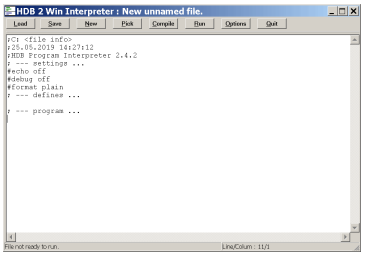


Hdb2Win 2.5

Interpreter

Version 2.1 – May 2021

Content

General remarks

What contains this document?

This document is a description how to use the interpreter of the database management system Hdb2Win. Hdb2Win includes the interpreter to realise multiple functions of the applications (lists in edit forms, complex queries, output data as text or HTML, or output for graphics or geographic data). The interpreter is a very powerful tool because it is able to realise very complex tasks mainly dedicated to data analysis and output. Generally, the interpreter writes the source files for PaleoTax/Graph.

Which software is needed?

To be able to use all here described functions you need an installation of Hdb2Win from version 2.5 on. Applications such as PaleoTax or PalCol are not needed, but the interpreter without data does not give you many options, except you design your own application.

What is new in this version?

The version number is not the version of the interpreter, it is the version of this dokument. The numbers of the dokument versions are generally lower than the global programme version because not each new version of Hdb2Win was accompanied by manuals for every modul.

Version 2 compared to version 1.0

Of course, new functions and commands are introduced. Perhaps really new is the incorporation of selected functions of PaleoTax/Graph that can be directly called from Hdb2Win. Version 2 encompasses a more detailed description of the Integrated Developers Environment (IDE).

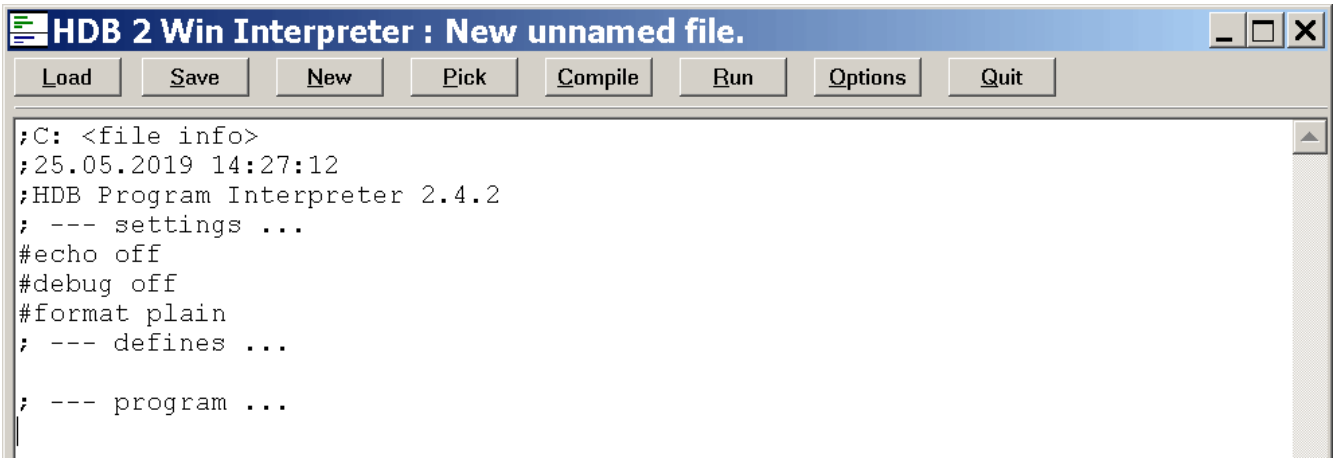
Version 2.1 compared to version 2.0

There are some new commands and we have also enlarged the description of the IDE. Note that all changes compared to version 2.0 in the introduction of this manual are marked in a deep blue colour to find them easier.

The Integrated Developers Environment

Screen

The Interpreter application is a simple environment for programme developing and testing.



Load – Loads an existing programme into the workspace.

Save – Saves the current programme to the hard disc.

New – Cleans the workspace and prepares a new programme.

Pick – Selects one of the last edited programmes.

Compile – Checks and translates the programme.

Run – Executes the programme.

Options – Modify the font and font size in the workspace.

Quit – Terminates the interpreter.

Sections

A new programme has the following structure:

```
;C: <file info>
;25.05.2019 14:32:48
;HDB Program Interpreter 2.4.2
; --- settings ...
#echo off
#debug off
#format plain
; --- defines ...

; --- programme ...
```

The first three lines, following the ;C: (content), are reserved for a short programme description. The first line is in German, the second in English and the third in Spanish (languages so far supported by the database programme). As for instance:

```
;C: Das ist ein Test.
;C: This is a test.
;C: Eso es una prueba.
```

Depending on the language, in which runs the programme, the exact line will be selected when the programme is selected by applications. If there is only one line, this line will be shown. So you do not need to put the German

explication. You may put only one line and describe the programme in your own language. The next lines are written by the programme, indicating the date, time and programme version. All lines starting with a semicolon are comments.

In the setting section (; --- settings) , the standard settings are indicated. Compare below for the meaning. In the define section (; --- defines) all variables used in the programme should be given. It is bad programme practice to define the variable when needed. After this section the programme section with the source code follows.

It is useful not to modify the #error setting. The #debug setting should be set to on when you want to pass step by step through the programme (see below). The #format setting defines the output format of your data. This setting is anyhow important. This type of settings can be modified within the source code when ever needed.

Variables

Variably must be declared. The use of an undeclared variable causes an error and terminates the programme. There three types of variables:

C String. From one to 255 characters.
N Real number.
I Integer number.

All of them may constitute arrays. Names must start with a letter and may contain numbers and some extra signs (such as _ $ #). Reserved names (names of predefined symbols such as date and time or names of functions) are not allowed and rejected. There is no difference between uppercase and lowercase letters.

Example:

```
;define    name,type
define     i,i
define     s,c
define     r,n
define     name,c
;define    date,c     ; illegal because date is an already existing symbol
;define    100,i      ; illegal because must start with a letter
define     today,c
define     power_user,c
```

Declaration of arrays:

```
define     ai,i,default,10      ; the array has a dimension of 10
```

Declaration with an initial value:

```
define     ip,i,default,0,100   ; initial value of ip is 100
```

Values are assigned to variables with the mov command:

```
mov        i,100           ; assigns 100 to i
;mov       100,i           ; illegal operation
stor       ai,1,100        ; first element of the field ai is 100
;mov       ai,i,100        ; illegal operation, use stor!
mov        name,'User'
mov        power_user,name
mov        name,user       ; user must be an existing variable
mov        r,i*100/10
mov        today,date
;mov       i,'User'        ; illegal operation, data
                           ; types do not coincide
```

Once variables are declared they can be used. All variables have an initial values; numerical variables are zero and strings ar empty. Field variables starts always with one as index and the items are addressed with brackets:

```
stor       ai,1,100
mov        ip,ip+ai[1]
mov        ai[ip],ais[ip]+1
;out       ai[0]      ; illegal because zero is an invalid index
```

Tables

Before using a table, it must be opened, by the commands open or file. Within a database, a table can be selected via file. A file command should be followed with a reset command to assure that the request starts with the first record.

```
; show deleted records:
open       species
file       genera
reset                 ; do not forget the reset command
:begin
cmp        deleted,0
je         skip
con        'Deleted : '+gname
:skip
skip                  ; do not forget to skip
jneof      begin
cla
exit
```

