DAInamic Software & Library
c/o W. Hermans
Mottaart 20 B-3170 HERSELT

D - BASIC

B version 2.2 + extensions

DAInamic software

INTRODUCTION

Because some bugs were detected in DBASIC V2.1 a new release of the DBASIC language was neccesary.
In this new release, DBASIC V2.2, all known errors are corrected. Also some new statements have been implemented and some signifficant changes concerning the DAI operating system have been made.
Also included on the DBASIC V2.2 package are some standard extensions of the language. These extensions handle I/O-device control, programmable function keys and structured listing & cross-reference output.
An SPL assembly language source of these extensions is added to enable you to match the extensions performance to your own needs. For compatibility reasons however, do not change the syntax of commands and functions !!

how to read this manual

This manual is a supplement to the DBASIC V2.1 manual.
It is devided into 3 chapters.

Chapter 1 : Covers a variety of topics.
Chapter 2 : Briefly describes new statements.
Chapter 3 : A general description about adding extensions.

Standard extensions are described in appendix A to B.

version 2.2 /B
W.Coremans 1/1/85

CHAPTER 1

1.1 ERRATA IN THE DBASIC V2.1 MANUAL

For a clear understanding of the DBASIC V2.1 manual some errors have to be corrected :

page 23

```
line 20   change   : 60 FOR I=0 TO 10:PRINT FAC!(N):NEXT
          in       : 60 FOR I=0 TO 10:PRINT FAC!(I):NEXT
line 23   change   : N*(N-1) if N>0
          in       : N*(N-1)! if N>0
line 28   change   : 30 ELSE FOR I=1 TO N:T!=T!*I:NEXT
          in       : 30 ELSE FOR I=1 TO N:T!=T!*I:NEXT:FN=T!
```

page 28

```
line 21   change   : Example : 10  PROCEDURE DUMMY I
          in       : Example : 10  PROCEDURE DUMMY VAR I
```

1.2 INITIALISATION

DBASIC V2.2 is an auto-start program (see DBASIC V2.1 manual). After DBASIC V2.2 has been loaded the textfile CONFIG.TXT will be loaded and submitted to the DBASIC command interpreter. This submit file facility allows DBASIC commands to be batched for automatic processing. The standard submit file just loads all the extensions from tape.
Since the file type byte of the file CONFIG.TXT is 1, it is easy to change the file contents :

-enter the commands in EDIT mode
 ex. *EDIT
     $EXTEND "DCR"
     DCR 1
     REW
     UT
     Z3
     R
     B
     LOAD "MYPROG"
     RUN
-type [BREAK],[BREAK]
-save the EDIT buffer :
 (ex. DSAVE DEEK(#A2),DEEK(#A4);"CONFIG.TXT")

After loading DBASIC V2.2, this submit file will load the DCR extension, select DCR 1, REWind DCR 1, load an utility file, load the DBASIC program "MYPROG" and run it.

Caution : allways include the commands

```
$EXTEND "DCR"          or        $EXTEND "CAS"
DCR [<dcr number>]               CAS [<cas number>]
```

in CONFIG.TXT, otherwise you could get problems like loosing the hard break facility.

Note : you can directly activate a submit file located in the EDIT buffer by POKE #296,1

1.3 COMPILATION

When loading a compiled DBASIC program an extra compilation pass will be performed before execution of the program. This compilation pass will initialise the absolute run address of extended commands and extended functions (see 1.4).
Due to this absolute run address, often used extended commands and functions will run much faster.

1.4 EXTENDED FUNCTIONS

In addition to extended commands, it is possible in DBASIC V2.2 to define extended functions. An example of such an exended function is the KEY$(<number of function key>) which will be explained in appendix B. How to implement your own extended functions can be read in chapter 3.

1.5 ERROR CODES

DBASIC V2.2 knows 2 new error-numbers and error-messages :

error-number	error-message
56	ELSIF WITHOUT IF An ELSIF statement is encountered for which there is no previous, unmatched IF statement.
57	UNDEFINED FUNCTION The appropriate DBASIC extension to which the function belongs is not in memory.

1.6 READ ONLY CODE

The complete DBASIC machine language program and the extensions $SYSTEM, DCR and CAS are written in read only code. This means that they all can be stored for instance in free EPROM banks on a KENDOS system. About 11k bytes of random acces memory would become available for data storage.
The system ram used by DBASIC is completely located below 2ech. So, using a switched eprom bank system, the same memory map as in BASIC V1.0 (1.1) can be maintained.
If there is enough interest, it will be concidered to devellop such a system.

