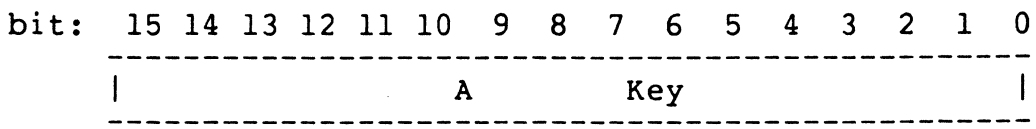
The following are errors or omissions in the FlexOS User's Guide, First Edition November 1986:

Page 2-2	Window 1 is the dedicated message window, and window 2 is the status window. The first window that actually shows is window 3. When you pull up the status window, you may not see them, but they are there at all times.
Page 2-5	The <HELP> key on the keyboard is actually CTRL-<INS>.
Page 7-7	Following the second paragraph in the explanation add the following: **Note:** Invoking a batchfile in the background causes a shell to be invoked, and the shell, in turn, runs the batchfile. Thus, the process ID returned is that of the shell. Therefore, batchfiles cannot be stopped with the CANCEL command, but the shell can be stopped.
Page 7-25	Omit the filetype CMD from the sentence reading ". . . with the extension 286, 386, 68K, CMD, COM, or EXE."
Page 7-32	The paragraph should read, "When you copy a file, the date and time of the source file are copied to the destination file's directory information."
Page 7-53	Delete the Note that states DISKCOPY will format an unformatted disk while copying. This is incorrect; DISKCOPY does not format an unformatted disk.
Page 7-79	In the example, remove the "Press any key to begin" line. LOGOFF does not issue any prompt.

Page 7-85	Omit the filetype CMD from the sentence "These files extensions include 286, 386, 68K, CMD, CMD, COM, and EXE."
Page 7-103	Omit "(a-p)" after current drive. There is no limitation on drive name.
Page A-1	Change the sentence "Use CONFIG.BAT tset up the LOGON/LOGOFF . . . " to read "Use CONFIG.BAT to set up the LOGON/LOGOFF . . ."

5.2 FlexOS System Guide

The figure on page A-3 of the FlexOS System Guide is incorrect. It should be:



5.3 FlexOS Programmer's Utilities Guide

FIX Cross-Reference Utility

FIX.286 is a system generation utility program that generates an output file containing a relocated FLEX286.SYS operating system image from a relocatable operating system file in standard .286 format.

FIX 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 command line), FIX does not create the GDT and IDT, which are used in protected mode only. FIX expects the OS Data Header to be the first item in the data segment.

FIX Command Syntax

FIX is invoked using the command form:

FIX input.fil output.fil [/r]

For example, to create the system boot loader BLOAD286.IMG, first assemble and link the BLOAD286.A86 file to create a BLOAD286.286 file, then enter:

FIX BLOAD286.286 BLOAD286.IMG /r

The BLOAD286.IMG file is then ready to be placed on the boot disk, using the SYS.286 utility, to create a bootable disk.

FIX also creates a protected mode system image from the relocatable file produced by the loader. For example, link the BOOT386.INP file to produce BOOTPROT.286, then give the command:

FIX BOOTPROT.286 FLEX286.SYS

The resulting image file FLEX286.SYS is copied to a disk with a bootloader image (BLOAD286.IMG) using the COPY.286 utility, or it is placed on a disk with the SYS.286 utility.

SYSTEM UPDATES

6.1 New Information

- Memory Usage -- FlexOS 386 is now able to access 384K of previously "hidden" memory in the Compaq Deskpro^{TM} 386 and use it for user program space.
- Hard Disk Support -- The hard disk driver for the Compaq Deskpro 386 has been improved to support all the controllers and drives available through Compaq (40, 70, 130 Mb.). Work is ongoing to correct the performance degradation on the 70 Mb. drive, however. The following utilities support the drives: CHKDSK, FDISK, FORMAT, and SYS.
- Compiler Released -- MetaWare High C-386, and its FlexOS 386 run-time-library is in beta release and may be used to construct native-mode, .386, 32-bit applications. Contact MetaWare at (408) 429-6382 to obtain a test copy.
- Driver Services -- There are two new driver service calls (CSALLOC and CONTIG). They are described at the end of this section.
- DOS Functions -- The FlexOS 386 DOS Application Environment emulates MS DOS 3.2 functions on a call-by-call, register-by-register basis. It also allows programs to support a mix of FlexOS and MS DOS functions. Careful management is required when mixing functions because the impact of this practice on file access and resources.
- DOS Application Environment Disk -- A disk has been included that contains the object files and system library necessary to link the MS/PC DOS Application Environment to the bootable system. See the README file on the disk for instructions.
- DOS Applications -- The Applications listed on the next page are currently being certified under the MS/PC DOS application environment.