Good programme style

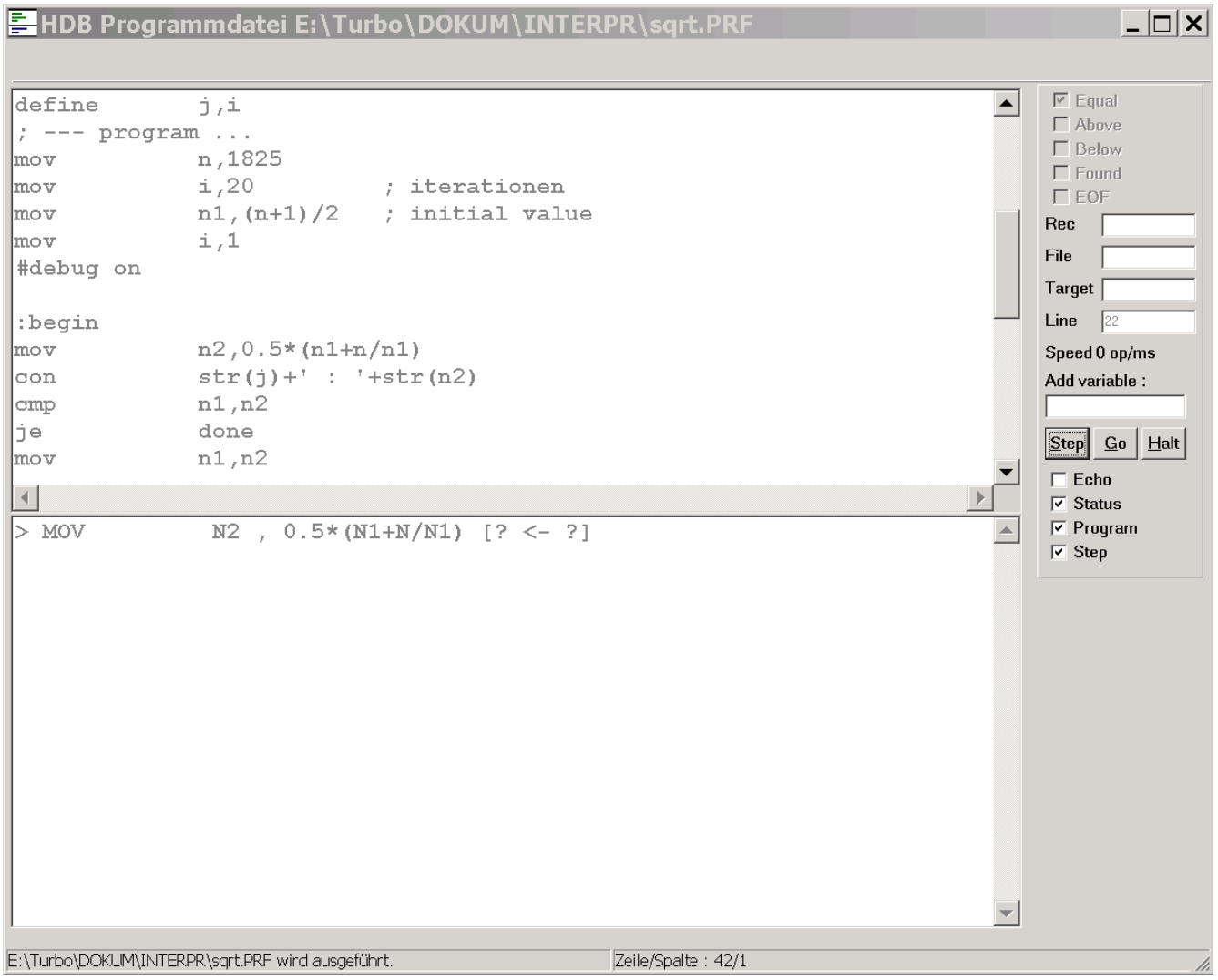
Commands, variables and labels can be in upper or lower case letters, it does not make any difference. You may arrange your source code more clearly using upper case letters for commands, and lower case letters for variables and labels. It is a good programme style to leave space between various blocks of the programme, insert head lines and comments on the programme lines. Otherwise a later modification is more difficult.

```
; - defines
define     sv,c  ; data
define     ic,i  ; counter
; - open table
cda        ; current directory
open       occurrs
; - reset data
file       ageiugs
reset
cmp        reccount,0
je         onerror    ; empty table makes no sense
:beginreset
put        value1,0
put        value2,0
flsh
skip
jneof      beginreset
; - write list file
file       regions
```

```
reset
cmp        reccount,0
je         onerror    ; empty table makes no sense
stream     corr.lst,o
:beginlist
outl       rname
skip
jneof      beginlist
; - check main file
file       occurr
reset
cmp        reccount,0
je         onerror    ; empty table makes no sense
; - open stream
stream     cdata.pgr,o
outl       ';C: Correlation
outl       ';D: 4,CORR.CFG,CORR.LST,0'
; - data output
:begindata
cmp        olocality.region*ocitate.now_ass,0   ; avoid invalid values
je         skipdata
cmp        sv,str(olocality.region,5)+str(ocitate.now_ass,5)
je         skip                                 ; avoid double output
mov        sv,str(olocality.region,5)+str(ocitate.now_ass,5)
outl       sv
mov        ic,ic+1
:skipdata
skip
jneof      begindata
; - close stream
strm
cla
con        str(ic)+' values.'
; - show
cmp        ic,0 ; any value?
Je         onerror
graph      cdata.pgr
; - exit
:onerror
cla
exit
```

Debugging

Debugging is the process to find errors in your program. You can start debugging right from the beginning, setting #debug on in the setting section, or within the programme text. In the example below the #debug setting is within the source code. When the programme reaches this line, the screen is divided in two three parts. The upper part shows the source code of the programme, the lower part the current line and programme output and the right panel the functions of the debugger.



On top the panel shows five flags for Equal, Above, Below, Found and End of file. These flags are set depending on the result of a comparison (CMP), search (FIND) or database command (SKIP). Below are four boxes that show the current record in the current table, the current table, the current target file (STRM), and the source code line. Next below follows a status field that gives the current speed of the interpreter. In the field "Add aviable:" you may add a variable (defined in the interpreter) which current value is shown every step.

Below of this field there are three buttons. **Step** executes the current line and stops (the **Space** key does the same). **Go** starts the programme until it ends or a new #debug on command is found. **Halt** stops the programme. The four settings at the lower end of the panel means the following: When Echo is checked, all outputs all data are also shown on the screen. Status means that all above status data are updated. Program means that all programme lines are shown and Step finally activated the stepwise execution of the programme.

Operators and functions

Basics

One of the principal components of the database engine is the parser module. This module analyses and resolves any type of equation, numerical or text. The module also resolves variables, symbols, functions, and field names of the tables. The interpreter, that is usually used to analyse data, makes intense use of this module. Therefore, data types, operators, and functions are described briefly. The names of the functions are not case sensitive.

Data types

Characters (C)		Numbers (N)	
C (char)	'A'	R (real)	3.142
C (string)	'Alpha'	I (Integer)	3

Operators

Numerical operators

(DT=data type, RT=return type)

Name/ Symbol	Explanation	DT	RT	Example
-	sign	N	N	-(-1) = 1
^ , sqr	square	N	N	2^8 = 256 2 sqr 8 = 256 9 sqr 0.5 = 3
% , mod	rest of a integer division	I	I	21 % 8 = 5
DIV	integer division	I	I	21 DIV 8 = 2
/	real division	R	R	21.0 / 8 = 2.625
*	multiplication	N	N	10 * 2 = 20
-	subtraction	N	N	10 - 13 = -3
+	numerical addition	N	N	10 + 13 = 23
shl	bit-by-bit shift to the left	I	I	2 shl 4 = 32
shr	bit-by-bit shift to the right	I	I	256 shr 4 = 16
and , &	logical and	I	I	1 & 0 = 0
or , |	logical or	I	I	1 or 0 = 1

Character operators

Name/ Symbol	Explanation	DT	RT	Example
$	subtext in text, case insensitive	C	I	'A' $ 'halt' = 1
in	subtext in text, case sensitive	C	I	'A' in 'halt' = 0
+	string addition	C	C	'Ha'+'lt' = 'Halt'

Operators without type

Name/ Symbol	Explanation	DT	RT	Example
<	smaller		I	'a' < 'b' = 1
<=	smaller or equal		I	'a' <= 'a' = 1
>	larger		I	'g' > 'G' = 1
>=	larger or equal		I	25 >= 25 = 1
<>	unequal		I	'a' <> 'A' = 1
=	equal		I	'a' = 'A' = 0

Functions

Functions without parameter

Functions without parameters return contants or status variables of the system or tables.

Owner	Name	Type	Example
System	Date	C	01/05/2019
	Time	C	20:00
	T	I	1 (true)
	F	I	0 (false)
	Totalfree	I	3735552
	Allocated	I	3231904
	Heapstat	I	0
	Release	C	2.4.2.55
	Language	C	ENG (can be ENG GER or SPA)
	Symbolstatus	I	0
	Userpath	C	C:\Documents and settings\Putzi\Documents
	Ideoutput	I	(current output format, see #format)
	Apppath	C	C:\Documents and settings\Putzi\Application data\Hdb2Win
	Programname	C	E:\Turbo\SRC\HDB2WIN.EXE
	Docpath	C	C:\Documents and settings\Putzi\Documents
	Random	R	0.000000000233 (a random value between 0 and 1)
	Currdir	C	C:\Documents and settings\Putzi\Documents\Hdb2Win
	Screenwidth	I	1680
	Screenheight	I	888
Table	Reccount	I	(number of records in a table)
	Fieldcount	I	(number of fields in a table)
	Filename	C	(path and name of the table)
Record	Recno	I	(current record)
	Deleted	I	(status of the record)

Numerical

Functions with parameters have at least one parameter. In some functions, a second and/or third parameter is optional (here shown in [brackets]).

