Toolkit 7

The programs from TOOLKIT 7

-Ø VPA.DOC	documentation program for VPA
-Ø FWP.DUBBELKOL	the program for double coloms with FWP
-1VPA.MLP	the machine language program VPA
-PROGRAMMOTEEK	
-BS.DEMO.VPA	A demo program for VPA
-1MAGNIFIER.MLP	the mlp "new zoom"
-1digi	digitiser program for commercial unit
-&DIGI.SRC.SPL	the source program in SPL-format
-ØMIREDETEST	a test program for digitiser
-ØDIGITISER	The digitiser program fromP.DE LAET
-ØREGLAGE	the tunung program for P.DE LAET digitiser
-$MSX	the MSX-extensions for DBASIC

DAInamic redaction & software
c/o W. Hermans
Mottaart 20
3170 Herselt

MSX-EXTENTIONS

INTRODUCTION

The MSX-extension (a DBASIC-extension) is an utility for a two-way-file transfert between the DAI pc and any MSX home computer.
Virtualy all kinds of files can be transfered : basic programs, text files, binary files.

CONNECTING BOTH COMPUTERS

Data transfert is done by the audio-cassette input-output ports of both computers.
This means :
- no special hardware is required
- data can be transfered from one computer to another by only connecting the audio-cassette I/O ports.However, without signal amplification, you could get reading errors on the DAI pc because the outputlevel of the MSX computer is rather small.
When amplifying the signal (by means of your audio-cassette recorder), you'll have a very reliable file transertsystem.
- data can be transfered without connection between the computers by means of an audio-cassette.

THE SIGNAL

The electrical signal which represents the data stream, generated by the DAIpc is exactly the signal MSX computers use as standard.(i.e. a 1200 baud signal, 1 startbit, 2 stopbits lsb first)
This means :
- no special software is needed on the MSX computers
- your DAI pc can read all commercial, available MSX software from audio-cassette.

STARTING FILE TRANSFERT

Before any file tranfer between DAI and MSX can take place, load the software on DAI :i.e.
- load DBASIC V2.2 as described in 1.1 of the DBASIC V2.1 manual.
- extend DBASIC with the MSX instructions : $EXTEND "MSX"

STATEMENTS AND FUNCTIONS

notation :
filename :- a string of 1 to 6 alpha-
numeric characters
- filenames of more than 6 characters will
be truncated
note : for other format notations see DBASIC V2.1 manual.

MSXBLOAD

Format : MSXBLOAD $filename , begin address
With : filename = $-expression
Valid : command/statement
Purpose : the contents of the binary file filename will
be loaded in memory beginning at address
beginaddress.
When omitting filename , the next binary file
will be loaded.
When omitting beginaddress the file will be
loaded at MSXPTR.(see MSXPTR)

Remarks : The first 6 bytes read are :
- byte 1-2 : MSX begin address = the address
where the file would be loaded in an MSX
computer
- byte 3-4 : MSX end address = the address up to
where the file would be loaded in an MSX
computer
- byte 5-6 : MSX start addres = the address where
program execution would start on an MSX computer
(! see also MSXBSAVE !)
Example : * MSXBLOAD "MONMSX",#5000
* MSXBLOAD,#5000

MSXBSAVE

Format : MSXBSAVE filename , begin address ,
end address , startaddres ; offset
Valid : command/statement
Purpose : To write the contents of the DAI memory
locations begin address to end address
exclusive in the MSX binary file filename .
If startaddress is ommited then the start-of-
execution address will be equal to the begin
address.
The offset will be added to beginaddress ,
end address and startaddress to define the
MSX begin address, end address and start
address, they will be written at the begin of
the file.
Remarks : see also MSXBLOAD
Example : *MSXBSAVE "test",#4000,#4200,#4100,#8000
will write DAI memory from#4000 to#4200
exclusive to the MSX binary file "TEST".
On the MSX computer, with
BLOAD"TEST",R
this file will be loaded in memory
#4000+#8000=#C000 to #4200+#8000=#C200
and execution will start at
#4100+#8000=#C100