- CASM Fix -- The COFF 386 assembler, CASM, is now able to assemble the **movsx** and **movzx** instructions correctly.
- SVC Fix -- The **s_control** SVC now properly traps page fault exceptions of .386 programs. This is of interest to programmers writing debuggers since this situation arises only when the debugged process attempts to access memory that not allocated.
- **Warning:** Do not create hard disk partitions greater than 32Mb if both MS/PC DOS and FlexOS are to share that partition. Corruption of data may occur. (FlexOS 386 supports partitions of greater than 32Mb but DOS does not.)
- **Warning:** The memory containing the code sections of .386 programs is marked read-only. Programs attempting to write data to variables within their code section will be **aborted** with a general protection error. This also happens if a .386 program's stack overflows its allocated memory pages.

6.2 Applications being Certified

These MS/PC DOS applications have been assigned top priority for certification:

dBase II, III, III+
Lotus 1-2-3 (v1A)
MASM
MicroEmacs
Multimate Advantage
Norton Utilities, Advanced
R:Base, System V
Solomon Accounting
Turbo Pascal
Turbo Prolog
GEM Desktop
GEM Draw Plus
GEM Graph
GEM Paint
GEM Desktop Publisher
GEM Wordchart
GEM Write
ThinkTank
Wordstar Professional

The following MS/PC DOS applications are of secondary priority during the beta period but will continue to be tested for subsequent certification.

AutoCad
Clipper
Microsoft Chart
Microsoft Project

Compu-brush	Microsoft Word
Crosstalk	Microsoft Codeview
Dataflex	Microsoft Windows
EGA Paint	Norton Utilities
Enable	Profs
EZ-VU	Perspective
Instant C	Storyboard
Lattice C	Supercalc 3
Lotus 1-2-3 (v.2.01)	Turbo C
Lotus Freelance Plus	Ventura Publisher
Microfocus COBOL	Volkswriter Deluxe
Microsoft C	Word Perfect

Beta certifiers are encouraged to report results with any and all applications tested under the FlexOS 386 DOS application environment. Contact Connie Paul at (408)-649-3896.

6.3 INSTALLING FLEXOS 386

The installation process involves creating subdirectories with specific names and transferring designated files into those directories. Since this is a complex and detailed process, a "batch file" is used to accomplish this transfer. The following procedure describes how to install the operating system, the Programmer's Toolkit, and the System Builder's Kit on your hard disk. You may install the system to boot from either a floppy or a hard disk.

This version of FlexOS 386 has been developed for use on the Compaq® Deskpro™ 386 with 20 megabyte or greater hard disk. FlexOS 386 can be used on other computers utilizing the Intel® 80386 microprocessor but the drivers must be modified or rewritten to accomodate differences in the hardware.

6.3.1 Installing the Programmer's Toolkit

Note: This method of installation allows you to boot FlexOS 386 from floppy disk. If you want to boot FlexOS from your hard disk, go to the next section.

Turn on your computer and put Disk #1 of the Programmer's Toolkit in

drive A. The system will load and soon you will be asked for a username. You have two options here: to bring up a single tasking system or a concurrent multitasking system. To bring up a single user system, enter "user" as your username. You are then asked for a password, again enter "user."

To bring up a multitasking system (with virtual consoles), enter "system" as your username and also as your password. FlexOS will continue to load and the system prompt (A>) will come on the screen.

If your hard disk or bootable partition is not labelled C:, you must tell FlexOS what its label is. For instance, if your hard disk is labelled D:, enter the following:

A>ASSIGN C:=D:

This tells FlexOS that it must go to drive D: to create subdirectories and transfer files. You will not be able to boot FlexOS from a hard disk that has a label other than C: using the hard disk system provided. If you would like to build a system that will boot from a D: drive or partition, see the note under 6.3.3 below.

To resume the installation, type the following:

A>INSTALL1

As INSTALL1 runs the instructions that it is executing appear on the screen.

Note: The INSTALL1.BAT batch file is used to transfer the Programmer's Toolkit and INSTALL2.BAT is used to transfer the System Builder's Kit.

When all the files from Programmer's Toolkit Disk #1 have been transferred, you will be asked to remove Disk #1 from drive A and put Disk #2 in drive A and close the latch.

When all the files from Programmer's Toolkit #2 have been transferred, you will be asked to remove Disk #2 from drive A and put Programmer's Toolkit Disk #3 in drive A and close the latch.

The installation process will continue once again and more directories will be created on the hard disk and the last of the Programmer's Toolkit files will be transferred.

6.3.2 Installing the System Builder's Kit

The System Builder's Kit is installed in the same way as the Programmer's Toolkit. After booting FlexOS 386, put the System Builder's Kit Disk #1 in drive A and close the latch.

Enter the following command:

A>INSTALL2

When the files on Disk #1 have been transferred, you will be asked to insert Disk #2. After the files on Disk #2 have been transferred, insert Disk #3 and close the latch. When the files on Disk #3 have been transferred, the installation process is complete.

6.3.3 Booting FlexOS 386 from a Hard Disk

In order to have FlexOS 386 boot from your hard disk, it must be installed as follows. First turn on your computer and then put Programmer Toolkit Disk #1 in drive A and close the latch. The system loads and soon you will be asked for a username. Enter "user" or "system" as explained in 6.3.1 above. You are then asked for a password, again enter "user" or "system." FlexOS will continue to load and the system prompt (A>) will come on the screen.

