'MICRO SERVICE'
Fabritiusstraat 15
6174 RG Sweikhuizen

Rev. 1.0
Febr. 1985

F D D T O O L K I T - 1

programs for the DAI-DOS 1541

1. COPDIR. #400-43F

This program consists of a object code part and a BASIC part. It must be loaded with UBL"COPDIR". It enables printing of the contents of the directory on a parallel printer. The directory cannot be printed directly due to reasons described in the DAI-DOS 1541 manual.

The directory is read from disk into the free RAM space. Then the filenames are stored in an array DIR$(144). Each time the '<' symbol is displayed on the screen, the spacebar has to be pressed in order to read the next page of the directory.

The contents of DIR$(144) can be printed on a parallel printer, or can be sent to the serial outlet.

Furthermore, the array DIR$ can be saved on disk for use in other programs.

2. READ.CBM. #400-78F

This program is also a combination of object code and BASIC. It must be loaded with UBL"READ.CBM". With this program, PRG-files from a CBM64 diskette can be transferred to DAI BASIC.

Via a menu, the following possibilities exist:

1 - load CBM64 file into edit buffer
2 - update contents edit buffer
3 - move contents edit buffer to DAI program buffer

Under (1), the file name as displayed in the directory listing must be entered. Then the program is read, and moved into the edit buffer. The CBM64 tokens are replaced by the BASIC command/function strings.

Some reservations must be made:

The CBM64 uses control characters for screen handling, which are identical to the tokens for BASIC statements/functions. The translator replaces these control characters by the particular BASIC words. Furthermore, the CBM64 uses BASIC commands, which are not available in the DAI-BASIC.

Under (2), the contents of the edit buffer can be adapted, in order to make it useable for the DAI. The editor can be left by pressing the <tab>-key.

Under (3), the program is moved into the DAI program buffer. This has to be done by CALLM #409.

Afterwards - BASIC V1.1 only -, the program can be started again with:

- RUN 60000 : The contents of the edit buffer is destroyed.
- RUN 60010 : The contents of the edit buffer is still available.

Because the BASIC part of the translator starts with linenumber 60000, it can easily be separated from the CBM64 program, after it is moved into the DAI program buffer.

3. ON ERROR GOTO. # 300-300

This object code program is an extension of the routine published in the DAInamic Newsletter 83/p.173. It is adapted for the DOS error handling. A demo is added to explain its use. Both parts can be loaded with UBL"ON ERROR GOTO". If only the object code part is loaded into the DAI, the pointer $029B/C must be updated till after this object code program.

After loading the object code program, the RST3 interrupt vector address has to be changed:

```
$UT
>V5 C6FD-300
>B
```

To enable the use of this 'on BASIC error' routine, it must be activated with:

```
$POKE 6,LINENUMBER IAND $FF:POKE 7,LINENUMBER SHR 8
```

LINENUMBER is the BASIC program line to go to if an error in a BASIC program occurs.

The error routine can be disabled by:

```
$POKE 6,0: POKE 7,0
```

4. FDD-MONITOR. #400-5BF

This object code program (FDDMON/O) contains an addition to the monitor in the DAI-DOS 1541. With the routines of the FDD-monitor, it is possible to use all the extra options of the VC1541 disk drive.

The most interesting of these options are:

Handling relative files.
Use of direct access commands.

Several demo programs are added to explain the use of this FDD-monitor.

The VC1541 disk drive uses up to 15 channels for data transfer to the outside world. Channels 0 and 1 are normally used by CBM64/VIC20 programs. Channel 15 is the command/error channel. For DAI programs via the DAI-DOS 1541, always channel 2 is used. Channels 3 and 4 are used by the FDD-monitor. Channel 3 is generally used for relative files, channel 4 for direct access commands.

Are all files opened and closed automatically by the DAI-DOS 1541. For the FDD-monitor this opening and closing must be made exclusively. The demo programs will explain it. Also the error channel is not read automatically. This can be done in your BASIC program by CALLM #F2E8.

