FlexOS™ 286 Version 1.31

Release Note 01

May 1987

Contents

Section 1 tells you the contents of the disks, the necessary development environment, and how to boot up FlexOS from the floppy disks and then install it on your hard disk.

Section 2 tells you how to use the bootscript, how to link FlexOS modules, how to build a non-bootable system, and how to use the ATLSID debugger.

Section 3 describes two new features in Version 1.31, shared memory and removable subdrivers.

Section 4 describes three demonstration programs that show some of the capabilities of the Virtual Device Interface.

Section 5 lists the known problems in Version 1.31.

Section 6 contains a list of errors or omissions in the FlexOS documentation set.

1073-1001-002

GENERAL SYSTEM INFORMATION

1.1 FlexOS System Software

FlexOS 286 Version 1.31 software is shipped on six 5 1/4 inch quad density (1.12 Mb) disks organized into three distinct product assemblies: Software Developer Kit, Driver Writer Supplement, and OEM Redistribution Kit.

1.1.1 Disk Contents

Generally, the files are grouped into subdirectories according to their purpose or use. The README files list the files according to these groups and provide subdirectory definitions. In addition, explanations of the files for the MAKE utility and linker are provided in README. We recommend printing out and reviewing the contents of the README file as soon as possible.

Each product assembly has a set of files appropriate to the tasks that purchasers will need to do. The Software Developer Kit supports creation of applications that use the FlexOS SVCs and run under FlexOS. This package does not contain FlexOS source code, or license further distribution of FlexOS.

The Driver Writer Supplement supports development of hardware drivers for specific applications. The kit provides the FlexOS system object files, driver source, and link scripts. This package does not license further distribution of FlexOS.

The OEM Redistribution Kit consists of the disks in the Software Developer Kit and Driver Writer Supplement and a license to distribute FlexOS. You can also purchase FlexOS system source code; contact your Digital Research... sales office for further information.

1.2 Development Environment

Two executable versions of FlexOS are provided in the Software Developer Kit and OEM Redistribution Kit: one boots FlexOS from floppy disk, and the other boots FlexOS from a hard disk. These versions of FlexOS require this hardware to run:

- An IBM PC AT or a 100% compatible IBM AT clone with an E2 stepping of the 80286 processor (for DOS emulation)
- A 20 or 30 Mb hard disk with up to 4 partitions
- A Quad density (1.2 Mb) 5 1/4 inch disk drive
- A serial port with the same adress as COM1 under MS-DOS™
- A centronics-compatible port with the same address as PRN under MS-DOS
- 1.1 Mb of RAM allocated as follows:
  - 512Kb on the mother board
  - 128Kb low (mapped by hardware switches to be contiguous with the mother board 512Kb)
  - 512Kb high (mapped by hardware switches to 1Mb)
- An IBM Color Graphics Adapter (CGA), monochrome mode only, or
- An IBM Enhanced Graphics Adapter (EGA), low resolution only

Optionally, a Zenith® Z-29 terminal connected to the serial port, if you want to develop multi-user applications

1.3 Booting FlexOS

Turn on the computer, then insert the disk marked

Developer Kit #1

in drive A and close the door. The system loaded includes the DOS front end (MS-DOS version 2.1 compatible) and the beta version of the FlexOS multitasking Virtual Device Interface (VDI). 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 file transfers system files onto your hard disk so you can boot from the hard disk rather than from a floppy. Simply run the INSTALL file or files. As with the README files, there are three different INSTALL files. The OEM Redistribution Kit requires running of all three of these INSTALL files to create a complete system. If you are unsure of which INSTALL file you need, print out the README file on the first disk. It will list the INSTALL file or files you need to run.

WARNING

You must have a 20 or 30 megabyte hard disk to develop FlexOS 286. FlexOS allows you up to four partitions on a 30 Mbyte hard disk. However, if you purchased the OEM source kit, you must reserve one partition of 20 Mbytes for FlexOS Accordingly, if you have a 20 Mbyte hard disk and you purchased the OEM source kit, it must be unpartitioned.

Furthermore, 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

GENERAL SYSTEM INFORMATION

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 add serial ports. Refer to listing 3.1 on page 3-9 of the FlexOS System Guide for an example of a bootscript.

The basic bootscript is a user-editable file named CONFIG.BAT. The distribution bootscript is on Developer Kit disk #1.