For the following procedure to be successful, your hard disk (or active partition) **must** be designated C:. If your hard disk or partition is designated D: (or any other letter) you will still be able to run FlexOS 386 but you will not be able to boot from the hard disk unless you set up the system differently.

Note: An Application Note is available that explains how you can set up FlexOS to boot from a D: drive or partition. Contact Connie Paul at (408)649-3896 to obtain one.

Type the following command at the system prompt from the root directory of drive A.

A>SYS C:

This command copies the executable form of FlexOS from the root directory of drive A to the root directory of drive C. When the files have been transferred, enter the following command:

A>INSTALL1

This will start the installation process as described in 6.3.1 above.

6.4 New Driver Service Calls

Two new driver services have been added to FlexOS 386. They have been added so that the FlexOS family of operating systems may more efficiently exploit memory management units (MMUs), which support paging of physical memory. These driver services will be supported in the current version of FlexOS 386 and all versions of FlexOS 286 starting with version 1.4.

6.4.1 CSALLOC Driver Service

In earlier versions of FlexOS, a driver would call the driver service, SALLOC, to allocate memory in the system address space. The driver could rely on the fact that this memory was always physically contiguous, so external devices under the driver's control, such as DMA controllers which use physical addresses and bypass the MMU, would work properly. In FlexOS 386, this assumption is no longer true. If the memory to be allocated must be physically contiguous, the CSALLOC service call must be used instead of SALLOC. A bit in the flags parameter determines whether contiguity is required.

The second reason for calling CSALLOC is to allocate memory which must be physically isolated from other system buffers. This use of the call is to exert control over a buffer which may be passed (by the DOS Application Environment to a user process. If the allocation for the Application Environment were in the same page as the allocation for an important system data structure, the user process would have the potential to corrupt system data. The "isolate" bit in the CSALLOC flags word is used to control this area of memory protection. In FlexOS 386, an isolated buffer will start on a 4Kb boundary and the allocation will extend in multiples of 4Kb. The 80286 does not support the same sort of hardware mapping, so a setting of the ISOLATE bit on a call to CSALLOC in FlexOS 286 is ignored.

C Interface for CSALLOC:

```
sysadr = csalloc (length, flags);
```

Parameters:

```
BYTE *    sysadr;       /* System address of memory block     */
                        /* allocated.  0 indicates no         */
                        /* memory available.                  */
ULONG     length;       /* Number of bytes to allocate        */
UWORD     flags;        /* Bit 0:
                             1 = Physical contiguity required
                             0 = Non-contiguity accepted
                          Bit 1:
                             1 = Physical isolation required
                             0 = Not required
                          Other bits must = 0.               */

/*'salloc(length)'is functionally equivalent to 'csalloc(length,0)'*/
```

6.4.2 CONTIG Driver Service

In earlier versions of FlexOS, drivers that did DMA to and from physical memory addresses would call the driver service, PADDR, to convert the system address of the buffer to a physical address. The driver could assume that the whole buffer was physically contiguous starting from the address returned. As described above, this assumption is no longer true in FlexOS v1.4. A new driver service, CONTIG, has been added which is used to find the physical address of a buffer and the number of bytes that are physically contiguous from that point.

C Interface:

```
size = contig (buffer, length, &phyadr);
```

Parameters:

```
ULONG     size;       /* Number of bytes that are physically   */
                      /*   contiguous.                         */
BYTE *    buffer;     /* System address of buffer              */
ULONG     length;     /* Length (bytes) of buffer              */
BYTE *    phyadr;     /* Physical address of buffer (returned) */
```

The following conditions must be met before calling CONTIG for the first time for a given buffer:

1. The owner of the buffer must be mapped into memory (MAPU).
2. The buffer must have passed a MRANGE call.

After finding the number of bytes that are contiguous in the buffer, the driver can do DMA to or from that portion of the buffer. Note that the physical address of the buffer is returned via 'phyadr'. If the return 'size' is not equal to the 'length', then the buffer is not contiguous and more calls to CONTIG will be required. For the next call, 'length' should be decreased by 'size', and the buffer address should be increased by 'size'. Then, assuming the process "owning" the buffer is still mapped in, another call to CONTIG can be made. Repeat the above until the returned 'size' and 'length' are the same.

End of Release Note

FlexOS™ 386

Version 1.4

Beta Release Note

June 1987

Contents

Section 1 describes the contents of the developer kits, the required hardware, and how to boot up FlexOS from the floppy disks and then install it on your hard disk.

Section 2 describes how to use the bootscript, how to link FlexOS modules, and how to use the SASID386 debugger.

Section 3 describes shared memory and removable subdrivers.

Section 4 describes some new features of the FlexOS 386 development environment.

Section 5 contains a list of errors or omissions in the FlexOS documentation set.