Name	Description	Parameter(s)	RT	Example
ABS	returns the absolute value	N	N	ABS(-12.2) = 12.2 ABS(6) = 6
ROUND	rounds a number	1 = N [2 = I] Number of decimals for rounding.	N	ROUND(100) = 100 ROUND(12.9) = 13 ROUND(12.9,1) = 12.9
SIN	returns the sinus of a value between 0 and 90	N	R	SIN(90) = 1

String

Name	Description	Parameter(s)	RT	Example
ANSI	converts an ASCII string into an ANSI string	C	C	
ASCII	converts an ANSI string into an ASCII string	C	C	
AT	position of a string in another string (case sensitive)	1 = C (search string) 2 = C (string)	I	AT('a','halt') = 2
DELETE	deletes a substring in a string	1 = C (string) 2 = I (start position) 3 = I (number)	C	DELETE('yes',2,1) = 'ys'
DELINS	deletes a substring in a string and inserts another substring	1 = C (string) 2 = I (start position) 3 = I (number of characters to be deleted) 4 = C (substring to be inserted)	C	DELINS('Good morning',6,7,'night') = 'Good night'
INSERT	inserts a substring into a string	1 = C (string) 2 = I (position) 3 = C (substring to be inserted)	C	INSERT('negate',6,'iv') = 'negative'
ISNUM	returns 1, if a string is a valid number	C	I	ISNUM('yes') = 0 ISNUM('7') = 1
LEN	returns the length of a string	C	I	LEN('HELLO') = 5
LOCASE	converts a string into small letters	C	C	LOCASE('ABC') = 'abc'
LPOS	returns the last position of a char in a string (case sensitive)	1 = C (char) (search value) 2 = C	I	LPOS('l','hello') = 4
LTRIM	removes spaces to the left of the string	C	C	LTRIM(' name ') = 'name '

POS	returns the first position of a char in a string	1 = C (char) (search value) 2 = C	I	POS('l','hello') = 3
REPLSTR	replaces a substring in a string by another substring	1 = C (string) 2 = C (search string) 3 = C (substring to be inserted)	C	REPLSTR('seven','se','hea') = 'heaven'
RTRIM	removes spaces to the right of the string	C	C	RTRIM(' name ') = ' name'
SIM	Fuzzy text comparison (delivers a percentage value)	C	I	SIM('LOESER','LOSER') = 64
SPACE	returns a number of spaces	I	C	SPACE(1) = ' '
SUBSTR	returns a substring	1 = C (string) 2 = I (start value) 3 = I (number of characters)	C	SUBSTR('hello',2,2) = 'el'
TRIM	removes all spaces in a string	C	C	TRIM(' my name ') = 'myname'
UPCASE	converts a string into capital letters	C	C	UPCASE('aBc') = 'ABC'

Date and time

Name	Description	Parameter(s)	RT	Example
DATSTR	converts an integer into a date (string)	I	C	DATSTR(1)=' 1. 1. 0') DATSTR(731094) = 31.08.2001
DATVAL	converts a date (string) into an integer	C	I	DATVAL('12.3.2001')=730922
DDAY	returns the day (within the months) of a numerical date	I	I	DDAY(DATVAL('12.3.2021')) = 12
DOW	returns the day of the week (0=Sunday, 1=Monday, etc.)	I	I	DOW(DATVAL(date)) = 1
DOY	returns the current day within the year	I	I	DOY(DATVAL('12.3.2001')) = 71
DYEAR	returns the year of a numerical date	I	I	DYEAR(DATVAL('12.3.2001')) = 2001
MONTH	returns the months of a date	I	I	MONTH(DATVAL('12.3.2001')) = 3
TIMESTR	converts a numerical value (seconds since midnight) into a time string	I	C	TIMESTR(70000) = 19:26:40
TIMEVAL	converts a time string into a numerical value (seconds since midnight)	C	I	TIMEVAL('12:28:12') = 44892

Database

Name	Description	Parameter(s)	RT	Example
FIELD	returns the name of the field (according to field list)	I	C	FIELD(10) = FNAME
FLDINF	returns information about the data field (according to field list)	I	C	FLDINF(10) = FNAME,C,25
FLDLEN	returns the length of a data field	C		FLDLEN('fname') = 25
FLDPOS	returns the position of a data field in the buffer			FLDPOS('fname') = 2
FULLTEXT	full text search in the current record	1 = C [2 = I] Indicates where to search: 1, not in text fields (default); 2, also in text fields; 3, only in text fields. [3 = I] Indicates wether the search is case sensitive: 0, not case sensitive (default), 1, case sensitive.	I	FULLTEXT('name') = 0
GRAW	gets data from the buffer	I (position of the field) I (length of the field)	C	GRAW(2,10) =’Anonymous’

Logical

Name	Description	Parameter(s)	RT	Example
NOT	logical negate	I	I	NOT(4) = 0 NOT(0) = 1

Type conversion

Name	Description	Parameter(s)	RT	Example
CHR	number to char	I (0 .. 255)	C	CHR(65) = 'A'
DECIM	converts a hexadecimal number into an integer	C	I	DECIM('FF') = 255
HEX	converts a integer into a hexadecimal number	I	C	HEX(255) = 'FF'
INT	cuts the decimal part from a real number without rounding	N	R	INT(12.9) = 12 INT(100) = 100
ORD	converts a char into the ASCII value	C (char)	I	ORD('@') = 64
REAL	converts an integer into a real number	I	R	REAL(6) = 6.0

STR	converts a number into a string	1 = N [2 = I (length)] [3 = I (decimals)]	C	STR(12.5) = 12.5 STR(12.5,5) = ' 12.5' STR(12.5,7,2) = ' 12.50'
TRUNC	converts a real number into an integer without rounding	N	I	TRUNC(2.2) = 2 TRUNC(9.9) = 9
VAL	converts a string into a number	C	R	VAL('12.5') = 12.5 VAL('x') = error
XSTR	converts a number into a string, but inserts commas	1 = N [2 = I (length)] [3 = I (decimals)]	C	XSTR(10^7) = 10,000,000

Without type

Name	Description	Parameter(s)	RT	Example
IFF	a value is returned, depending on the result of a condition	1 = I (condition) 2 = CN (if true) 3 = CN (if false)	CN	IFF(3>2,'true','false') = 'true' IFF(name='',0,len(name))=12

System

Name	Description	Parameter(s)	RT	Example
FUNCINFO	gives information about a function	I	C	FUNCINFO(15) = DELETE(C,I,I) = C
SYMBOL	returns a value or 0, depending whether a variable or field exists	C	I	? SYMBOL('author') = 5

Variables and constants

Variable	**Constant**
Its value may change.	Its value does not change.
Are declared during programme execution. Data fields are also variables but are declared by the DBMS when opening a table	Are not declared.
Variables have a type through declaration.	Constants have a type, but are not declared.

Commands

Introduction

A programme of the interpreter consists of the following elements.

Comments

A comment starts with a semicolon. A commend can also stand behind a command or parameter.

Interpreter directives

Directives for the interpreter start with the number sign (#). They define how the programme es executed.

Labels

A label is a marker in the source code where the execution can be continued through a jump command.

Command

A command of the interpreter consists of the key word for the command and none, one, or several parameters. Parameters can be optional.

No command, interpreter directive or label is case sensitive. If you like you may write all commands in uppercase letters and labels in lowercase letters, or vice versa. It is a good style to leave an empty line above a label. Programme sections can be separated by a semicolon followed by a line of dashes. It helps to insert comments that describe what is happening in various sections of a programme.

Conventions in the formal description

<cconst>	Character constant
<cvar>	Character variable
<iconst>	Integer constant
<ivar>	Integer variable
<const>	Constant without type (!)
<var>	Variable without type (!)
&	macro operator
<&cvar>	Character variable
<&ivar>	Integer variable
<cconst | &cvar>	| = or
[ ]	optional parameter
<expression>	complex combination of variables and constants

Where a <var> is allowed not always a <const> is allowed and viceversa.

Set of constants:	<on | off >
	< 1 | 0>
	<A | B | O | Q>

Other keywords:

‘default’	cconst
<label>	label for programme control
<format>	any format information

<array>	field of variables
<dbfield>	database field

Following the commands are listed according to their functional group. Most commands are provided with additional commands or example. For commands that can cause an error that terminates the programme, the possible errors are listed.

Compreter directives

#COMMENT <on | off> | <1 | 0> (Rarely)

Allows to insert a section of comments.

Example

```
#comment on
This programme does nothing.
#comment off
```

#DEBUG <on | off> | <1 | 0> (Occasionally)

Switches Step-by-step execution on or off.

Remarks

Works only in the IDE (Integrated Developers Environment).

#ECHO <on | off> | <1 | 0> (Rarely)

Copy all output data to the screen on or off.

Remarks

Works only in the IDE.

#ERROR <on | off> | <1 | 0 | 2> (Rarely)

Defines which action is taken when an error occurs.

Parameters

ON	0	Displays error and terminates programme. Default value.
1	Displays error and queries whether programme should be terminated.
OFF	2	Does not display the error but stores the error value in the variable LASTERROR.

Remarks

When the directive is set off (value 2) and the execution of a command causes an error, in LASTERROR a value other than zero will be stored. The error can be controlled and must be handled by the user. LASTERROR becomes zero when #ERROR 2 is called again.

When an error occurs and there exist a label :onerror, the programme continues at this label in order to return to a specified table or record. This label should be used mainly in programmes called by the database through edit forms. Be careful what follows below the :onerror label. If any error occurs here, it is not reported and fatal things may happen.

Example

```
#error   2
FILE     test          ; try to use the table test
#error   0
CMP      lasterror,0         ; ask whether an error occurred
JE       ok      ; no error, so go to label ok
CON      'Table test not found!'   ; message
JMP      end           ; do anything ese
:ok
CON      'Table test is ready !'
; do anything with this table
:end
EXIT
```

#FORMAT <iconst | &ivar> (Occasionally)

Defines the output format via STRM.

Parameters

<iconst | &ivar>	Format Definition
	0 | Plain – ASCII (Standard)
	1 | Word – Word
	2 | RTF – RTF
	3 | HTM – HTM
	4 | Simple RTF – simple RTF (when ANSI is exported)
	5 | N – ANSI

Remarks

Take care that options 1, 2, and 4 require proper formatting (see OUT, OUTL). When the CVT command will be used (to convert the text into RTF or Word), corresponding format files (style sheets) must be available. The RTF style sheets can be created in Application library / Options / Tools / Edit style sheets 2.0. Word conversion is not up to date and should not be used anymore. The current output mode is available through the symbol IDEOUTPUT:

0, ASCII (standard)

1, Word (proper templates for conversion must be available)

2, Rich Text Format (proper templates for conversion must be available)

3, HTML

4, simple RTF (do not use)

5, ANSI

6, (internal)

7, HTML without line breaks

8, ANSI to ASCII.