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

Refer to the FlexOS User's Guide to modify CONFIG.BAT, and refer 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 optional DOS front end and the VDI are built from their own libraries.

The Driver Writer Supplement provides the object sources and sample input files for building all these libraries, and for building the DOS front end and the VDI. Refer to the README files for for discriptions of input and object files.

2.2.1 Linking FlexOS Modules

The files DISP286.OBJ, CONFIG.OBJ, ACONFAT.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 Driver Writer Supplement provides linker input files for creating bootable and non-bootable versions of FlexOS. Link the system using the ATLINK.INP input file to create a non-bootable system for debugging drivers.

There are four linker input files for making different types of FlexOS systems:

Filename	Type of system generated
BOOTAT.INP	FlexOS with DOS front end and VDI
BOOTALL.INP	Complete FlexOS system with DOS front end and VDI
BOOTFE.INP	FlexOS with DOS front end only
BOOTGIF.INP	FlexOS with VDI only

To build a bootable system, process the output file from the linker with the FIX286.286 utility, then write the resulting file to the boot disk using the SYS.286 utility or FORMAT.286 under FlexOS.

2.3 Building a Non-Bootable System For Debugging

First, link the system with ATLINK.INP. This creates RTM.CMD, a loadable version of FlexOS, and RTM.SYM, its corresponding symbol table. Next, boot Concurrent DOS (either version 4.1 or 5.0) and execute ATLD.CMD under it. ATLD automatically loads the debugger ATLSID.CMD, which in turn loads RTM.CMD and RTM.SYM for execution. The debugger executes from the attached serial terminal; Concurrent DOS runs on the primary console.

Note: The non-bootable system runs the debugger on the serial terminal, so it is necessary to connect it.

The version of ATLD provided runs under Concurrent DOS version 4.1 on an IBM PC AT equipped with one quad density floppy disk drive and one 20 megabyte hard disk. If your hardware configuration differs from this, or if you want to run ATLD.CMD under Concurrent PC-DOS version 5.0, refer to the file ATLD.DOC for an explanation of the procedure to follow.

2.4 Using ATLSID

The ATLSID debugger is similar to the SID 286 debugger, and you will recognize many of the same commands from that utility.

Invoke the ATLSID program by first booting up your developmemt system, then type

ATLD

The ATLD loader looks for the RTM.CMD and RTM.SYM files, loads ATLSID at segment 0A00, then loads FlexOS 1.31. at segment 0A30.

Here is a list of the currently implemented ATLSID commands:

Display Memory

b<address>,<length>,<address>	Compare memory
d<address>,<address>	Display bytes
dw<address>,<address>	Display words
dll<address>,<offset>,<size>	Display linked list by bytes
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
y	Reboot

2.5 Adding Drivers

Digital Research® provides driver source code in the Driver Writer Supplement and OEM Redistribution 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 Version 1.31 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

NEW FEATURES IN VERSION 1.31

3.1 Shared Memory

Shared memory lets multiple processes share memory in the same manner that Fortran programs use a COMMON routine. Processes can also access specific physical memory locations, for dual ported RAM or system ROMs.

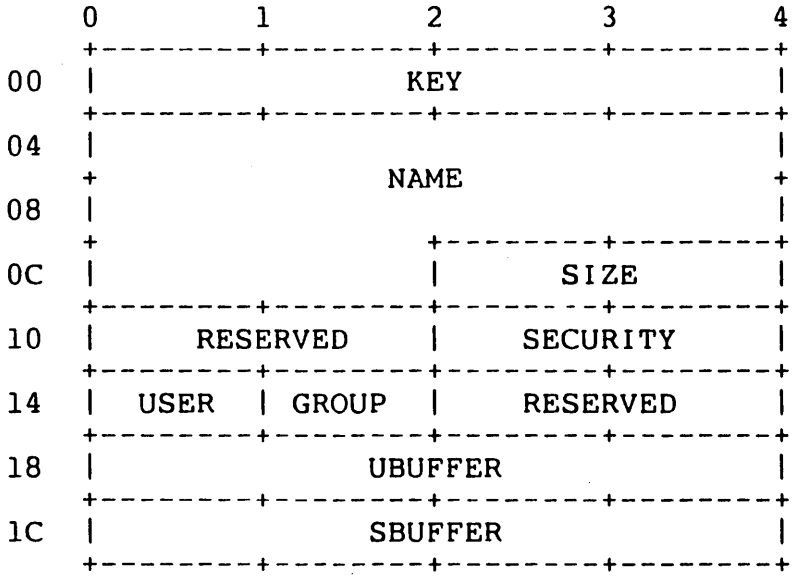
The processes can share data regions with drivers for fast communications in both protected and unprotected FlexOS environments, and multiple user processes can share data regions with each other. FlexOS grants access to shared memory only to those user processes with access rights established during 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 Data File specifying a name for the memory allocation, a security word, and the size of the memory. A subsequent OPEN SVC provides and verifies access to this file. The GET SVC returns a valid address for the Shared Data Region, and the CLOSE SVC disables access via this address.