CHAPTER 2

NEW COMMANDS AND STATEMENTS IN DBASIC V2.2

Note : for format notation see DBASIC V2.1 manual

2.1 IF . THEN ... ELSIF . THEN ... ELSE ... END IF

Format : IF <logical expression 0> THEN [<statements>]
[[ELSIF <logical expression I> THEN [<statements>]]
...
[[ELSE [<statements>]]
ENDIF

Valid : statement

Purpose : The ELSIF . THEN statement is added and can be very efficiently used to make selections. More than 1 ELSIF . THEN statement can be placed between the IF ... END IF.
When the result of <logical expression I> is true, the THEN clause is executed.

Example :
```
...
100 IF ERR=11 THEN PRINT "block length error"
110 ELSIF ERR=12 THEN PRINT "insufficient memory"
120 ELSIF ERR=13 THEN PRINT "checksum error"
130 ELSIF ERR=14 THEN PRINT "data drop-out error"
140 ELSE PRINT "error number";ERR
150 END IF
...
```

2.2 TITLE

Format : TITLE <string expression>

Valid : statement

Purpose : To name the program.
The evaluation of <string expression> will be the name of the program.

Remarks : The program name will be used as a default for some commands. When there is no file name specified in the SAVE command, the program name will be used as file name.
The $LIST and XREF commands of the standard XREF-extension take the program name as a default listing header.

Example :
```
10  TITLE "GRAPHICS ANIMATE"
20  ...
```


CHAPTER 3

HOW TO IMPLEMENT NEW STATEMENTS AND FUNCTIONS

3.1 GENERAL

Extension can be used to add new functions, commands and/or statements to the existing instruction set of DBASIC.
Using a standard DBASIC extension as example, it will be explained how extra functions and statements can be programmed in assembly language.
Some knowledge of 8080 assembly language programming and the DAI operating system is desired for understanding.

3.2 THE EXAMPLE : function keys

Function keys are certainly a very usefull extension of DBASIC : they can save you a lot of typing work and they can be reprogrammed whenever needed.
The statements and functions controlling the function keys are :

KEY	to program the function keys
KEYLIST	to list the function keys
KEYON	to enable function keys
KEYOFF	to disable function keys
KEY$	represents the contents of a function keys

note : For the correct syntax see appendix B.
See also the SPL assembly language source file KEY V2.2 on the DBASIC V2.2 package.

3.3 COMMANDS AND STATEMENT

To link commands or statements to DBASIC you have to provide a table specifying the syntax, runaddresses etc...
Listing 1 (page 9), a piece of SPL macro assembler source code of a program to create the KEY extension, defines the table-layout.

The elements of the table are described below :

-extension name :

Is used for error-reporting and the $DELETE command.

-extension id :

Not used anymore in DBASIC V2.2 due to an extra compilation pass. Any value between 0h and 0ffh is valid.

-relocation table :

Is used in $EXTEND and points to a table with all the addresses to be relocated. In order to be completely relocatable a machine language program should only contain 2-byte word memory-references (ex. avoid the use of LOW and HIGH operators in MACRO 80). After loading and relocation of the extension the relocation table will not be kept on line.

note : In the SPL assembly language sources on the DBASIC V2.2 package, a macro is included to generate this relocation table automaticaly.

-separators :

Is a set of 8 punctuation marks needed during encoding and listing of the commands. Any argument is always preceeded by one of these separators.

-command string :

Identifies the command. Only the 1st character of the command string can be non-alphanumeric (ex. $ in $EXTEND).
Note that the length byte preceeding the command string has to be of the binairy form 0000 1111, with 1111 the length of the command string.

-encode control :

It's binairy form is ccXX 1111 (X stands for don't care).

with	1111	number of possible arguments+2
	cc=X1	statement valid in program
	cc=1X	command valid in direct command mode

-execution address :

Points to the startaddress of the command's execution code.

-argument syntax :

it's binairy form is tttt sssf

with	sss	the number of the separator which preceeds the argument (from 0 to 7).
	f=1	the argument preceded by separator sss is obligatory.
	f=0	the argument preceded by separator sss is optional.
	tttt=0000	the argument is a floating point expression.
	tttt=0001	the argument is an integer expression.
	tttt=0010	the argument is a string expression.
	tttt=0011	the argument is a variable reference.
	tttt=0111	the argument is an array reference.
	tttt=1011	the argument is a group of variable references separated by ',' (cfr. READ).
	tttt=1111	the argument is a group of array-references separated by ','.
	tttt=1100	the argument is a group of expressions (either integer-, floating point- or string expessions) separated by ','.

In our example the encode control of KEY is 0c4h, thus KEY can be used as direct command or as statement in a program. The length of the info is 4, 2 bytes for the run-address and 2 bytes of argument syntax description :

argument syntax 17h : an integer expression preceeded by separator 3 (a blank) has to be supplied.
argument syntax 21h : an string expression preceeded by separator 0 (a ',') has to be supplied.

