The FDI Directory Device Driver

This version 2, of the FDI device driver is for use with standard QDOS floppy disk image files. The driver uses the image file as if it was a real floppy disk. So you can use all the usual disk commands, SAVE, LOAD, DIR etc.

It is based on the device level 3 FLP driver from SMSQ/E (version 3.16) source code.
(see licence notice at the end of this document)

Version 2 of the FDI driver supports Single, Double, High, and Extra High density disk images.
It also supports sub-directories, and many of the features of a Level 3 device driver.
However it does not support DOS disk images.

Installing the Driver

Two versions of the driver is supplied. A RAM based one, and a 16K byte ROM image.

To load the RAM based version of the driver, load the driver into memory and call it.

```
example:   i.  LRESPR flp1_FDI2driver_cde
           ii. x=RESPR(14734)                {if you don't have Toolkit 2, or equivalent}
               LBYTES flp1_FDI2driver_cde,x
               CALL x
```

An installation message, and a version number will be displayed in #0

To load the ROM based version, see the user instructions of your emulator. For example in QPC2 use the command:

```
EPROM_LOAD flp1_FDI2driver_rom
```

Using the Driver

Before the FDI device driver can use an image file. It must first be mounted, onto one of the eight available drive slots. See the **MOUNT_FDI** and **UNMOUNT_FDI** commands for associating image files with drive numbers.

To create new, blank formatted FDI image files. See the **MAKE_FDI** command.

To convert actual floppy disks into FDI images, see the ReadMe document supplied with the device driver.

Formatting FDI devices

When the **FORMAT** command is used on an image file. It will format the image to it's current size. So if you have an HD image file mounted, it will format it to 2880 sectors. Note, adding the format density directives to the end of the format name will have no effect.

If you require an image file of a particular size, use the **MAKE_FDI** command below.

If a Format fails, the image file is unmounted, as it's condition is unknown, and it would not be safe to continue to use it.

MOUNT_FDI
FMOUNT_FDI

The **MOUNT_FDI** command is used to associate a floppy disk image file to one of the eight available drive slots. The optional protect status parameter determines whether or not, the image file will be write protected. The default being 0, for not write protected.

FMOUNT_FDI is a function version of **MOUNT_FDI** that returns 0, or an error code, without stopping the running program.

syntax: *fdi_no* := *numeric_expression* {1 to 8}
protect := *numeric_expression* {0 or 1, default 0}

MOUNT_FDI *fdi_no*, *filename* [,*protect*]
FMOUNT_FDI(*fdi_no*, *filename* [,*protect*]**)**

example:
i. **MOUNT_FDI 1,win2_Quill_img**
ii. **MOUNT_FDI 3,win1_Games_img,1** {image is write protected}
iii. **result=FMOUNT_FDI(4,dos1_Xchange_img)**

comment: Once an image file has been mounted, it appears to the system to be an ordinary directory device. So you can LOAD, SAVE, DELETE etc.

UNMOUNT_FDI
FUNMOUNT_FDI

The **UNMOUNT_FDI** command is used to disassociate a floppy disk image file with its drive slot.

It is equivalent to removing a floppy disk from it's drive.

FUNMOUNT_FDI is a function version of **UNMOUNT_FDI** that returns 0, or an error code, without stopping the running program.

When an image file is unmounted, The FDI driver will attempt to tidy up behind itself by closing any open channels on the FDI device, and removing the drives Physical Definition Block from memory. It will not close any programs that have open channels to the device, but if the program attempts to access the device, an 'invalid channel ID' error may occur.

syntax: *fdi_no* := *numeric_expression* {0 to 8}

UNMOUNT_FDI *fdi_no*
FUNMOUNT_FDI(fdi_no**)**

example:
i. **UNMOUNT_FDI 3** {dismounts the image file connected to slot 3}
ii. **result=FUNMOUNT_FDI(1)**

comment: In the special case of **UNMOUNT_FDI 0**, all mounted image files will be dismounted.

FDI_USE **directory devices**

FDI_USE allows renaming of the FDI device. **FDI_USE** without a parameter will reset the name of FDI back to FDI.

syntax: **FDI_USE** [*name*]

example:
i. **FDI_USE mdv : LOAD mdv1_prog** {loads 'prog' from FDI1_}
ii. **FDI_USE** {the driver now uses FDI again}
iii. **MOUNT_FDI 1,win2_Quill_img**
FDI_USE mdv
LOAD mdv1_boot {the Quill Image file acts as if it was in mdv1_}

FDI_FILE$

The **FDI_FILE$** function returns a string containing the filename of a mounted image file. If no image file is mounted, **FDI_FILE$** will return an empty string.

syntax: *fdi_no* := *numeric_expression* {1 to 8}

FDI_FILE$ *fdi_no*

example: **PRINT FDI_FILE$ (1)** {print the name of the image file of FDI1_}

MAKE_FDI
FMAKE_FDI

The **MAKE_FDI** command will create a blank formatted disk image file.

The density parameter defines the size of the disk image to be created.

s - Single density, 360K, 720 sectors
d - Double density, 720K, 1440 sectors
h - High density, 1.4M, 2880 sectors
e - Extra density, 3.2M, 6400 sectors

FMAKE_FDI is a function version of **MAKE_FDI** that returns 0, or an error code, without stopping the running program.

syntax: *mediun_name* := *name* | *string_expression* {maximum of 10 characters}
density := *name* | *string_expression* {**s** | **d** | **h** | **e**}

MAKE_FDI *filename*, *medium_name* ,*density*
FMAKE_FDI(*filename*, *medium_name* ,*density***)**

example: i. **MAKE_FDI win2_myfiles_img,Games,d**
{creates an image file named 'myfiles_img' on flp2_, With a medium name of ‘Games’, and 1440 Sectors}

ii. **MAKE_FDI win1_Work1_img,Data,s**
{creates an image file named 'Work1_img' on flp1_, With a medium name of ‘Data’, and 720 Sectors}

iii. **result=FMAKE_FDI(“dos1_Games.img”,Vol1,d)**

comment: This command will only create an Image file, it does not mount it.

Copyright and Disclaimer

This driver should not cause any problems, damage, or loss of data. However by using this device driver, you do so at your own risk, and I do not accept responsibility for any damage, or loss of data. You should always only work on copies of important disk images.

The driver also contains portions of the SMSQ/E source code