1079-1001-001

SECTION 1

GENERAL SYSTEM INFORMATION

1.1 Beta Site Developer Kit Contents

FlexOS 386 beta software is shipped in two forms:

1. for those who already have the FlexOS 286 Developer Kit
2. for those who do not have the FlexOS 286 Developer Kit

Those who already have the FlexOS 286 Developer Kit receive the new material relevant to the 386 development environment; that is, the system disks, the FlexOS 386 Programmer's Utilities Guide and this release note. Those who do not have the 286 kit, receive the 386 information and the FlexOS 286 Developer Kit.

The FlexOS 286 Version 1.31 Developer Kit contains the FlexOS User's Guide, FlexOS Programmer's Guide, FlexOS System Guide, FlexOS Programmer's Utilities Guide, FlexOS Programmer's Utilities Guide Supplement.

Note: FlexOS 386 supports both 286-mode 16 bit addresses and 386-mode 32-bit addresses. However, you must use the Metaware 286 High C Compiler to compile drivers. You must also use the 286 linker and debugger provided to generate and test system space code. See the FlexOS System Guide, Versions 1.3, for a full discussion of driver development.

FlexOS 386 is shipped on seven 5 1/4 inch quad density (1.12 Mb) disks organized into three product assemblies: Programmer's Toolkit, System Builder's Kit and the Debugger Disk.

1.1.1 Disk Contents

Generally, the files are grouped into subdirectories according to their purpose or use. The README file for each kit lists the files according to these groups and provide subdirectory definitions. We recommend printing out and reviewing the contents of the README.1 and README.2 files as soon as possible.

Each product assembly has a set of files appropriate to the tasks that purchasers will need to do. The Programmer's Toolkit supports creation of applications that use the FlexOS SVCs and run under FlexOS. This package does not contain FlexOS driver source code nor does it license further distribution of FlexOS.

The System Builder's Kit supports development of hardware drivers. The kit provides the FlexOS system object files, driver source, and link scripts. This package does not license further distribution of FlexOS.

The Debugger Disk contains SASID386, a stand-alone Symbolic Instruction Debugger, and a help file, DEBUG.DOC, that explains how to run it.

Be sure to backup all disks as soon as you open them to prevent against mishaps.

1.2 Required Hardware

Two executable versions of FlexOS are provided in the Programmer's Toolkit: one boots FlexOS from floppy disk, and the other boots FlexOS from a hard disk. These versions of FlexOS require the following hardware to run:

- Compaq® Deskpro 386™
- Minimum of 2Mb of RAM memory
- 20 Mb or greater hard disk with up to 4 partitions
- Quad density (1.2 Mb) 5 1/4 inch disk drive
- Serial port with the same adress as COM1 under MS-DOS™
- Centronics®-compatible port with the same address as PRN under MS-DOS
- IBM® Color Graphics Adapter (CGA), monochrome mode only, or
- IBM Enhanced Graphics Adapter (EGA), low resolution only
- Serial terminal for debugging and for developing multi-user applications (FlexOS was developed using Zenith® Z-29 terminals)

1.3 Booting FlexOS on the Target System

Turn on the computer, then immediately insert the disk marked

Programmer's Toolkit #1

in drive A and close the door. The system is self-booting. Refer to the FlexOS User's Guide for instructions on creating windows and running the FlexOS utilities.

1.4 FlexOS Hard Disk Installation

The INSTALL.BAT file transfers system files onto your hard disk so you can boot from the hard disk rather than from a floppy. As with the README files, there are INSTALL.BAT files for each kit, INSTALL1.BAT for the Programmer's Toolkit and INSTALL2.BAT for the System Builder's Kit.

WARNING

If you create partitions larger than 32K, under no circumstances should you run MS/PC-DOS. Writing a file in that environment will destroy the FAT tables.

The FlexOS **FDISK** and **FORMAT** utilities give unpredictable results with hard disks above 64 Mb.

If you are going to change the partitioning on the hard disk before you install FlexOS, you **MUST** do a physical format of the hard disk first. If you do not, you may obtain unpredictable results.

End of Section 1

SYSTEM DEVELOPMENT

2.1 System Configuration

Certain portions of FlexOS can be defined during system initialization in the bootscript. For example, systems with sufficient memory can install a RAM disk from the script or, if the hardware is present, add serial ports. Refer to listing 3.1 on page 3-9 of the FlexOS System Guide, Version 1.3, for an example of a bootscript.

The basic bootscript is a user-modifiable file named CONFIG.BAT. The distribution bootscript is on the **Programmer's Toolkit #1** disk.

Other portions of FlexOS are defined in CONFIG.OBJ. This module specifies, for example, the resource managers and drivers to be loaded with the system. User changes are generally made in CONFIG.H. The source of CONFIG.OBJ is the C language file, CONFIG.C.

Refer to the FlexOS User's Guide to modify CONFIG.BAT, and to the FlexOS System Guide, section 3.3, to modify the CONFIG.C file.