Errors

1 Macro evaluation of the first parameter failed

Example

```
#format   RTF
```

#I <programme file> (Rarely)

Adds another programme file to the current programme.

Parameters

<programme file>	Programme file to be included. If the file does not exist, the compreter will report this and abort compilation of the programme.

Remarks

An included file may be a library of often used functions (that can be applied using the CALL command). Note that these files should be included at the very end of the programme, after the exit command. A library usually starts with the EXIT command to avoid that is executed accidentally. A programme that includes a library, that includes another library is translated properly. When a programme includes various libraries and one of these libraries includes an already included library, no error occurs. The interpreter include any library only one time.

Example

```
CDA
; lot of programme stuff
EXIT
; ---- include files MUST be at the end of the programme
#I library.lib
#I c:\Users\Putzi\AppData\Roaming\Hdb2Win\HTMLLIB.LIB
```

#PROGRAM <on | off> | <1 | 0 > (Rarely)

Displays the current programme line.

Remarks

Works only in the IDE.

#REFR (Rarely)

Refresh display.

Remarks

Works only in the IDE.

#STATUS <on | off> | <1 | 0 > (Rarely)

Opens the status window on the right hand side of the screen.

Remarks

Works only in the IDE.

#VAR (Rarely)

Lists all variables with their values.

Remarks

This is a help in finding programme errors.

Example

```
DEFINE   i,i           ; integer variable
```

```
DEFINE    r,n           ; float (or real) variable
DEFINE    ca,c,default,7; field of ten strings
DEFINE    v,i,default,0,100   ; integer variable with an initial value of 100
DEFINE    hw,c,default,0,'Hello, World'   ; string variable with an initial value
          of 'Hello, World'
MOV       i,100         ; constant
MOV       r,i+10        ; expression
MOV       v,v+v         ; expression
MOV       hw,hw+', folks.'    ; expression
;MOV      ca,'Monday'   ; that would cause an error
#var

I = 100
R = 110
CA =
V = 200
HW = 'Hello, World, folks.'
---
OK.
```

#VERSION <cconst> (Rarely)

Assigns a version to the programme.

Remarks

The version number may help if different versions exist, but all were published under the same name (as it is usual the case in programmes of applications). The version is reported in the case of a programme error.

Example

```
#version 1.2
```

Variables

DEFINE <cconst | &cvar>, < C | R | I > [,<iconst | 'default'>[,<iconst>,[<const>]]] (Always)

Defines a variable.

Parameters

<cconst | &cvar>	Name of the variable
< C | R | I >	Data type. I, integer; R, real; C, string of characters (up to 250 characters). Variables referring to record numbers in tables should always be of the integer type (I).
<iconst | 'default'>	The constant or the reserved word 'default' refers to a programme modul. Do not put anything else than 'default'.
<iconst>	If the variable should be an array, here the number of elements should be indicated.
<const>	Initial value that must coincide with the data type.

Remarks

Variables can be defined in any place of the programme but it is recommendable to gather all definitions in the section Defines in the top section of the programme. The name of the variable must start with a letter. All variables are deleted when terminating the programme. When persistent variables are needed, they must be declared with a different owner ID, and with preference in a PTX start file.

Errors

1 Macro evaluation of the first parameter failed

2 ID constant (third parameter) is invalid (must be an integer value)
3 Field dimension constant (fourth parameter) is invalid (must be an integer value)
4 The variable could not be created (probably because it already exists, or it is an reserved or invalid name)
5 The initial value is invalid and could not be assigned to the variable (possibly a data type error)
6 There is no data type indicated for the variable

Example

```
DEFINE   i,i           ; integer variable
DEFINE   rl,r          ; float (or real) variable
DEFINE   ca,c,default,10     ; field of ten strings
DEFINE   v,i,default,0,100   ; integer variable with an initial value of 100
DEFINE   hw,c,default,0,'Hello, World'   ; string variable with an initial value
         of 'Hello, World'
#var

I =
RL =
CA =
V = 100
HW = 'Hello, World'
---
OK.
```

FNC <var>,<expression> (Rarely)

Assigns a function to a variable.

Parameters

<var>	The name of the function is a variable that must be have declared before.
<expression>	The expression must have the same result type as the function.

Remarks

This command helps to shorten complicated expressions.

Errors

1 Evaluating the expression caused an error

Example

```
DEFINE   cfunc,c
DEFINE   i,i
FNC      cfunc,IFF(i=0,'Sunday',IFF(i=6,'Saturday','week day'))
MOV      i,dow(datval(date))
CON      'Today is a '+cfunc
```

SX <cconst | &cvar> (Rarely)

A request whether a certain variable exists.

Parameters

<cconst | &cvar>	Variable to be checked.

Errors

1 Macro evaluation of the first parameter failed

Example

```
SX        name
JE        ok
DEFINE    name,c
:ok
#var

NAME =
---
OK.
```

LOCAL <cconst | &cvar>, < C | R | I > [,<const>] (Rarely)

Defines a local variable.

Parameters

<cconst | &cvar>	Name of the variable
< C | R | I >	Data type. I, integer; R, real; C, string of characters (up to 250 characters). Variables referring to record numbers in tables should always be of the integer type (I).
<const>	Initial value that must coincide with the data type.

Remarks

Local variables can only be defined within a sub routine that was called by CALL. This variable is only valid in the sub routine and is released when leaving the sub routine. The name must be a name different of any other symbol. It is recommendable to put all local definitions just below the label. The name of the variable must start with a letter. All variables are deleted when terminating the sub routine.

Errors

1 Macro evaluation of the first parameter failed
2 The command is only allowed within a sub routine
3 The initial value is invalid and could not be assigned to the variable (possibly a data type error)

Example

```
define    i,i
mov       i,1
#var
call      test
#var
exit
:test
local     j,i
mov       j,1
mov       i,i+j
con       i
#var
ret

Programme
I = 1
2
Programme
I = 2
Local
J = 1
Programme
I = 2
```

MOV <var>,<const | var | expression> (Always)

Assigns a value to a variable.

Parameters

<var>	Name of a variable
<const | var | expression>	A constant, a variable, an numerical or character expression, or the content of a data field can be assigned to a variable. To assign a value to field variables, the command STOR must be used.

Remarks

Assigns the expression of the right hand side to the variable of the left hand side. MOV is a very important and often used command.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Could not write value into variable (because it is probably a constant)
4 Could not write value into variable (because the data types are not compatible)
5 The second parameter could not be resolved
6 Could not write value into variable or data field

Example

```
DEFINE    i,i           ; integer variable
DEFINE    r,n           ; float (or real) variable
DEFINE    ca,c,default,7; field of ten strings
DEFINE    v,i,default,0,100   ; integer variable with an initial value of 100
DEFINE    hw,c,default,0,'Hello, World'   ; string variable with an initial value
          of 'Hello, World'
MOV       i,100         ; constant
MOV       r,i+10        ; expression
MOV       v,v+v         ; expression
MOV       hw,hw+', folks.'    ; expression
;MOV      ca,'Monday'   ; that would cause an error
#var

I = 100
R = 110
CA =
V = 200
HW = 'Hello, World, folks.'
---
OK.
```

MMOV <var | &var>,<const | var | expression | &var> (Rarely)

Assigns a value to a variable with involved macro operators.

Parameters

<var | &var>	Name of a variable. Macro operators are allowed.
<const | var | expression | &var>	A constant, a variable, an numerical or character expression, or the content of a data field can be assigned to a variable. To assign a value to field variables, the command STOR must be used.

Remarks

Assigns the expression of the right hand side to the variable of the left hand side with macro operators involved. This is very rarely the case.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Could not write value into variable (because it is probably a constant)
4 Could not write value into variable (because the data types are not compatible)
5 Evaluating the expression caused an error
6 Could not write value into variable (not exist? wrong type?)
11 Macro evaluation of the first parameter failed
12 Macro evaluation of the second parameter failed

STOR <array>,<iconst | ivar>,<var | expression> (Occasionally)

Writes a value into an array.

Parameters

<array>	The array variable.
<iconst | ivar>	The index in the array.
<const | var | expression>	The value that should be stored in the array.

Remarks

The lowest index is 1, the highest that indicated in the corresponding DEFINE command. Array values will be obtained using brackets.

Errors

1 Macro evaluation of the first parameter failed
2 Variable does not exist
3 Variable is not an array
4 Memory allocation problem (anything internally that should not happen)
5 Parsing : index or expression are invalid (variable or data field probably unknown)
6 Could not write value into variable (because the data types are not compatible)
7 Evaluating the index caused an error
8 Evaluating the expression caused an error

Example

```
DEFINE    ar,c,default,7
STOR      ar,1,'Monday'
STOR      ar,2,'Tuesday'
CON       ar[1]+' / '+ar[2]+' / '+ar[3]

Monday / Tuesday /
---
OK.
```

RANDOM <nvar> (Very rarely)

Returns a random value between 0 and 1.