MSXDUMP

Format : MSXDUMP beginaddress , endaddress
Valid : command/statement
Purpose : To write the comtents of the DAI memory locations beginaddress to endaddress exclusive in an MSX file block.
Remarks : - An MSX tape file contains 2 or more blocks each separated by a synchronisation signal.
- see also MSXLOG
Example : *MSXDUMP#5000,#5010

MSXLOAD

Format : MSXLOAD filename
Valid : command
Purpose : To load a MSX basic program, which was saved in ASCII format, and interpret it by Dbasic.
Remarks : - No MSX basic keywords will be converted to Dbasic keywords.
- MSX basic keywords, not known by Dbasic, will be considered as a procedure or will generate a syntax error.
- Screen coordinates will not be converted
- See also MSXSAVE.
Example : *MSXLOAD "SIMON"
will load the MSX basic program "SIMON"
MSXLOAD
will load the next MSX basic program.

MSXLOG

Format : MSXLOG beginaddress
Valid : command/statement
Purpose : To load the contents of an MSX file block into memory starting at beginaddress, when beginaddress is ommitted, the block will be loaded at MSXPTR (see MSXPTR).
Remarks : - see also MSXDUMP.

MSXPTR

Format : MSXPTR
Function : Returns the address where a MSX file or MSX file block will be loaded when giving a MSXBLOAD,MSXLOG or MSXTLOAD command.
Remarks : When address 0 is returned data will be loaded in the free ram space, just behind the symbol table.
Example : ...
100 PRINT MSXPTR
110 MSXLOG
120 PRINT MSXPTR
....

Op de MSX:
type
LOAD"CAS:

MSXSAVE

Format : MSXSAVE filename
Valid : command
Purpose : To save a Dbasic program in MSX-basic ASCII format, while converting Dbasic keywords to MSX-basic keywords and converting screen coordinates.
Remarks : - For a complete list of convertions see appendix A.
- see also MSXLOAD
Example : *MSXSAVE "OTHELLO"
Note that only the six first characters of the file name are used.
The MSX file name hence is "OTHELL".

MSXTLOAD

Format : MSXTLOAD filename , beginaddress
Valid : command/statement
Purpose : To load a MSX ASCII file at memory address beginaddress.
When beginaddress is omitted, the file will be loaded in the free ram space.
Remarks : see also MSXTSAVE
Example : *MSXTLOAD "TXTFIL", 5000

MSXTSAVE

Format : MSXTSAVE filename
Valid : command/statement
Purpose : - to copy a 'TEXT' file from the DAI mass storage device to tape or to a MSX computer according to the sequence :
1. The DAIpc will prompt you with :
'FOR LOADING TEXT FILE, TYPE SPACE'
2. Typing space will load the textfile
3. The DAIpc will prompt you with :
'FOR SAVING TEXT FILE, PRESS RECORD/PLAY, TYPE SPACE'
4. Typing space will send the file to tape or the MSX computer
Remarks : - By 'TEXT' files on DAI is ment :
. FWP files
. saved edit buffers
. in general : all files having a file type 1 containing text
- By 'TEXT' files is not ment :
. standard Dbasic text files on DISK (see Dbasic diskversion)
. standard CP/M text files (see CP/M Manual)
- See also MSXTLOAD
Example : *MSXTSAVE "FWPLET"

APPENDIX A : conversion in MSXSAVE

1.Coordinates and colorconversion

- a coordinate pair X,Y used within a graphic mode Dbasic statement will be converted to X,192-Y
- a coordinate pair X,Y used within a text mode Dbasic statement will be converted to X,23-Y
- a color Z used within a Dbasic statement will be converted to Z MOD 16