The following entry-points to the FDD-monitor can be used from BASIC programs:

#400	OPEN3	Open channel 3 for relative files
#403	OPEN4	Open channel 4 for direct access
#406	PRNT3	Channel 3: Send string with CR added
#409	PRNT4	Channel 4: idem
#40C	PRNT15	Channel 15: idem
#42A	PUT3	Channel 3: Send string without CR
#42D	PUT4	Channel 4: idem
#430	PUT15	Channel 15: idem
#40F	GET3	Channel 3: Read one character
#412	GET4	Channel 4: idem
#415	GET15	Channel 15: idem
#418	INP3	Channel 3: Read string until CR
#41B	INP4	Channel 4: idem
#41E	INP15	Channel 15: idem
#421	CLOSE3	Close channel 3
#424	CLOSE4	Close channel 4
#427	GETDIR	Read directory into RAM

For a good explanation of the use of relative files and direct access commands and the way they are handled, please refer to 'Das grosse Floppy Buch', published by Data Becker GmbH in Duesseldorf. In this book, an extended explanation is given. It gives a much better explanation than the one in the VC1541 manual.

4.1. Relative files:

Relative files are handled via channel 3. At first, a relative file must be opened:

```
FILENAME$=<filename>
RECLENGTH=<length of the record string>.   Max.254.
OPEN$=FILENAME$+",L,"+CHR$(RECLENGTH)
CALLM OPEN3,OPEN$
```

In order to store data in a relative file, in the VC1541 a pointer must be set, pointing to the required record. This positioning is done as follows:

```
RECNR=<number of required record>
HB=RECNR SHR 8: LB=RECNR IAND 255
BYTE=<number of byte in the record to start at>
POS$="P"+CHR$(3)+CHR$(LB)+CHR$(HB)+CHR$(BYTE)
CALLM PRNT15,POS$
```

Now the data can be read from the record:

```
IP$="                ".  The string length must be >= RECLENGTH !!
CALLM INP3,IP$
```

Now the data read is in the variable IP$.

To write a record to the diskette, the following format has to be used:

```
CALLM PRNT15,POS$
```

```
OP$=<string to be send>     The stringlength must be RECLENGTH-1 !!!
CALLM PRNT3,OP$
```

The file can be closed by using:

```
CALLM CLOSE3.
```

See the program 'DEMO REL.FILES' for a demonstration of the use of these commands in a program. This program can be used only in combination with 'FDDMON/0'.

4.2. Direct access commands:

For direct access commands, channel 4 is used. Here again, command strings are send via channel 15, and data handling is done via channel 4.

Open channel:	OPEN$="#" CALLM OPEN4,OPEN$	
Send command:	CALLM PRNT15,OP$	CR is added
Send data:	CALLM PUT4,OP$	No CR is added
Read one character:	IN$=" " CALLM GET4,IN$	
Read a string :	IP$=" CALLM INP4,IP$	"

The demo programs will explain the use of these commands in detail.

5. DEMO B-R.

The demo program 'Block-Read' must be used in combination with the FDD monitor. It makes a list of all sectors on the diskette used by a certain program, if the fist track/sector is known.

After opening the direct access channel, the first track and sector must be entered. Try track 18, sector 0, for the directory.

Via OP$="U1 4 0"+TRACK$+SECTOR$: CALLM PRNT15,OP$ the pointer is set to the given track/sector. Here, the next track and sector can be found. They are read via GET4. As long as the tracknumber read is <> 0, the program conti-nues reading the the next track/sector.

In this way, a listing of all sectors used by a certain program can be made.

6. DEMO B-W.

To be used in combination with 'FDDMON/0'!!

The direct command channel is opened by sending '#' to the drive. This reserves a buffer in the drive for direct access operations.

This program changes the name of a diskette. A new name can be entered. If the name is shorter than 16 characters, the string is filled up with #A0 (space with msb set).

The info 'block-read, track 18, sector 0' is send via the string OP$="U1 4 0 18 0". The pointer is set to the first byte of the file name by sending OP$="B-P 4 144". The new name is send via the PUT4 routine, because no CR is allowed after the name.