Parameters

<nvar>	Variable for the random value.

Remarks

Generates a random values and stores it in the given variable.

Errors

1 Parameter(s) is/are lacking
2 Could not write value into variable (not exist? wrong type?)

Example

```
define   n,nv
random   nv
con      nv
exit
```

Programme control

REQ <cconst | &cvar>,<iconst>[,<ivar>] (Regularly)

Opens a message box.

Parameters

<cconst | &var>	Text of the question.
<iconst>	Mode describes which buttons should appear: 0 – OK 1 – OK + CANCEL 2 – ABORT + RETRY + IGNORE 3 – YES + NO + CANCEL 4 – YES + NO 5 – RETRY + CANCEL
<ivar>	Optional variable for the result (which button has been selected): 1 – OK 2 – CANCEL 3 – ABORT 4 – RETRY 5 – IGNORE 6 – YES 7 – NO 8 – CLOSE 9 – HELP

Remarks

REQ is the classical Windows dialog box. The return value is optional.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression

3 Could not write value into variable or data field

Example

```
DEFINE    iq,i
DEFINE    sf,c
FSEL      sf,*.bak
REQ       &("Do you want to delete the file "+sf+"?"),4,iq  ; 4 = YES | NO
CMP       iq,7                              ; NO
JE        nodel
FDEL      &sf                               ; File is deleted
REQ       &("File "+sf+" was deleted."),0             ; return value is optional
:nodel
```

XREQ <cconst | &cvar>,<iconst>[,<ivar>[,<ivar>]] (Rarely)

Opens a message box with more options.

Parameters

<cconst | &var>	Text of the question.
<iconst>	Mode describes which buttons should appear: bit 1 (1) – OK bit 2 (2) – YES bit 3 (4) – NO bit 4 (8) – CANCEL bit 5 (16) – (small question box)
<ivar>	Optional variable for the result (which button has been selected): 1 – OK 2 – YES 4 – NO 8 – CANCEL
<ivar>	Optional variable for the result (whether the small box was marked): 1 – NO 2 – YES

Remarks

XREQ allows in comparison to REQ to store the result globally and apply it automatically if the user wants it. Both return values are optional.

Errors

1 Macro evaluation of the first parameter failed
2 Macro evaluation of the second parameter failed
3 Invalid numerical expression
4 Could not write value into variable (not exist? wrong type?)
5 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    iq,i
DEFINE    io,i
XREQ      "Do you want to create the index file?",22,iq,io  ; YES + NO + small box
CMP       iq,2                              ; NO
JE        noindex
; INDEX ...
```

```
:noindex
CON      "You will be futurely "+IFF(io=1,"not","")+" be asked."
```

CALL <label> (Occasionally)

Call of a sub routine.

Parameters

<label> Label where the sub routine starts.

Remarks

The command continues at the indicated label and returns with the command after the CALL command when comming to the return command (RET). They should be located at the very end of the programme, still behind the EXIT command. They can be also gathered in a programme library that is indicated behind the EXIT command:

EXIT

#i library.lib

This library is another programme file that starts with the command EXIT to avoid that it is started accidently. Large projects that gather hundreds of lines ususally work with libraries. Parameters must be transmitted through variables. Sub routines can have local variables.

Example

```
CALL     outputimage
...
EXIT
; --- subroutines following here
:outputimage
OUT      image
...
RET
```

RET (Occasionally)

Returns after proceeding a su broutine to the calling command.

Remarks

See under CALL for explanation.

Errors

1 This command can only be called at the end of a sub routine

EXIT (Always)

Terminates the programme.

Remarks

The command is not mandatory but it is good style to use it. Since sub routines are mostly at the end of the source code, and the EXIT command is missing, the programme runs into the subroutine. See example.

Example

```
DEFINE   i,i
DEFINE   k,i
OPEN     base
RESET
```

```
:begin
MOV       i,field1
CALL      check
CMP       k,0
JE        skip
MOV       i,field2
CALL      check
CMP       k,0
JE        skip
CON       'Record '+str(recno)+' is valid.'
:skip
SKIP
JNEOF     begin
CLA
EXIT      ; wenn missed, the programme continues with the check subroutine
; ---
:check
MOV       k,0
CMP       i,0
JE        checkret
CMP       i,100
JA        checkret
MOV       k,1
:checkret
RET
```

TERM (Very rarely)

Terminates the execution of the database system after termination the interpreter.

Remarks

This command is only used for installation purposes or programme updates.

Conditions and programme flow

CMP <expression>,<expression> (Always)

Compares two expressions.

Parameters

<expression>	Left expression.
<expression>	Right expression.

Remarks

Depending on the result, to flags are set: the EQUAL flag if the two expressions are equal and the ABOVE flag if the left expression is larger than the right one. CMP should be followed by one of these commands: JE, JNE, JA, JAE, JB, JBE.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Parameter(s) is/are lacking
4 Parameter(s) is/are lacking
5 Evaluating the expression caused an error

Example

```
DEFINE    i,i
MOV       i,10
CMP       i,100
JE        equal    ; no
JA        above    ; no
JB        below    ; yes
```

SFLT <expression> (Rarely)

Sets a global filter.

Remarks

The filter is valid for the commands CND, CPY, ADDF, ADDR, CMPR, BRW. Do not confound this filter with the field list. For some operations, the table must be opened using FILE, not OPEN.

Errors

1 Task is nil or table name is empty (= no file)
2 Memory allocation problem (anything internally that should not happen)
3 Could not resolve the condition (complex error, see history)

Example

```
FILE      authors
SFLT      recno<50
BRW
```

CND (Rarely)

Checks whether or not a condition is fulfilled.

Remarks

The condition can be set by an external programme (for instance the database machine) or by the command SFLT. If the condition is fullfilled, the EQUAL flag is set. A following JE (jump if equal) proceeds with the command next to the indicated label.

Errors

1 Could not resolve the condition (complex error, see history)

Example

```
FILE      authors
RESET
SFLT      substr(fname,6,1)='a'
:begin
CND
JNE       skip     ; jump if not equal
CON       fname
:skip
SKIP
JNEOF     begin
FILE

Ali-Zade
Arzamastsev
Beauvais
Beauvais
Vrblyanski
```

```
Bhargava
...
---
OK.
```

CFLT (Rarely)

Clears the global filter.

JMP <label> (Regularly)

Continues the programme at the label without condition.

Remarks

Forces the programme to continue at the given label.

JE <label> (Always)

Continues the programme at the label if the compared values are equal.

Remarks

JE is preceeded by a CMP command. The programme continues at the label when the EQUAL flag is set after a comparison.

Example

```
FILE      genera
RESET
:begin
CMP       gname,''
JE        genusempty
CON       genus
:genusempty
SKIP
JNEOF     begin
```

JNE <label> (Always)

Continues the programme at the label when the preceeding comparison results in not equal.

Remarks

Must be preceeded by a CMP command. The programme continues at the label when the EQUAL flag is not set after a comparison.

Example

```
CMP       dow(datval(date)),0
JNE       workday
CON       'Sunday'
JMP       exit
:workday
CON       'Monday to Saturday'
:exit
```

JA <label> (Regularly)

Continues the programme at the label if the left expression of a comparison is larger.

Remarks

This command follows directly after a CMP command. The programme continues at the label when the ABOVE flag is set after a comparison.

Example

```
CMP       6,5
JA        larger  ; yes, the programme continues at larger
            ; because 6 ist larger than 5
; ...
:larger
CMP       1,1
JA        notlarger     ; does not jump to notlarger
            ; because the values are equal
```

JAE <label> (Regularly)

Continues the programme at the label if the left expression is larger or equal to the left.

Remarks

JAE is preceeded by a CMP command. The programme continues at the label when the both ABOVE and EQUAL flag is set after a comparison.

Example

```
CMP       value,100
JAE       v1 ; jumps if value>=100
```

JB <label> (Regularly)

Continues the programme at the label if the left expression of a comparison is smaller.

Remarks

This command follows directly after a CMP command. The programme continues at the label when neither the EQUAL nor the ABOVE flag is set after a comparison.

Example

```
CMP       6,5
JB        smaller ; the programme does not continues at smaller
            ; because 6 ist larger than 5
; ...
:smaller
CMP       1,1
JB        notlarger     ; does not jump to notlarger
            ; because the values are equal
```

JBE <label> (Regularly)

Continues the programme at the label if the left expression is below or equal to the right expression.

Remarks

JBE is preceeded by a CMP command. The programme continues at the label when the ABOVE flag is not set after a comparison. The EQUAL flag has no meaning.

Example

```
CMP       value,100
```

```
JBE       v1 ; jumps if value<=100
```

JNEOF <label> (Always)

Continues the programme at the label if the end of the table has not been reached.

Remarks

This command (Jump if Not End Of File) is used if a table is systematically revised until the end of the table is reached.

Example

```
DEFINE    rv,n
FILE      payments
RESET
:begin
MOV       rv,rv+amount
SKIP
JNEOF     begin   ; if the end of the table is NOT reached, the programme
            ; continues with the :begin label, otherwise it goes on
CON       'Total : '+str(rv)
FILE
```

File system

CDA (Occasionally)

Sets the current directory to the place where the interpreter programme is stored.

Remarks

Let's assume the default working directory is c:\Users\<username>\AppData\Roaming\Hdb2Win\ but your data are stored in e:\data\fossils\, also most programmes will be stored in e:\data\fossils\. Calling CDA sets the current directory from ...\Hdb2Win\ to e:\data\fossils\.

Errors

1 Invalid path

CD <cconst | &cvar> (Occasionally)

Selects a file path.

Parameters

<&cconst> The path to be selected.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid path

Example

```
CD        e:\paleo\database
CD        &path   ; path is a variable
```

MD <cconst | &cvar> (Rarely)

Creates a new directory.

Parameters

<cconst | &var>	Name of the directory.

Errors

1 Macro evaluation of the first parameter failed
2 Error creating the directory

Example

```
MD      data
```

DSEL <cvar>[,<0 | 1>] (Occasionally)

Selection of a directory.

Parameters

<cvar>	Variable for the selected path.
<0 | 1>	Optional value that defines whether the change of the drive is allow (1) or not (0).

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the first parameter failed
3 Variable does not exist
4 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE  dn,c
DSEL    dn
CON     dn
```

FFND <cconst | &cvar>,<cvar>[,<ivar>] (Occasionally)

Searches a file.