2.Converted keywords

Dbasic keywords	MSX basic keywords
DOT X,Y Z	PSET (X,192-(Y)),Z MOD 16
DRAW X,Y U,V Z	LINE (X,192-(Y))-(U,192-(V)),Z MOD 16
FILL X,Y U,V Z	LINE (X,192-(Y))-(U,192-(V)),Z MOD 16,BF
CURSOR X,Y	LOCATE X,23-(Y)
MODE arguments	SCREEN arguments
COLORG "	COLOR "
COLORT "	COLOR "
ON BREAK GOTO line	ON STOP GOSUB LINE
BREAK ON	STOP ON
BREAK OFF	STOP OFF
CONTINUE	RETURN
SOUND arguments	PLAY arguments
GETC	ASC(INPUT$(1))
SCRN(X,Y)	POINT (X,192-(Y))
SPC(argument)	SPACE$(argument)
CURX	POS(0)
CURY	23-CSRLIN
XMAX	256
YMAX	192
RND(A)	RND(1)*A
MID$(A$,B,C)	MID$(A$,1+B,C)

APPENDIX B : Error codes

Error nr Error message

1 MSX INSUFFICIENT MEMORY
No more free ram space is available to store the MSX file or MSX file block.

2 MSX DATA DROP OUT
The data stream from tape or from the MSX computer suddenly stopped.
Note : usualy this is not a error condition, because the MSX file length is not known in advance.

3 MSX END OF FILE
An END OF FILE mark (#1A) has been detected im the data stream from tape or from the MSX computer.
Note : this is not the error condition.

****
** VPA: Variable Precision Arithmetic **
****

Was ist VPA ?

VPA ist ein Unterprogrammpaket in Maschinensprache, mit dem arithmetische Operationen mit einer in Zweierschritten waehlbaren Stellenanzahl von 2 bis 504 Stellen und Exponenten im Bereich -32768 bis 32767 moeglich sind.

Was kann VPA ?

VPA besteht aus 17, von BASIC aufrufbaren Routinen:
VOP1,VOP2,VCON,VPRT,VADD,VSUB,VMUL,VDIV,VSQR,
VHLF,VABS,VNEG,VSGN,VINT,VFRC,VMOD,VRND

Wie arbeitet VPA ?

Die VPA-Zahlen werden als Strings mit folgendem Format gespeichert:
L , EL , EH , V , 10^0;10^-1 , 10^-2;10^-3 , ...
 L: Laenge des Strings > 3
EL: Exponent low byte
EH: Exponent high byte + #80
 V: Vorzeichen 0=+ #FF=-
L-3 BCD-Bytes Mantisse, also 2*L-6 Stellen.

Die Zahlenstrings muessen bereits vor ihrer Verwendung in VPA-Routinen die gewuenschte Laenge haben.
Wenn in einer VPA-Routine ein Fehler auftritt, wird STATUS auf #F gesetzt. STATUS wird durch die VPA-Routinen nicht geloescht, muss also von BASIC mit POKE auf 0 gesetzt werden.
Aus folgenden BASIC-Zeilen, die zweckmaessigerweise am Anfang des BASIC-Programms eingebaut werden sollten, sind die Adressbelegungen fuer VPA V1.1 zu ersehen. VPA kann natuerlich auch in andere Speicherbereiche assembliert werden.

```
10    VPA%=#2F0:STATUS%=VPA%
20    VOP1%=VPA%+#10:VOP2%=VOP1%+#8:VCON%=VOP2%+#8:VPRT%=VCON%+#190
30    VADD%=VPRT%+#130:VSUB%=VADD%+#1F0:VMUL%=VSUB%+#30:VDIV%=VMUL%+#F0
40    VSQR%=VDIV%+#210:VHLF%=VSQR%+#90:VABS%=VHLF%+#10:VNEG%=VABS%+#10
50    VSGN%=VNEG%+#10:VINT%=VSGN%+#20:VFRC%=VINT%+#D0:VMOD%=VFRC%+#50
60    VRND%=VMOD%+#40
```

In #300-#E70 liegt das Programm selbst, bis #2700 der Arbeitsbereich.
Der Grund fuer den grossen Arbeitsspeicherbedarf wird bei der Beschreibung von VMUL weiter unten erlaeutert.