3.4 FUNCTIONS

The table layout is different for functions.
The elements are :

-function string :

Identifies the function. The 1st character of the function string can be non-alphanumeric, the last character may indicate its type (ex. $ in KEY$).
Note that the length byte preceeding the function string has to be of the binairy form 0001 1111, with 1111 the length of the function string.

-encode control :

It's binairy form is tttt 1111.

with	1111	number of possible arguments+2
	tttt=0000	it is a floating point type function
	tttt=0001	it is an integer type function
	tttt=0010	it is a string type function

-execution address :

Points to the startaddress of the function's execution code.

-argument syntax :

it's binairy form is tttt 0000

with	tttt=0000	the argument is a floating point expression.
	tttt=0001	the argument is an integer expression.
	tttt=0010	the argument is a string expression.
	tttt=0011	the argument is a variable reference.
	tttt=0111	the argument is an array reference.

In our example the encode control of the function KEY$ is 23h, thus KEY$ is a string type function. The length of the info is 3, 2 bytes for the run-address and 1 byte for the syntax description :

argument syntax 10h : an integer expression has to be supplied.

3.5 RUNTIME

The runtime code usualy can be seen as a sequention of 2 parts :

part 1 : evaluate the arguments.
part 2 : do some processing using the evaluated arguments as parameters.

For evaluation of the arguments you need the addresses of standard DBASIC routines. As you can see in the SPL sources on the DBASIC V2.2 package the number of these DBASIC routines is very small.

Note that in evaluating optional arguments a test is done on a 0-byte in the textbuffer. This is because for a non-supplied optional argument a 0-byte is encoded in the textbuffer (see XREF source).

3.6 EXTENSION CONTROLS

Five pointers in the DBASIC system ram are reserved to control encoding, listing and evaluation of extended commands.
These five controls are :

USCMTB : (0c8h) is a pointer to the first extension-root (=start of 1st table)
A next extension is linked to the previous extension through the next table pointer (=relocation table pointer).

SEPTAB : (0cah) is a pointer to the separator table during encoding.

ROTSAV : (0cch) is used in error-handling. If ROTSAV=0h an error will be considered to be generated in a DBASIC command, else the error will be considered to be a specific extension error and ROTSAV points to the extension root. Thus if we want explicit extension error messages we have to set ROTSAV equal to our extension root (PFKROT in our example).
Then we would get error messages of the form :

KEY ERROR nnn or
KEY 'special error message' (see ERRREP)

ERRREP : (0ceh) is a pointer to a special extension-error-reporting-routine. Note that in our example ERRREP is set to PDBERR i.e. DBASIC error messages will be printed.

USCREC : (33h) is a jump to the extension's auto-recovery routine. If an error occures during execution of an extended command, you may have to restore some system data or anything else that has been changed by the extended command. USCREC allows you to do it.
USCREC can also be used to set or reset ROTSAV and ERRREP (see DCR V2.2)

LISTING 1

```
;
;---extension table---
;
PFKROT  DB      3H
        DB      'KEY'           ;extension name
        DB      0BH
        DB      PFKID           ;extension id.
        DW      PFKTBL          ;relocation table
        DB      ',;: /.=#'      ;separators
        DB      5H
        DB      'KEYON'         ;enable function keys
        DB      0C2H            ;encode control KEYON
        DW      RPFKON          ;run-address KEYON
        DB      6H
        DB      'KEYOFF'        ;disable function keys
        DB      0C2H            ;encode control KEYOFF
        DW      RPFKOF          ;run-address KEYOFF
        DB      3H
        DB      'KEY'           ;define function key
        DB      0C4H            ;encode control KEY
        DW      RPFK            ;run-address KEY
        DB      17H             ;int. ex./sep ' '
        DB      21H             ;$ ex./sep ','
        DB      7H
        DB      'KEYLIST'       ;list function keys
        DB      0C2H            ;encode control KEYLIST
        DW      RPFKLS          ;run-address KEYLIST
        DB      14H             ;!! 1x for functions !!
        DB      'KEY$'          ;function KEY$
        DB      23H             ;$ type
        DW      RPFK$           ;run-address KEY$
        DB      10H             ;int. ex.
        DB      0H
;
```

APPENDIX A

STRUCTURED LISTING AND CROSS-REFERENCE

GENERAL

The XREF extension adds 2 new commands to the DBASIC instruction set. These commands enable you to list your programs in a structured way and/or to become a cross-reference output of all the symbols.
By simply typing $LIST a structured listing of the complete program, followed by a cross-reference is produced.
The XREF command will skip the structured listing and will directly output a cross-reference.

Structured listing means :

DBASIC program lines are spread out on several listing lines to enable correct indentation of :

- .iteration structures (ex. FOR/NEXT,REPEAT/UNTIL,WHILE /WEND).
- .selection structure (ex. IF/ELSIF/ELSE/END IF).
- .definition structures (ex. PROCEDURE/END PROC,FUNCTION /END FN).