Parameters

<cconst | &cvar>	Search mask.
<cvar>	Variable for the filename.
<ivar>	Optional file attribute. This must be a variable. Set this value to search for a specific type of file. $01 - Read only files (decimal 1) $02 - Hidden files (decimal 2) $04 - System files (decimal 4) $08 - Disk drives (decimal 8) $10 - Directories (decimal 16) $20 - Archive files (decimal 32) $3F - All files (decimal 63)

Remarks

This command is used to initialize file searching. To get the next file use NFND.

Errors

1 Macro evaluation of the first parameter failed
2 The file name could not be assigned to the variable (does not exist, or a data type error)
3 The attribute could not be assigned to the variable (does not exist, or a data type error)

Example

```
DEFINE    s,c
DEFINE    i,i
FFND      *.prf,s,i
CON       s
CON       i

$test.PRF
128
---
OK.
```

NFND <cvar>[,<ivar>] (Occasionally)

Continues with file searching.

Parameters

<cvar>	Variable for the filename.
<ivar>	Optional file attribute.

Remarks

The command FFND should be used before using NFND.

Errors

1 The command FFND must used before NFND
2 The file name could not be assigned to the variable (does not exist, or a data type error)
3 The attribute could not be assigned to the variable (does not exist, or a data type error)

Example

```
DEFINE    s,c
DEFINE    i,i
FFND      b*.prf,s,i
:begin
CMP       s,''
JE        notmorefiles
CON       s+' ('+str(i)+')'
NFND      s,i
JMP       begin
:notmorefiles
CON       "That's all."

BINOM1.PRF (128)
BT2PAS.PRF (128)
That's all.
---
OK.
```

FSEL <cvar | &cvar>[,<cconst>[,<cconst | &cvar>]] (Rarely)

Selects a file.

Parameters

<cvar | &cvar>	Character variable or macro with variable for the result.
<cconst>	Optional character constant for the file search mask.
<cconst | &var>	Optional constant or macro with initial path.

Remarks

Selects a file. If the selection is canceled the programme will stop.

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the third parameter failed
3 Macro evaluation of the first parameter failed
4 Macro evaluation delivered an empty string
5 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    fname,c
FSEL      fname,*.txt
FSEL      fname,*.db2,e:\paleo\database
```

FILEX <cconst | &cvar> (Occasionally)

Checks whether a file exists.

Parameters

<cconst | &var>	File name.

Remarks

The result of the command sets the EQUAL flag. After calling FILEX the command JE or JNE should follow.

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the first parameter failed

Example

```
FILEX     authors.dbf
JE        found
CON       'Not found : authors.dbf'
JMP       end
:found
CON       'File found : authors.dbf'
:end
EXIT
```

FSIZE <cconst | &cvar> , <ivar> (Rarely)

Gets the file size in byte of a file.

Parameters

cconst | [&]cvar	Filename (Char) as constant or variable with macro operator.
ivar	Integer variable for the return value.

Remarks

Evaluation of the length of a file in byte. Returns -1 if the file was not found (programme will not interrupt, but you have to handle the reported file size). Long names are supported.

Errors

1 Parameter(s) is/are lacking
2 Second parameter is lacking
3 Macro evaluation of the first parameter failed
4 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    ifs,i
FSIZE     $test.prf,ifs
CON       ifs
FSIZE     doesnotexist.txt,ifs
CON       ifs

249
-1
---
OK.
```

CPF <cconst | &cvar>,<cconst | &cvar>[,<cconst | &cvar>] (Rarely)

Copies a file.

Parameters

<cconst | &cvar>	Source file.
<cconst | &cvar>	Target path.
<cconst | &cvar>	Optional target file name.

Remarks

The command copies any given file to another path. If no target file name is indicated, the target file has the same name as the source file.

Errors

1 Macro evaluation of the first parameter failed
2 Macro evaluation of the second parameter failed
3 Macro evaluation of the third parameter failed
4 Complex file copy error (see history)

Example

```
CPF       *.txt,\copy\
CPF       name.db2,,newname.db2
```

FDEL <cconst | &cvar> (Rarely)

Deletes one or more files.

Parameters

<cconst | &cvar>	File mask.

Remarks

Take care with this command, there is no request.

Errors

1 Macro evaluation of the first parameter failed
2 Parameter(s) is/are lacking

Example

```
FDEL      *.BAK
```

Screen input/output

CON <const | var | expression> (Always)

Screen output.

Remarks

This commands has only effect within the IDE or if an output screen is defined. Each CON output is written in a separate line. CON without parameter clears the output area.

Errors

1 Expression invalid

Example

```
CON       date
CON       time
CON       1+1
CON       100*10
CON       'Hello '+'World'

04.03.2018
13:09:39
2
1000
Hello World
---
OK.
```

CONX <const | var | expression> (Occasionally)

Screen output.

Remarks

The difference to CON is, that the result is written in the same line.

Errors

1 Expression invalid

Example

```
FILE      authors
RESET
:begin
CONX      recno
SKIP
```

```
JNEOF     begin
```

KBD <cconst | &cvar>,<var>[,<C | N>] (Regularly)

Reads a value from the screen.

Parameters

<cconst | &cvar>	Description of the value
<var>	Variable
<C | N>	Optional datatype (C for character, N for numerical)

Errors

1 Macro evaluation of the first parameter failed
2 Variable does not exist
3 The value could not be assigned to the variable (possibly a data type error)

Example

```
DEFINE    name,c
DEFINE    year,i
KBD       'Enter name',name,C
KBD       'Enter year',year,N
```

SELCOL <cvar>[,<D | H>] (Rarely)

Selects a color from a color palette.

Parameters

<cvar>	Variable where the colour will be stored.
[<D | H>]	Optional selection of the format, (H) for hexadecimal, (D) for decimal.

Remarks

Take into account that the variable is a character not an integer. When the selection was not successful (= canceled), nothing will be assigned to the variable.

Errors

1 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    sc,c
SELCOL    sc,H
CON       'Selected color '+sc
```

SELDATE <ivar>[,<iconst>[,<iconst>]] (Rarely)

Selects a date from a table.

Parameters

<ivar>	Variable of the input and return value.
<iconst>	Optional start year (default 1900).
<iconst>	Optional number of years (default 150).

Remarks

Selects a date from a table. The input and return value is an integer that can be via DATSTR converted into a string, as DATVAL converts a string date into an integer.

Errors

1 First parameter is lacking
2 Evaluating the first parameter caused an error
3 Evaluating the first parameter caused an error
4 Second parameter is invalid (not a valid year)
5 Third parameter is invalid (must be larger than zero and below 2020)
6 The starting value is beyond the defined limits of the year
7 Could not write value into variable (not exist? wrong type?)

Example

```
define    idate,i
mov       idate,datval('01.01.2020')
req       'Select birth date!',0
seldate   idate,1910,110
cmp       idate,0
je        exit
req       &('Birthdate : '+datstr(idate)),0
:exit
exit
```

OPT.INI <iconst | &ivar>[,<cconst | &cvar>[,<iconst>[,<iconst>]]] (Regularly)

Creates an option table.

Parameters

<iconst | &ivar>	Number of items.
<cconst | &cvar>	Optional heading.
<iconst>	Does not allow the selection of more than one option when set to 1. Optional.
<iconst>	Forces to convert the header from ASCII to ANSI when set to 1. Optional.

Remarks

To use a table of (program) options, this command must be the first. The caption is optional, but has to be used when a third and/or fourth parameter is applied.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Macro evaluation of the second parameter failed

Example

```
DEFINE    res,i
OPT.INI   3,"Select value ... ",1
OPT.LBL   1,'100'
OPT.LBL   2,"1,000"
OPT.LBL   3,"10,000"
OPT.SET   1,1
OPT.EXE
OPT.ONE   res
CMP       res,0    ; Canceled
```