Die einzelnen Subroutinen:

1) VOP1
   Zweck: Auswahl des ersten Operanden
   Aufruf: CALLM VPAOP1,X$
   Funktion: Der Pointer auf X$ wird fuer die folgende VPA-Routine zur Verfuegung gestellt
2) VOP2
   Zweck: Auswahl des zweiten Operanden
   Aufruf: CALLM VOP2,X$
   Funktion: Der Pointer auf X$ wird fuer die folgende VPA-Routine zur Verfuegung gestellt
   Wenn sich der Pointer auf einen Operandenstring nicht aendert, da keine Stringlaengen geaendert werden etc., braucht der Operandenstring nicht vor jedem VPA-Aufruf neu definiert werden (siehe Beispielprogramm).

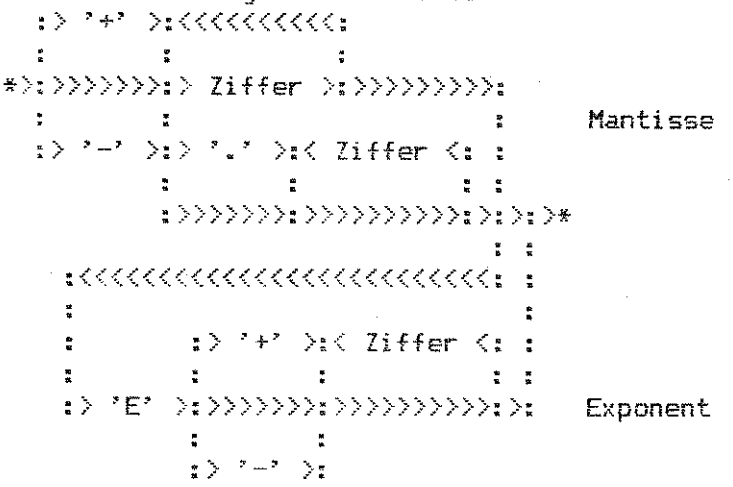
3) VCON

Zweck: Umwandlung ASCII-Darstellung -> VPA-Darstellung

Aufruf: CALLM VOP1,X$:CALLM VCON,Z$

Funktion: - LEN(Z$)<4 -> Abbruch
- sinnloses Zeichen in Mantisse -> der
- sinnloses Zeichen in Exponent -> Routine ;
- Overflow/Underflow -> STATUS := #F
- sonst -> Z$:= VAL(X$)

Wie das Formatdiagramm zeigt, ist VCON sehr grosszuegig, was den Syntax des ASCII-Strings X$ betrifft:



4) VPRT

Zweck: Ausgabe einer VPA-Zahl

Aufruf: CALLM VPRT,Z$

Funktion: PRINT Z$

Das Ausgabeformat richtet sich nach dem Inhalt von STATUS:

STATUS= #0: normiert

STATUS= #1: mit 0en-Unterdrueckung

STATUS=#FF: mit 0en-Unterdrueckung und +-Unterdrueckung

Aufgrund des einfachen Aufbaus der VPA-Zahlen koennen leicht Ausgaberoutinen fuer andere Formate geschrieben werden (auch in BASIC). Der Exponent ergibt sich in BASIC aus ASC(MID$(Z$,0,1))+256*ASC(MID$(Z$,1,1))-#8000, das Vorzeichen aus ASC(MID$(Z$,2,1)) und die Ziffern aus den ASC der restlichen Bytes, jeweils linker ((asc IAND #F0) SHR 4) und rechter (asc IAND #F) BCD-Teil.

4) VADD

Zweck: Addition zweier VPA-Zahlen

Aufruf: CALLM VOP1,X$:CALLM VOP2,Y$:CALLM VADD,Z$

Funktion: - Overflow -> Abbruch der Routine ; STATUS := #F
- Underflow -> Z$:= 0
- sonst -> Z$:= X$ + Y$

5) VSUB

Zweck: Subtraktion zweier VPA-Zahlen

Aufruf: CALLM VOP1,X$:CALLM VOP2,Y$:CALLM VSUB,Z$

Funktion: - Overflow -> Abbruch der Routine ; STATUS := #F
- Underflow -> Z$:= 0
- sonst -> Z$:= X$ - Y$