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 or if Read and Write access is not available. Different processes may obtain unique addresses of the same physical memory, which is only used by the specific process or if the process is currently mapped into context through the MAPU() driver service.

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

Before a user process can share memory with other user processes, a driver must map physical memory into system space through the MAPPHYS() or SALLOC() driver services. Next, this system memory is mapped to user space through the SHMEM() driver service. (The region is released with the UN_SHMEM driver service.) This gives a user process direct control of memory related devices.

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 Data Regions

A user process uses shared memory through shared data files, which are managed by the Pipe Resource Manager and accessed through the device name "sm:".

To create a shared data 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 data.

If another user process wants to use the above shared data region 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 data 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 now have to 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 data region 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 data file obtained through the CREATE or OPEN calls.

When it no longer needs access to the shared data the user process would the make the call:

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

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

If the driver wanted to remove user access to the shared data it created it would make the call:

```
un_shmem(usr_addr);
```

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

3.2 Removable Subdrivers

FlexOS 286 version 1.31 adds 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 - Permanent subdriver
0x04 - Removable 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

VDI DEMO PROGRAMS

Developer Kit disk #2 contains three programs that demonstrate the capabilities of the Virtual Device Interface (VDI).

The three demo programs are:

thing [**-l** <# of points>]

This program draws a simple line-figure; it is useful for testing lines and line colors. The **-l** switch tells **thing** to draw a design with the specified number of points around the perimeter. By default, **thing** draws the figures with from four to ten points around the perimeter.

polyline

This program is documented in the GEM VDI reference manual.

logos

This program draws a number of Digital Research logos on the screen, using the copy transparent function, draws DRI's slogan over that using a one of the fonts loaded, and then cycles the colors on 16 color screens (on an EGA device).

All of these programs have the following switches in common:

-f device	Sends output to the specified device instead of the screen.
-m	Emulate monochrome on color devices.
-c colors number	Set the maximum number of colors to the argument. If you specify too many colors, the number defaults to the maximum available. **-m** overrides this switch.

End of Section 4

KNOWN PROBLEMS

This section contains a list of known problems in FlexOS 286 Version 1.31 as of the time of release. The list is arranged in the order of severity where the severity levels are defined as follows:

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

5.1 Severity 3 Problems

There are no known severity 3 problems in FlexOS Version 1.31.

5.2 Severity 2 Problems

Problem: Asynchronous non-destructive F_READ of keyboard enables keyboard wait state.

An async non-destructive read of the keyboard should return immediately to the calling process; instead, it acts as if it were a synchronous read and waits until a key is pressed before returning. It should return with an event mask.

Problem: Failure of pipelines

The command line:

```
'DIR | FIND "BAT" > TEST'
```

fails to produce complete output in the file 'TEST'. The problem is in the SHELL (COMMAND.286), and the workaround is to add another piped command to the command line. For example:

'DIR | FIND "BAT" | FIND "BAT" > TEST'

Problem: Day of week entry in the TIMEDATE table is incorrect.

Prompt $D returns the wrong day.

Problem: Failing DVRLOAD does not free driver memory.

When an attempt is made to load a device driver, memory is allocated in the system memory area before the driver code is read. The resource manager does not deallocate the memory when it rejects the driver on an error condition.

Problem: FORMAT/FDISK incompatibility.

If you attempt to FORMAT a hard disk partition made using the FDISK utility, you may obtain unpredictable results. Physically format the hard disk before partitioning the disk with FDISK.

Problem: SYS.286 utility searches wrong root directory for the BLOAD286.IMG.

If SYS.286 is executed from a subdirectory of a bootable device that is not the device the current system booted from, SYS.286 attempts to access the wrong root directory.