2.2 FlexOS Library Modules

FlexOS is built primarily from library modules. There are three main libraries:

- COMLIB.L86 contains nonconfigurable modules
- ATLIB.L86 contains configurable driver modules
- FILESYS.L86 contains the file system manager

The disks in the System Builder's Kit provide the object sources and sample input files for building all these libraries. Refer to the README files for for descriptions of the input and object files.

2.2.1 Linking FlexOS Modules

The files DISP386.OBJ, CONFIG.OBJ, ACONF386.OBJ, CLOCK.OBJ, and CLOCKAT.OBJ are linked together with the modules in the ATLIB.L86, FILESYS.L86, and COMLIB.L86 libraries to form FlexOS.

The System Builder's Kit provides linker input files for creating bootable and non-bootable versions of FlexOS. Link the system using DEBUG.INP to create a non-bootable system for debugging drivers.

Link the system using BOOT386.INP to create a bootable system. A complete FlexOS system can be generated with this file.

After the link is complete, process the output file with the FIX.286 utility, then write the resulting file to the boot disk using the SYS.286 utility or FORMAT.286 under FlexOS. FIX.286 is explained in section 5.3 in this release note.

2.3 Building FLEX386.CMD, a Debuggable System

First, link the system with DEBUG.INP. This creates FLEX386.CMD, a debuggable version of FlexOS, and FLEX386.SYM, its corresponding symbol table. Boot the disk labelled **DEBUG Disk #1** which contains the debugger. This automatically loads the debugger SASID386.CMD, which in turn loads FLEX386.CMD and FLEX386.SYM for execution. The debugger executes from the attached serial terminal; FlexOS runs on the primary console.

Note: Since the debuggable system runs the debugger on the serial terminal, it is necessary to have that terminal connected.

The version of SASID386 provided runs under the "FlexOS Debug System" on a Compaq Deskpro 386 equipped with one quad density floppy disk drive.

2.4 Using SASID386

The SASID386 debugger is similar to the SID 286 debugger, and you will recognize many of the same commands from that utility. First, backup the disk containing SASID386 and put the copy in drive A of your machine. Invoke the program by first booting up the FlexOS Debug System. After a while, you will see the SASID386 prompt on the serial terminal screen.

The SASID386 loader looks for the FLEX386.CMD and FLEX386.SYM files, loads SASID386 at selector 0A00, then loads FlexOS 386 at selector 0A30. For more information on running SASID386, and changing the debugging file, see the text file called DEBUG.DOC included on **DEBUG Disk #1.**

Here is a list of the SASID386 commands:

Display Memory

b<address>,<length>,<address>	Compare memory
d<address>,<address>	Display bytes
dw<address>,<address>	Display words
dlw<address>,<offset>,<size>	Display linked list by words
l<address,address>	Disassemble code
sr<address>,<length>,<value>	Search for value

Examine Memory

s<address>	Display and set bytes
sw<address>	Display and set words
f<address>,<address>	Fill memory bytes
fw<address>,<address>	Fill memory words
m<address>,<length>,<value>	Move memory block

Execute

g<address>,<address>,<address>	Go at address until break
p<address>,<count>	Set passpoint
t<count>	Trace instructions
tw<count>	Trace without calls
u<count>	Trace without display
x	Display registers
c<address>,parm,parm...	Call a function

Miscellaneous

h	Display symbols
h.symbol	Display symbol offset
h<value>	Hex-decimal conversion
h<value1>,<value2>	Hex arithmetic
n<name>,<address>	Add name to symbols
qi<port>	Input byte from port
qiw<port>	Input word from port
qo<port>	Output byte to port
qow<port>	Output word to port

Note: A symbol or register can be used as an <address>.

2.5 Adding Drivers

Digital Research provides driver source code in the System Builder's Kit for the target system:

- Floppy Disk Driver
- Hard Disk Driver
- Printer Driver
- Serial Driver
- RAMdisk Driver
- Mouse Driver
- Console Driver

Note: This code is subject to ongoing revisions and optimizations. It is provided only as an example of how the driver code interfaces with FlexOS.

Drivers in this release are linked into the system by including the driver object files in the system build .INP files.

Alternatively, you can load drivers in the bootscript or load them interactively with the DVRLOAD command. To load them interactively, you must be a superuser.

When you make changes to files and recompile them, be sure the object files you create do not write over object files of the same name provided with this release.

All the C modules for FlexOS 386 Version 1.4 were compiled using MetaWare High C Version 1.3. Because of parameter passing conventions and other differences, you must also use High C to compile your driver code.

End of Section 2

Shared Memory Considerations

3.1 Shared Memory

This feature lets multiple processes share common memory regions. Processes can also access specific physical memory locations, for dual ported RAM or system ROMs. (The following information is substantially the same as that provided with the Version 1.31 release of FlexOS 286.)