6) VMUL

Zweck: Multiplikation zweier VPA-Zahlen

Aufruf: CALLM VOP1,X$:CALLM VOP2,Y$:CALLM VMUL,Z$

Funktion: - Overflow -> Abbruch der Routine ; STATUS := #F
- Underflow -> Z$:= 0
- sonst -> Z$:= X$ * Y$

Um bei groesseren Stellenanzahlen Rechenzeit zu sparen, wird nach folgender Methode multipliziert: Es werden zwei Tabellen mit Vielfachen von Y$ aufgebaut (1*,2*,...,9*), eine davon normal, die andere um eine BCD-Ziffer, also ein Halbbyte, verschoben. Die jeweiligen Vielfachen aus diesen Tabellen werden dann stellenverschoben addiert. Der Nachteil dieses Verfahrens ist der hohe Speicherbedarf, naemlich 2*9*256 Byte = 4608 Byte. Aber: wer hat, der kann. Es wird nur bei wenigen Anwendungen hohe Rechengenauigkeit und Graphik zusammen gebraucht und die Rechenzeitersparnis rechtfertigt den grossen Speicherbedarf.
Da das Produkt sowieso auf doppelt so viele Stellen berechnet werden muss, wie die Faktoren besitzen, kann Z$ doppelt so viele Stellen umfassen wie X$ bzw. Y$.

7) VDIV
Zweck: Division zweier VPA-Zahlen
Aufruf: CALLM VOP1,X$:CALLM VOP2,Y$:CALLM VDIV,Z$
Funktion: - Overflow,Y$=0 -> Abbruch der Routine ; STATUS := #F
- Underflow -> Z$:= 0
- sonst -> Z$:= X$ / Y$
Auch hier werden zunaechst Tabellen aus Vielfachen von Y$ aufgebaut. Diese Vielfachen werden dann stellenverschoben mit dem Dividenden verglichen und subtrahiert.

8) VSQR
Zweck: Ziehen der Quadratwurzel aus einer VPA-Zahl
Aufruf: CALLM VOP1,X$:CALLM VSQR,Z$
Funktion: - X$<0 -> Abbruch der Routine ; STATUS := #F
- sonst -> Z$:= SQR(X$)
Die Berechnung erfolgt nach:
$z_{n+1} := (z_n + x/z_n)/2$ until $z_{n+1} = z_n$ wobei z_0 = x mit halbiertem Exponenten
Achtung: X$ muss (als Variable) verschieden von Z$ sein !

9) VHLF
Zweck: Halbieren einer VPA-Zahl
Aufruf: CALLM VHLF,Z$
Funktion: Z$:= Z$*0.5
kein Check auf Underflow (Exponent laeuft durch)
Exponentfehler, wenn 0 geteilt wird
VSQR verwendet aus Geschwindigkeitsgruenden eine eigene Routine zum Halbieren, die durch VHLF auch vom BASIC erreichbar ist.

10) VABS
Zweck: Berechnung des Absolutwertes einer VPA-Zahl
Aufruf: CALLM VABS,Z$
Funktion: Z$:= ABS(Z$)

12) VNEG
Zweck: Negieren einer VPA-Zahl
Aufruf: CALLM VNEG,Z$
Funktion: Z$:= -Z$

13) VSGN
Zweck: Berechnung des Signum einer VPA-Zahl
Aufruf: CALLM VSGN,Z$
Funktion: STATUS := SGN(Z$)
(Z$<0 -> STATUS := #FF
Z$=0 -> STATUS := 0
Z$>0 -> STATUS := 1)

14) VINT
Zweck: Berechnung des Integers einer VPA-Zahl
Aufruf: CALLM VINT,Z$
Funktion: Z$:= INT(Z$)