Problem: SYS.286 overwrites files.

If you invoke SYS.286 in this syntax:

SYS C: A:YOURFILE.SYS

SYS copies YOURFILE on A: onto the C: drive, but it also transfers any copy of FLEX286.SYS that might be on the A: drive, and it writes this copy of Flex286 over YOURFILE.SYS. Use the COPY.286 utility instead.

Problem: SYS.286 overlooks COMMAND.286.

SYS.286 does not transfer COMMAND.286 to the new system disk; it must be done manually.

Problem: SYS and FORMAT -S do not copy all the files marked with the hidden, system, and R/O attributes.

Copy the remaining files separately.

Problem: XON/XOFF does not function correctly in the serial port driver, SDRV.DRV.

After a user or the system sends XON/XOFF multiple times to the serial port driver, either XON/XOFF stops working or a system General Protection Error occurs.

Problem: LOADER shared-code checking insufficient.

When executing a program that is designated as shared-code, the source can be compiled and linked and re-executed inadvertantly while the first copy is executing. In such a case, the loader will use the old program's code and the new program's data. This can result in anything from incorrect results to a catastrophic system failure.

Problem: No recovery of file size after a system crash.

If a system crash occurs during a file update, the size of the file as indicated in the directory may not reflect the actual file size.

Problem: S_COMMAND SVC CHAIN option does not use PINFO data.

The S_COMMAND SVC CHAIN option does not use the NAME and PRIORITY information from the PINFO parameter block. These fields should replace the existing NAME and PRIORITY fields in the Process Descriptor.

Problem: Passing bad array pointers or structure pointers to the VDI can cause a GP error.

If you pass bad pointers to the Virtual Device Interface the most likely result is a system failure (General Protection Error). Therefore, take care to ensure that all pointers are valid.

5.2.1 Problems in Beta VDI

EGA - VDI interaction

If you are running multiple color VDI programs under EGA, the color registers are not saved correctly during a screen switch.

Loading the mouse driver

If you have a loadable mouse driver, load it in CONFIG.BAT. If you load it later, only the child virtual consoles receive a mouse.

Virtual console borders

When you close a virtual console with borders, some of the borders may remain on the screen.

Writing to the borders of the top virtual console has no effect until you make it the top console again.

You can open border files for a virtual console even if no border files were defined.

Virtual console placement

When you create a virtual console, it appears on top of the other consoles, even though it is not the active virtual console and should have no view on the physical screen.

PRINTSCREEN function VCCREATE support

When the Window Manager starts the shell running during PRINTSCREEN, it needs to obtain the virtual console file number to transfer the screen buffer contents to the MALLOCed memory buffer. In this one case, the COPY/ALTER call will accept the file number returned by VCCREATE.

Cursor tracking in windows

Cursor tracking in windows does not work as it is documented in version 1.31.

RWAIT rectangles

You can wait for RWAIT rectangles outside the size of the parent virtual console.

Virtual consoles outside parents

You can position virtual consoles outside their parents. In release 1.31 they are not clipped, but they will be in future versions.

MCTRL

The MCTRL is not implemented in version 1.31.

GENERAL CAUTION

We strongly discourage you from running PC-DOS programs that use command line I/O redirection.

End of Section 5

DOCUMENTATION ERRATA

6.1 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 IBM AT 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, 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.

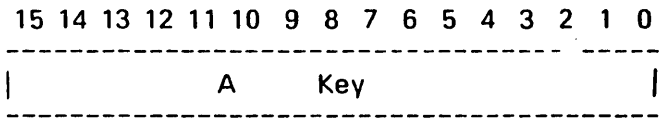
Page 7-85 Omit the filetype CMD from the sentence "These files extensions include 286, 68K, CMD, CMD, COM, and EXE."

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 . . ."

6.2 FlexOS 286 System Guide

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



6.3 FlexOS 286 Programmer's Utilities Guide

FIX-286 Cross-Reference Utility

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

FIX-286 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, which are used in protected mode. FIX-286 expects the OS Data Header to be the first item in the data segment.

FIX-286 Command Syntax

FIX-286 is invoked using the command form:

FIX286 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:

FIX286 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-286 also creates a protected mode system image from the relocatable file produced by the loader. For example, link the BOOTAT.IND file to produce BOOTPROT.286, then give the command:

FIX BOOTPROT.286 FLEX286.SYSH

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