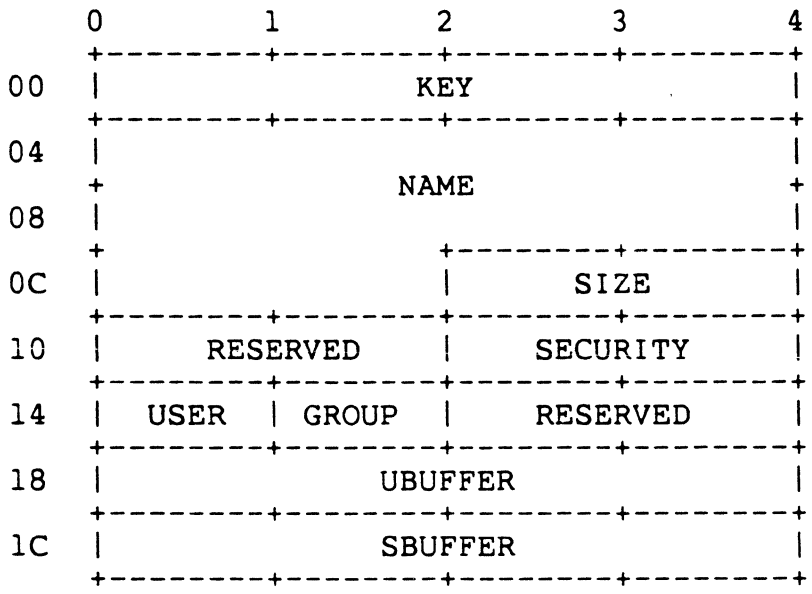
The processes can share data regions with drivers for fast communications in both protected and unprotected FlexOS environments, and multiple user processes can share data regions with each other. FlexOS grants access to shared memory only to those user processes with access rights established during system implementation.

There are two ways to access shared memory; through shared memory files, which work like pipes, and through the new driver services SHMEM and UN_SHMEM.

With the SHMEM table (illustrated below), a driver or process can create a shared memory file specifying a name for the memory allocation, a security word, and the size of the memory. Shared memory files have the "sm:" device name. You create a shared memory file with the CREATE SVC. See the description of CREATE in the FlexOS Programmer's Guide on page 7-26.

A subsequent OPEN SVC provides and verifies access to this file. The GET SVC returns a valid address for the shared memory region, the CLOSE SVC disables access via this address, and DELETE releases the region. Each shared data file also contains a semaphore, so drivers and processes can synchronize usage through the READ and WRITE SVCs.

The Pipe Resource Manager disallows an open request of "sm:" devices by any process with an rnid <> 0. This prevents a process on a remote node of a network from gaining access to shared data. Note that pipes are different in this respect: processes on one node can access pipes on remote nodes.



20H - Maximum Size of SHMEM Table

KEY	Unique ID
NAME	Name
RESERVED	Must be 0
SIZE	Size of memory area in bytes
SECURITY	Security word
USER	User ID of creator
GROUP	Group ID of creator
UBUFFER	User address of shared memory. This value is zero if the table was obtained through the LOOKUP SVC. You must use GET to obtain the address.

SBUFFER	System address of shared memory. This value is used by drivers and system processes independent of process context.
Device Type	0x11
Device Name	"sm:"
Table Number	0x11
SVC's supported	CREATE, OPEN, READ, WRITE, CLOSE, DELETE, GET and LOOKUP.

3.1.1 Shared Memory Driver Services

A shared memory driver service must be used for a new process to gain access to shared memory. The driver itself obtains system memory by utilizing the MAPPHYS() or SALLOC() driver services. It then allows a user process access to memory through a SHMEM() driver service. The region is released with the UN_SHMEM driver service. This gives a user process direct control of memory related devices. The OEM must write a shared memory driver to support an application's use of shared memory, but FlexOS 386 provides a set of subroutines that the driver can call.

SHare MEMory

```
BYTE    *usr_addr, *sys_addr;
UWORD   flags;

usr_addr = shmem(sys_addr, flags);
```

Parameters:

flags

bit 0: 0 = Read/Write buffer.
1 = Read Only buffer.

bits 1-15 are reserved.

sys_addr

System address obtained through SALLOC() or MAPPHYS().

Return Code:

usr_addr

User buffer address. 0 Indicates failure.

The SHMEM driver service lets a user process address system memory while running in user space.

UN_SHare MEMory

```
LONG    ret;
BYTE    *usr_addr;

ret = un_shmem(usr_addr);
```

Parameters:

usr_addr

User buffer address obtained through shmem().

Return Code:

ret0 indicated success; error code indicates bad usr_addr.

The UN_SHMEM() driver service reverses the function of a previous SHMEM() call. After this call, the user process gets an exception if it tries to access shared memory. If the user process passes an address to UN_SHMEM() that was not previously obtained through an SHMEM() call, it receives an error.

3.1.2 How to Use Shared Memory Files

A user process gains access to shared memory regions through shared a memory file, which is managed by the Pipe Resource Manager and accessed through the device name "sm:".

To create a shared memory region a user process performs the following calls:

```
fnum = s_create(0, flags, "sm:name", 0, security, size);
s_get(T_SHMEM, fnum, &shmem, sizeof(shmem))
buff_ptr = shmem.ubuffer;
```

BUFF_PTR now points to the shared memory.