```
JE        exit
CON       'Your selection : '
CON       STR(IFF(res=1,100,IFF(res=2,1000,10000)))
:exit
EXIT
```

OPT.LBL <iconst | &ivar>,<cconst | &cvar> (Regularly)

Assigns a label to a specified option in the list.

Parameters

<iconst | &ivar>	Number of the option.
<cconst | &cvar>	Label.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Macro evaluation of the second parameter failed
4 Option object not created (use OPT.INI first)

OPT.SET <iconst | &ivar>,<iconst | &ivar> (Occasionally)

Sets or clears an option.

Parameters

<iconst>	Number of the option in the list.
<iconst>	Zero clears the box, 1 marks it.

Errors

1 Invalid numerical expression
2 Macro evaluation of the first parameter failed
3 Macro evaluation of the second parameter failed
4 Option object not created (use OPT.INI first)

OPT.ENB <iconst | &ivar>,<iconst | &ivar> (Rarely)

Enables or disables an option.

Parameters

<iconst | &ivar>	Number of the option in the list.
<iconst | &ivar>	Zero disables the option, any value different from zero enables it.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Macro evaluation of the second parameter failed
4 Invalid numerical expression
5 Option object not created (use OPT.INI first)

Example

```
DEFINE    iopt1,i
DEFINE    iopt2,i
DEFINE    iopt3,i
```

```
OPT.INI   4,"Mother's birthday..."
OPT.LBL   1,"Call mom in the morning"
OPT.LBL   2,"Buy flowers"
OPT.LBL   3,"Get the pie"
OPT.LBL   4,"Meet later the boys at the pub"
OPT.ENB   4,0     ; Option 4 is disabled and cannot be selected
OPT.EXE
OPT.RES   1,iopt1
OPT.RES   2,iopt2
OPT.RES   3,iopt3
```

OPT.RD <ccont | &cvar> (Occasionally)

Reads options from a file.

Parameters

<ccont | &cvar> Textfile.

Remarks

Normally the option file was created by the command OPT.WR.

Errors

1 Option object not created (use OPT.INI first)
2 Macro evaluation of the first parameter failed
3 Textfile not found or could not be opened
4 The text file has a format error
5 Option object not created (use OPT.INI first)

Example

```
OPT.INI   3
OPT.LBL   1,'ABC'
OPT.LBL   2,'DEF'
OPT.LBL   3,'GHI'
FILEX     optfile.opt
JNE       exe
OPT.RD    optfile.opt
:exe
OPT.EXE
OPT.WR    optfile.opt
```

OPT.EXE [<ivar>] (Regularly)

Requests the options.

Parameters

[<ivar>] Bitwise coded initial values (that overruns values set by OPT.SET). If the option is not enabled, the value is not set. The same variable returns the result. <ivar> must be a variable, it may not be a constant value.

Remarks

This command must be preceeded by OPT.INI and OPT.LBL. When the operation is canceled, the programme will be aborted. From version 2.4.2 on: an optional parameter (that must be a variable) may contain bitwise coded initial values (that overruns values set by OPT.SET). In the same variable, the results are bitwise returned.

Errors

1 Evaluating the variable caused an error
2 Invalid numerical expression
3 Could not write value into variable (not exist? wrong type?)
4 Option object not created (use OPT.INI first)

Example

```
define    iopt,i
reg.rint  user.myapp.programs.analysis1.option1,iopt
opt.ini   5,"Options 1"
; ...
opt.exe   iopt
reg.write user.myapp.programs.analysis1.option1,iopt
```

OPT.RES <iconst | &ivar>,<ivar> (Regularly)

Returns the result of a option in the list.

Parameters

<iconst | &ivar>	Number of option.
<ivar>	Variable to store the result.

Errors

1 Invalid numerical expression
2 Could not write value into variable (not exist? wrong type?)
3 Macro evaluation of the first parameter failed

Example

```
DEFINE    iopt1,i
DEFINE    iopt2,i
DEFINE    iopt3,i

OPT.INI   4,"Mother's birthday..."
OPT.LBL   1,"Call mom"
OPT.LBL   2,"Buy flowers"
OPT.LBL   3,"Get the pie"
OPT.LBL   4,"Hang out in the bar"
OPT.ENB   4,0
OPT.EXE
OPT.RES   1,iopt1
OPT.RES   2,iopt2
OPT.RES   3,iopt3
```

OPT.ONE <ivar> (Occasionally)

Store the first (or only) selected option.

Parameters

<ivar>	Variable to store the value.

Errors

1 Could not write value into variable (not exist? wrong type?)

OPT.WR <ccont | &cvar> (Occasionally)

Writes the options to a file.

Parameters

<ccont | &cvar> Textfile.

Errors

1 Option object not created (use OPT.INI first)
2 Macro evaluation of the first parameter failed
3 Could not create the file (? invalid name or path)

OL.INI <iconst | &ivar>[,<cconst | &cvar>[,<cconst | &cvar]] (Rarely)

Creates an option table.

Parameters

<iconst | &ivar> Number of items.
<cconst | &cvar> Optional heading.
<cconst | &cvar> Optional explaining text.

Remarks

To use a list of (programme) options, this command must be the first. The captions are optional. The difference to OPT is, that the order of the items can be modified. The command is very complex, moreover if you want to restore the order of former selections from the registry. From version 2.5 on.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Macro evaluation of the second parameter failed
4 Macro evaluation of the third parameter failed

Example

```
define    k,i
define    i,i
define    j,i
req       'Select the desired brands and put most tasty on the top of the list.',0
ol.ini    5,'Request','Beer purchase list'
ol.lbl    1,'Tekate'
ol.lbl    2,'Indio'
ol.lbl    3,'DOS XX'
ol.lbl    4,'Bohemia'
ol.lbl    5,'N.Buena'
ol.exe
mov       k,1
:beginout
ol.res    &k,i,j
con       iff(i=0,'Do not buy brand ','Buy brand ')+str(k)+' with priority '+str(j)
mov       k,k+1
cmp       k,6
jb        beginout
exit
```

OL.LBL <iconst | &ivar>,<cconst | &cvar> (Rarely)

Assigns a label to a specified option in the list.

Parameters

<iconst | &ivar>	Number of the option.
<cconst | &cvar>	Label.

Errors

1 Option object not created (use OL.INI first)
2 Macro evaluation of the first parameter failed
3 Invalid numerical expression
4 Macro evaluation of the second parameter failed
5 Macro evaluation of the third parameter failed
6 Invalid numerical expression
7 Number beyond limits.

OL.SET <iconst | &ivar>,<iconst | &ivar> (Rarely)

Sets or clears an option.

Parameters

<iconst | &ivar>	Number of the option in the list.
<iconst | &ivar>	Zero clears the option, any value different from zero sets it.

Errors

1 Option object not created (use OL.INI first)
2 Macro evaluation of the first parameter failed
3 Invalid numerical expression
4 Macro evaluation of the second parameter failed
5 Invalid numerical expression

OL.ENB <iconst | &ivar>,<iconst | &ivar> (Rarely)

Enables or disables an option.

Parameters

<iconst | &ivar>	Number of the option in the list.
<iconst | &ivar>	Zero disables the option, any value different from zero enables it.

Remarks

Disabled entries can not be activated or deactivated and they cannot be moved within the list.

Errors

1 Option object not created (use OL.INI first)
2 Macro evaluation of the first parameter failed
3 Invalid numerical expression
4 Macro evaluation of the second parameter failed
5 Invalid numerical expression

OL.EXE [<ivar>] (Rarely)

Requests the options.

Parameters

[<ivar>]	Sets and conserves the values.

Errors

1 Option object not created (use OL.INI first)
2 Evaluating the variable caused an error
3 Invalid numerical expression
4 Nothing to select because all items ar disabled.
5 Could not write value into variable (not exist? wrong type?)

OL.RES <iconst | &ivar>,<ivar>[,<ivar>] (Rarely)

Returns the result of a option in the list.

Parameters

<iconst | &ivar>	Number of option.
<ivar>	Variable to store the result.
[<ivar>]	Variable to store the position within the list.

Errors

1 Option object not created (use OL.INI first)
2 Macro evaluation of the first parameter failed
3 Invalid numerical expression
4 Cannot find entry with this number
5 Could not write value into variable (not exist? wrong type?)
6 Could not write value into variable (not exist? wrong type?)

LB.TOC <ivar> (Rarely)

Assigns the token of a listbox to a variable.

Parameters

<ivar>	Token of the listbox.

Remarks

This command is only used for process comunication, e.g. when a programme needs the address of a listbox to write items.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 Evaluating the expression caused an error
4 Invalid numerical expression

LB.CLR (Rarely)

Clears the listbox.

Remarks

The command LB.TOC must be applied before using this command.

Errors

1 The object (label, listbox, image) is unknown

LB.ADD <ccont | &cvar> (Rarely)

Adds an item to a listbox.

Parameters

<ccont | &cvar> Item to be added.

Remarks

The command LB.TOC must be applied before using this command.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the expression caused an error
3 The object (label, listbox, image) is unknown
4 Evaluating the expression caused an error

LB.SEL <ivar> (Rarely)

Sets the current item in the list box.

Parameters

<ivar> Item to be set.

Remarks

The command LB.TOC must be applied before using this command. If the listbox is empty or the item above the capacity of the listbox, the command is ignored.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the expression caused an error
3 The object (label, listbox, image) is unknown
4 Evaluating the expression caused an error
5 Invalid numerical expression

LB.CAP <ivar> (Rarely)

Stores the capacity of a listbox in a variable.

Parameters

<ivar> Variable.

Remarks

The command LB.TOC must be applied before using this command.

Errors

1 The object (label, listbox, image) is unknown
2 Could not write value into variable (not exist? wrong type?)

LB.IDX <ivar> (Rarely)

Stores the currently selected item of a listbox in a variable.

Parameters

<ivar> Variable.

Remarks

The command LB.TOC must be applied before using this command.

Errors

1 The object (label, listbox, image) is unknown
2 Could not write value into variable (not exist? wrong type?)

LS.INI <ccont | &cvar>[,<0 | 1>] (Rarely)

Initiated a list from which an item can be selected.

Parameters

<ccont | &cvar> Caption of the list.
<0 | 1> When 1, the caption is converted into ANSI.

Errors

1 Macro evaluation of the first parameter failed

Example