15) VFRC
    Zweck: Berechnung der Fraction einer VPA-Zahl
    Aufruf: CALLM VFRC,Z$
    Funktion: Z$:= FRAC(Z$)
              (Z$:= Z$ - INT(ABS(Z$)) * SGN(Z$))
16) VMOD
    Zweck: Berechnung des Modulo zweier VPA-Zahlen
    Aufruf: CALLM VOP1,X$:CALLM VOP2,Y$:CALLM VMOD,Z$
    Funktion: Z$:= X$ MOD Y$
              (Z$:= X$ - INT(X$ / Y$) * Y$)
    Achtung: X$ und Y$ muessen (als Variable) verschieden von Z$ sein,
             da Z$ bei der Rechnung als Zwischenspeicher gebraucht wird !
17) VRND
    Zweck: Runden der letzten beiden Stellen einer VPA-Zahl
    Aufruf: CALLM VRND,Z$
    Funktion: letzte zwei Stellen >44 -> Aufrunden der vorvor-
                                         letzten Stelle
              0-Setzen der letzten zwei Stellen

Beispielprogramm:

Als praktisches Beispiel hier noch ein kleines Programm zur Berechnung von e
nach der Formel e = S 1/i!
                   i=0...

```
10    VPA%=#2F0:STATUS%=VPA%
20    VOP1%=VPA%+#10:VOP2%=VOP1%+#8:VCON%=VOP2%+#8:VPRT%=VCON%+#190
30    VADD%=VPRT%+#130:VSUB%=VADD%+#1F0:VMUL%=VSUB%+#30:VDIV%=VMUL%+#F0
40    VSQR%=VDIV%+#210:VHLF%=VSQR%+#90:VABS%=VHLF%+#10:VNEG%=VABS%+#10
50    VSGN%=VNEG%+#10:VINT%=VSGN%+#20:VFRC%=VINT%+#D0:VMOD%=VFRC%+#50
60    VRND%=VMOD%+#40
100   INPUT "Auf wieviele Stellen soll gerechnet werden ";STELLENANZAHL%:PRINT
110   IF STELLENANZAHL%<1 OR STELLENANZAHL%>504 THEN 100
120   LEER$=SPC((STELLENANZAHL%+1)/2+3)
130   ZEINS$="1":EINS$=LEER$
140   CALLM VOP1%,ZEINS$:CALLM VCON%,EINS$
150   E$=EINS$:I$=EINS$:SUMMAND$=EINS$
160   POKE STATUS%,0:CALLM VPRT%,E$:PRINT
170   EALT$=E$
180   CALLM VOP1%,SUMMAND$:CALLM VOP2%,I$:CALLM VDIV%,SUMMAND$
190   CALLM VOP2%,E$:CALLM VADD%,E$
200   CALLM VOP1%,I$:CALLM VOP2%,EINS$:CALLM VADD%,I$
210   IF E$<>EALT$ THEN 160
```

SUMMAND$ wird bei jedem Durchlauf durch I$ dividiert, enthaelt also den Kehrwert der momentanen Fakultaet und wird zu E$ addiert. Bei jedem Durchgang wird die Konstante EINS$ zu I$ addiert. Die Berechnung endet, wenn die Rechengenauigkeit erreicht ist und E$ sich nicht mehr aendert (E$=EALT$).

SOFTWARE voor EE digitiser
(distributed by ENTRAC BV 020/981937 Holland)

Entry Points:

		kleuren
400	MODE 8	4
403	MODE 6	4
406	MODE 8 (4x)	4
409	MODE 4	4
40C	MODE 7	16
40F	MODE 7 (1/2)	16
412	SCRNCOPY 9	
415	SCRNCOPY 1/2 x 1/2	
418	MODE 2	4
41B	MODE 8 midden	4
41E	MODE 6 midden	4
421	MODE 8 CALLM421H,adres	4
424	MODE 8 CALLM 424H,adres	4
427	MODE 5	16
42a	MODE 5 (1/2)	16

De software voor de camera interface bestaat uit 2 delen:
1/ 400 - 9FF
2/ F800 - F84f (stack ram)

MAGNIFIER is an enlarged ZOOM.
Start in MODE 6/6A with CALLM 700H.
Use TAB to switch ON/OFF the output of X,Y,XSCALE,YSCALE,
CURSOR to change X,Y
shift+CURSOR to change XSCALE,YSCALE
RETURN to display the choosen part of the screen
break for end.