If another user process wants to use the above shared memory file it performs the following calls:

```
fnum = s_open(flags, "sm:name");
s_get(T_SHMEM, fnum, &shmem, sizeof(shmem));
buff_ptr1 = shmem.ubuffer;
```

All references to *BUFF_PTR1 will access the named shared memory region.

A driver could give a user process access to a ROM of length LENGTH at address PHYS_ADDR by using the following calls:

```
struct
{
        LONG    link, pstart, plength;
} phys_mem = { 01, PHYS_ADDR, LENGTH };

sys_addr = (BYTE *)mapphys(&phys_mem, 1);
usr_addr = shmem(sys_addr, read_only_flag);
```

The user process would then use a SPECIAL() or GET() call to receive the user buffer address from the driver.

If two user processes need to synchronize access to a shared memory file they could each make the following calls:

```
s_read(0, fnum, "", 01, 01);     /* Get exclusive access  */
critical_code();                 /* Perform critical code */
s_write(0, fnum, "", 01, 01);    /* Release semaphore */
```

FNUM is the file number of the shared memory file obtained through the CREATE or OPEN calls.

When it no longer needs access to the shared memory file, the user process makes the call:

```
s_close(0, fnum);
```

FNUM is the file number that was attained by the create or open calls.

If the driver wants to remove user access to the shared memory it created it makes the call:

```
un_shmem(usr_addr);
```

usr_addr is the address obtained by the SHMEM() call.

3.2 Removable Subdrivers

FlexOS has the ability to remove subdrivers. This feature is implemented through the standard user DVRUNLK command and supervisor INSTALL function. For example, the user enters the subdriver device name in the DVRUNLK command to remove the subdriver from a driver. Similarly, the programmer uses the INSTALL SVC with the option field set to 0 and the devname field set to the subdriver name address to remove a driver.

Subdrivers like drivers are set as removable or permanent in INSTALL flag bit 5. When the bit is set the subdriver is marked as removable; otherwise it should not be removable. Permanent versus removable install status is reflected in the DEVICE Table's INSTAT field. For subdrivers, the fields are defined as follows:

0x00 - Not installed
0x01 - Requires subdriver
0x02 - Owned by Miscellaneous Resource Manager
0x03 - Owned by another driver
0x04 - Optional subdriver

Drivers are informed to remove a subdrive through the SUBDRIVE function entry point. This entry point is now used both to associate and disassociate a subdriver. To indicate which operation to perform, bit 10 in the Access field is set as follows:

Bit 10: 0 = Install subdriver
1 = Uninstall subdriver

The remainder of the Access flags remain as defined in Table 4-4, INSTALL Flags in the FlexOS System Guide.

The driver should then do what's necessary to remove the subdriver. Note, however, that the driver can ignore the request, for example, if the subdriver is currently in use. The following sample code illustrates a SUBDRIVE routine that handles both installation and removal of the subdriver.

```
LONG    s_subdrvr(pb)
DPB     *pb;
{
    PHYSBLK     *d;

    if(pb->dp_flags & 0x400){
        sfree(sdev[pb->dp_option]);
        sdev[pb->dp_option] = 0;
        return((((LONG)DVR_PORT << 16) | (LONG)DVR_SER));
        }

    ser_unit[pb->dp_option] = pb->dp_unitno;
    pt_hdr[pb->dp_unitno] = (DH *) pb->dp_swi;
    pt_unit[pb->dp_unitno] = pb->dp_option;

    d = sdev[pb->dp_option] =
        (PHYSBLK *) salloc( (LONG)sizeof(PHYSBLK));
    d->Qrear = d->Qfront = d->evpend =
        d->xoffed = d->Qlen = 0;

    return(E_SUCCESS);
}
```

The return code from the SUBDRIVE function should indicate the type of subdriver required or 0, if no subdriver is required.

Typically, the SUBDRIVE routine is not the only portion of the driver involved in the subdrive interface. For example, you should also free the resources (for example, flags, pipes and memory for data structures) used to enable device I/O when the remove command is received. A general rule of thumb regarding subdriver removal is: Everything done in INIT and SELECT to support device I/O should be undone in UNINIT and FLUSH, respectively.

End of Section 3

386 Development Environment

4.1 Overall Considerations

FlexOS 386 uses the paging portion of the 386 chip for efficient memory management. It also supports a flat 32-bit program load format. Also, there is no longer a 64K limit on memory allocations when the S_MALLOC SVC is called by a 386 program.

There is a change in the exception numbers for the FlexOS EXCEPTION SVC as they correspond to the 80286 exception vectors. FlexOS EXCEPTION numbers 11 - 18 (for the condition "emulated instruction group 0") now correspond to 80286 vector number 7.

FlexOS 386 runs all 286 drivers and applications. In fact, FlexOS 386 Release 1.4 supports 286 mode drivers only.