A counter also shows the level of indentation.

Special care has been taken for some details :

- .line numbers are printed rigth justified
- .keywords, symbols or text will not be splitted at the end of the line
- .the title will be truncated if it is to long
- .pages are numbered

cross-reference :

For clarity the DBASIC cross-reference groups all symbols into 6 different classes :

- .extended commands & functions..(X option)
- .procedures.....(P option)
- .functions.....(F option)
- .labels.....(L option)
- .arrays.....(A option)
- .variables.....(V option)

Special care has been taken for some details :

- .for functions,arrays and variables the type is indicated.
- .for functions and arrays the () indicates the necessity of arguments.
- .line numbers are rigth justified.
- .the first line number indicates the line where the symbol is defined (ex. procedures,functions,labels, arrays) or used the first time.

1. $LIST

Format :

$LIST[<first line>][<last line>][,<header>][;<options>]

with : <first line> : line from where to start listing
<last line> : line where to end listing
<header> : the header on each new page
<options> : options for the cross-reference

Purpose : to produce a structured listing followed by a cross-reference output.

Remarks : <first line> and <last line> are integer expressions, <header> and <options> are string expressions. The default for <first line> is the first line of the program. The default for <last line> is the last line of the program. The default for <header> is the string expression following the TITLE statement. The default for <options> is "XPFLAV".

Example : *$LIST 100 1000,"TEST PROGRAM";"FP"

This command asks for a structured listing starting with line number 100 up to line number 1000 and produces a cross-reference of functions and procedures. The heading TEST PROGRAM is printed on top of each page.

XREF

Format : see $LIST

Purpose : to produce a cross-reference output.

Example : *XREF 100 1000;"FP"

This command asks for a cross-reference of all functions and procedures used somewhere between line number 100 and line number 1000. The default heading is printed on top of each page.

ERROR MESSAGES

The XREF extension knows 2 specific error messages :

error-number	error message
1	XREF WRONG DBASIC VERSION XREF can not work correctly with this version of DBASIC (ex. DBASIC V2.1)
2	XREF INVALID OPTION An option has been specified, other than X,P,F,L,A or V.

Besides these 2 error messages XREF could also generate DBASIC error messages because a compilation has to be done.
(ex. MISSING WEND...etc...)

APPENDIX B

PROGRAMMABLE FUNCTION KEYS

GENERAL

The KEY extension is designed to add function keys to the keyboard of the DAI pc. Because the DAI pc does not provide separate keys to represent function keys, 2 keys have to be pressed simultaniously : press [CTRL] together with a numeric key (0 to 9). The shift lock can be set or reset by pressing [SHIFT] and [CTRL] at the same time.
If you are using a separate keyboard (ex. connected via RS232) you will have to adapt the detection of a function key in the SPL assembly language source file KEY V2.2.
The function keys can be used where ever you want : in direct command mode, in edit mode, in the INPUT statement, in the GETC function and so on.
Function keys can contain ascii characters from 0h to 7fh, characters from 80h to 0ffh will represent a BREAK.
By default there are 256 bytes reserved to store function keys, however this can be easily modified by changing the KEY V2.2 source file.

STATEMENTS AND FUNCTIONS

Four statements (also valid in direct command mode) and one string type function can be used to program and to control the function keys :

1. KEY

Format : KEY <integer expresion>,<string expression>

Purpose : to define a function key

Example : *FOR I=0 TO 9:KEY I,"":NEXT
this command will erase all the function keys

*KEY 0,CHR$(0)+CHR$(18)+CHR$(#D)
this command will define the end-of-buffer mark which can be used in edit mode to delete all the text following this mark.

2. KEYON

Format : KEYON

Purpose : to enable function keys

3. KEYOFF

Format : KEYOFF

Purpose : to disable function keys

4. KEYLIST

Format : KEYLIST

Purpose : to list the function keys

5. KEY$

Format : KEY$(<int. ex.>)

Purpose : represents the contents of function key <int. ex.>

Example : *KEY 0,"":FOR I=1 TO 10:KEY 0,KEY$(0)+CHR$(8):NEXT
this command line will program function key 0 to delete 10 characters.

```
10  FOR I=0 TO 9
20  IF LEFT$(KEY$(I))<>#D THEN KEY I,KEY$(I)+CHR$(#D)
30  END IF
40  NEXT
```

this few program lines will add a carriage return to the function keys if needed.

ERROR MESSAGES

The KEY extension does not have specific error messages. Instead DBASIC error messages are printed while the ERR$ 'intrinsic' function will be set to 'KEY'.

The most frequent error messages are :

KEY NUMBER OUT OF RANGE
The key number specified in the KEY statement or the KEY$ function was not in the range [0,9].

KEY OUT OF STRING SPACE
No memory is available anymore to store function keys.