At the end, the drive is initialised again by sending the 'I0' command to the drive. In this way, the BAM is updated with the new name too.

7. DEMO B-P.

Only to be used in combination with 'FDDMON/0'!!

This demo reads the diskettename directly from the diskette.

The direct access channel is opened, the 'B-R' command is given, and the buffer pointer is set to the beginning of the diskette name on track 18, sector 0, byte 144 by:

```
U1 4 0 18 0    Block-read - channel 4 - drive 0 - track 18 - sector 0
B-P 4 144      Buffer-pointer - channel 4 - byte 144
```

Via GET4, byte after byte the name is read, until 16 characters are read, or a space (#A0) is found.

8. DEMO M-R.

Only to be used in combination with 'FDDMON/0'!!

This demo reads the diskettename from the BAM, which is in the VC1541 RAM memory.

After initialisation, the filename can be found on the addresses #790-#79F in the VC1541 RAM area.

The drive is initialised by sending 'I0'. The command to read the memory for 16 bytes is given by:

M-R + CHR$(loaddr) + CHR$(hiaddr) + CHR$(nr of bytes)

Then the string can be read via CALLM INP15,IP$. The variable IP$ must be set to a long enough empty string before !

9. DISPLAY SECTOR.

Only to be used in combination with 'FDDMON/0'!!

With this program, it is possible to display on the screen the contents of a block of 256 bytes on the diskette.

After entering the required track and sector, the block is displayed by using direct access commands for 'B-R'. Note, that all numeric data is transferred into a string before sending it to the drive !

If you try e.g. track 18, sector 1, the first block with directory entries will be displayed. The first two bytes indicate the track/sector of the next block of the directory.

10. CHANGE DRIVE NR.

Only to be used in combination with 'FDDMON/0'!!

Via this program, the addressing of the disk drive can be changed. This can be useful, if temporarily two or more drives are connected to the system. Switch on drive 1 first, set it via this program to the hardware address 9 (FDD1) and then switch on drive 0 (=FDD0 with address 8).

This program is at the same time a demo for the Memory-Write commands.

The datastring sent to the drive consists of:

M-W + CHR$(loaddr) + CHR$(hiaddr) + CHR$(nr of bytes) + the bytes to be send (also in CHR$'s)

The pointers in which the drive number in the VC1541 RAM are kept, are #77/78.

After changing the drive number, the BAM must be initialised again by sending 'I0'.

11. LOCK/UNLOCK.

Only to be used in combination with 'FDDMON/0'!!

It is possible, by means of this program, to protect files on a diskette against scratching. In the directory entry of the file, one bit can be set, indicating a scratch protection.

Via this program, it can be done for all files, for one file, and 'on request' by stepping through the directory.

With the same program, the protection can be cancelled.

When stepping through the directory, the ESC-key (cursor left) can be used to abort the program.

12. DEMO REL.FILES.

Only to be used in combination with 'FDDMON/0'!!

See also the explanation given in chapter 4 on the addressing of relative files. A good explanation of the set-up of the relative files can be found in 'Das grosse Floppy Buch'.

A relative file is opened by sending:

```
OPEN$=filename$+",L,"+CHR$(recordlength)
```

If #FF is written in the last record, then the access to the separate files will be much quicker:

```
POS$="P"+CHR$(channel)+CHR$(lonr)+CHR$(hinr)+CHR$(1)
OP$=CHR$(255)
```

```
CALLM PRNT15,POS$:CALLM PRNT3,OP$
```

The pointer is set to the highest record, to byte 1 and #FF is written in this record. This will result in a 'FILE NOT PRESENT' message. This is not a real error message, but happens always if a record is addressed which has a higher number than the last one in which has been written.

Data is read from the record by setting the pointer to this record in the manner given above, and reading the string with CALLM INP3,IP$.

Data can be send to a record via:

```
CALLM PRNT15,POS$
CALLM PRNT3,OP$
```

Before, the string OP$ must get the correct length. See the lines 900-930 for it.

Remember here too, that only strings can be handled. That means, that all numeric values must be converted to strings before sending them to the drive !! All data of one record must be put together in one concatenated string.