4.1.1 Program Development Tools

In addition to the 286 Assembly Language Programming tools provided with FlexOS, there are new 386 model program development tools provided with this release that are based on the Common Object File Format (COFF). They are:

Tool	Description
CASM	386 Assembler
CLINK	COFF Linkage Editor
CLIB	COFF Librarian
CSID	386 Model Program Debugger
CTO	COFF to OBJ conversion utility

4.1.2 Reserved File Extensions

In addition to the reserved file extensions listed in the FlexOS Supplement for Intel iAPX 286-based Computers, the following extensions have been reserved for the 386 development environment.

Extension	Description
A	CASM Assembly language file
O	COFF 80386 object file
386	Command file that runs directly under the operating system in 32-bit mode

4.1.3 Entry Mechanism

Entry into FlexOS 386 by 386 programs is made by application code using INT 221 as the entry point with the parameter and return value as shown below.

Register	Contents
ECX	SVC number
EAX	Parameter block address
EAX (return)	Return Value

For a sample program illustrating the entry mechanism, see the COFF Programming Utilities Guide. Also described in that manual is the 386 application program memory model.

4.1.4 System Generation Utilities

FlexOS itself is compiled into OBJ format object files using the 16-bit addressing mode. Therefore, driver code must be compiled into the same format object files so that they can be linked with the system object files. The following 16-bit utilities are used to generate a FlexOS 386 system, to construct drivers, and to develop FlexOS 286-compatible applications. They are described in full in the FlexOS Programmer's Utilities Guide.

Utility	Description
RASM86.286	Relocatable Assembler
LINK86.286	Linker
LIB86.286	Librarian
SID86.286	Symbolic instruction debugger
FIX.286	Generates a relocated operating system image

End of Section 4

DOCUMENTATION ERRATA

5.1 FlexOS User's Guide

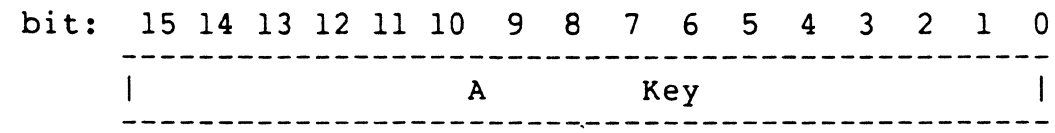
The following are errors or omissions in the FlexOS User's Guide, First Edition November 1986:

Page 2-2	Window 1 is the dedicated message window, and window 2 is the status window. The first window that actually shows is window 3. When you pull up the status window, you may not see them, but they are there at all times.
Page 2-5	The <HELP> key on the keyboard is actually CTRL-<INS>.
Page 7-7	Following the second paragraph in the explanation add the following: **Note:** Invoking a batchfile in the background causes a shell to be invoked, and the shell, in turn, runs the batchfile. Thus, the process ID returned is that of the shell. Therefore, batchfiles cannot be stopped with the CANCEL command, but the shell can be stopped.
Page 7-25	Omit the filetype CMD from the sentence reading ". . . with the extension 286, 386, 68K, CMD, COM, or EXE."
Page 7-32	The paragraph should read, "When you copy a file, the date and time of the source file are copied to the destination file's directory information."
Page 7-53	Delete the Note that states DISKCOPY will format an unformatted disk while copying. This is incorrect; DISKCOPY does not format an unformatted disk.
Page 7-79	In the example, remove the "Press any key to begin" line. LOGOFF does not issue any prompt.

Page 7-85	Omit the filetype CMD from the sentence "These files extensions include 286, 386, 68K, CMD, CMD, COM, and EXE."
Page 7-103	Omit "(a-p)" after current drive. There is no limitation on drive name.
Page A-1	Change the sentence "Use CONFIG.BAT tset up the LOGON/LOGOFF . . . " to read "Use CONFIG.BAT to set up the LOGON/LOGOFF . . ."

5.2 FlexOS System Guide

The figure on page A-3 of the FlexOS System Guide is incorrect. It should be:



5.3 FlexOS Programmer's Utilities Guide

FIX Cross-Reference Utility

FIX.286 is a system generation utility program that generates an output file containing a relocated FLEX286.SYS operating system image from a relocatable operating system file in standard .286 format.

FIX 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 command line), FIX does not create the GDT and IDT, which are used in protected mode only. FIX expects the OS Data Header to be the first item in the data segment.

FIX Command Syntax

FIX is invoked using the command form:

FIX input.fil output.fil [/r]

For example, to create the system boot loader BLOAD286.IMG, first assemble and link the BLOAD286.A86 file to create a BLOAD286.286 file, then enter:

FIX BLOAD286.286 BLOAD286.IMG /r

The BLOAD286.IMG file is then ready to be placed on the boot disk, using the SYS.286 utility, to create a bootable disk.

FIX also creates a protected mode system image from the relocatable file produced by the loader. For example, link the BOOT386.INP file to produce BOOTPROT.286, then give the command:

FIX BOOTPROT.286 FLEX286.SYS

The resulting image file FLEX286.SYS is copied to a disk with a bootloader image (BLOAD286.IMG) using the COPY.286 utility, or it is placed on a disk with the SYS.286 utility.

End of Release Note