```
DEFINE   isel,i
DEFINE   ssel,c
LS.INI   'Select'
LS.ADD   'Monday'
LS.ADD   'Tuesday'
LS.ADD   'Wednesday'
LS.SIZE  400,300
LS.EXE   isel,ssel
CON      isel
CON      ssel
```

LS.ADD <ccont | &cvar> (Rarely)

Adds an item to the list.

Parameters

<ccont | &cvar> The item to be added.

Errors

1 Macro evaluation of the first parameter failed

LS.SIZE <ivar>,<ivar> (Rarely)

Defines the size of the form.

Parameters

<ivar>	Width of the form.
<ivar>	Height of the form.

Errors

1 Invalid numerical expression

LS.EXE <ivar>[,<cvar>] (Rarely)

Returns the result of the selection.

Parameters

<ivar>	Number of the item in the list that was selected.
<cvar>	Optionally the selected item.

Errors

1 Could not write value into variable (not exist? wrong type?)
2 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE   isel,i
DEFINE   ssel,c
LS.INI   'Select'
LS.ADD   'Monday'
LS.ADD   'Tuesday'
LS.ADD   'Wednesday'
LS.SIZE  400,300
LS.EXE   isel,ssel
CON      isel
CON      ssel
```

LAB.TOC <ivar> (Rarely)

Assigns the token of a label to a variable.

Remarks

This command is only used for process comunication, e.g. when a programme needs the address of a label to write a caption.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 Evaluating the expression caused an error
4 Invalid numerical expression

LAB.CAP <ccont | cvar> (Rarely)

Assigns a caption to a label.

Parameters

<ccont | cvar> Text that should be assigned to the label.

Remarks

Before using LAB.CAP, LAB.TOC should be used. The most frequent application is the function of showing a small blue text box at the top of various tables in the application library of the database.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 The object (label, listbox, image) is unknown
4 Evaluating the expression caused an error

Example

```
LAB.TOC   lbparam
LAB.CAP   genent_s
```

IMG.TOC <ivar> (Rarely)

Assigns the token of a image area to a variable.

Parameters

<ivar> Token of the image. The image must be declared in a FRM file with the ID number 198. The token will be assigned by the database to the variable imparam.

Remarks

This command is only used for process comunication, e.g. when a programme needs the address to display an image.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 Evaluating the expression caused an error
4 Invalid numerical expression

Example

```
IMG.TOC     imparam
IMG.SHOW  &pcrecord.pgraph
```

IMG.SHOW <cconst | &cvar> (Rarely)

Shows an image.

Parameters

<cconst | &var> Filename.

Remarks

The command IMG.TOC must have been executed before.

Errors

1 Macro evaluation of the first parameter failed

2 The object (label, listbox, image) is unknown

ALB.CR <iconst|&ivar>[,<cconst|&cvar>[,<iconst>]] (Rarely)

Opens an album of images.

Parameters

<iconst|&ivar>	The size of the album in percent from the screen size. The value must be between 11 and 100.
[,<cconst|&cvar>]	An optional caption of the album.
[,iconst]	When this option is set to 1, it is possible to tag items. When you wish to use this command, the second parameter cannot be skipped.

Remarks

An album is an rectangular area where up to 48 images can be displayed. These images comes normally from a table. This command creates the album.

Errors

1 Macro evaluation of the first parameter failed
2 Percent value must be larger than 10 and lower than 100
3 Macro evaluation of the second parameter failed

Example

```
ALB.CR   80,'Album',1
```

ALB.ADD <cconst|&cvar>[,<cconst|&cvar>] (Rarely)

Adds an item (an image) to the album.

Parameters

<cconst|&cvar>	The name of the file.
[,<cconst|&cvar>]	An optional caption.

Remarks

The item must be an image file.

Errors

1 Album is not opened (use ALB.CR beforehand)
2 Macro evaluation of the first parameter failed
3 Macro evaluation of the second parameter failed

Example

```
; This programme shows all images of
; the type specimens of a selected
; genus of a PaleoTax database.
; --- settings ...
#echo off
#debug off
#format plain
; --- defines ...
define   igenus,i
define   itypesp,i
define   scaption,c
```

```
; --- programme ...
cda
open      types,4
open      dbpictur,4
file      genera
rsel      'Select genus',igenus,0
cmp       igenus,0
je        exit
go        igenus
alb.cr    80,&gname
file types
reset

:begintypes
cmp       t_spec.c_genus,igenus
jne       skiptypes
con       t_spec.key
mov       itypesp,t_specmn
mov       scaption,t_specmn.spmncoll.acronym+#32+t_specmn.spmnno
file      dbpictur
reset

:beginimages
cmp       dbrecord,itypesp
jne       skipimages
cmp       at('.SPECMENS',owner),0
je        skipimages
alb.add   &(pcrecord.pgraph,2,200),&scaption

:skipimages
skip
jneof     beginimages
file      types
:skiptypes
skip
jneof     begintypes
alb.show

:exit
exit
```

ALB.SHOW [<ivar>] (Rarely)

Displays the album.

Parameters

[<ivar>]	Returns optionally the value of a tagged item.

Remarks

Shows the album. If no items were added, no album is shown. An optional variable stores the selected image.

Errors

1 Album is not opened (use ALB.CR beforehand)
2 Could not write value into variable (not exist? wrong type?)

Example

```
define    i,i
define    ai,default,48
alb.cr    80,'Select image',1
```

```
file     images
reset
:begin
cmp      i,48
je       show
cnd      ; any condition
jne      skip
alb.add  imagename
mov      i,i+1
stor     ai,i,recno
:skip
skip
jneof    begin
:show
mov      i,0
alb.show i
cmp      i,0
je       exit
go       ai[i]
req      &('Selected item = '+name),0
:exit
exit
```

File input/output

STRM [<con> | <cconst | &cvar>[,<cconst>]] (Always)

Opens a file for output.

Parameters

<con>	Console, the output windows below the programme area.
<cconst | &cvar>	The target file name.
<cconst>	An optional letter stands for (A)ppend, (B)ackup, (O)verwrite, and (Q)uest, if the file already exists. (A) appends the next to the existing file, (B) changes the name of the existing file (to *.BAK), (O) overwrites the existing file, and (Q) a

Remarks

Opens an output channel, which is usually the console or a text file. STRM without parameter just closes the current output stream. When the parameter of the STRM command is default, it refers to the file name set by the application library. Only programmes started from the principal search form within the application library may use this parameter.

Errors

1 Macro evaluation of the first parameter failed
2 Could not create the output file (invalid name, no writing rights, invalid path)

OUT <const | var | expression>[,<format>] (Always)

Output of an expression into a file.

Parameters

<const | var | expression>	Expression to be sent to the output channel.
<format>	Character [and paragraph] format. Valid formats are '|nn' for the character and '|n' for the paragraph. There can be up to 99 character formats ('01' to '99'), and up to 16 paragraph formats ('1' to 'F').

The formats must correspond to definition of the style sheet used for conversion (see CVT). A paragraph format cannot stand alone (a character format can). So the convention is the following:

```
OUT     <expression>,|CC     ; character format
or
OUT     <expression>,|CC|P   ; character and paragraph format
but never
OUT     <expression>,|P      ; paragraph format alone
```

Remarks

The parameter is resolved and sent into the output channel defined by STRM. The format is only valid for files which are to be later converted into Word or RTF. The difference to OUTL is, that no end of line code is inserted.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Evaluating the expression caused an error
4 Output of expression/constant failed (complex error, see history)
5 Output of format failed (complex error, see history)

Example

```
OUT       'Hello World' ; constant
OUT       date          ; predefined variable
OUT       100+10+value  ; expression
OUT       author.fname  ; dbfield
OUT       c_genus.gname,|02   ; dbfield with a character format
OUT       c_genus.author.fname,|03|1; dbfield with character and paragraph format
```

OUTL <const | var | expression>[,<format>] (Always)

Output of an expression into a file.

Parameters

See above (OUT).

Remarks

The parameter is resolved and sent into the output channel defined by STRM. The format is only valid for files which are to be later converted into Word or RTF. The difference to OUT is, that a end of line code is inserted.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Evaluating the expression caused an error
4 Output of expression/constant failed (complex error, see history)
5 Output of format failed (complex error, see history)

OUTTEXT <cconst>[,<format>] (Occasionally)

Output of the data of a text field.

Parameters

<cconst>	A database field of the M type, also called memo field. Can be also a field of interconnected tables.
<format>	A format specification (see OUT).

Errors

1 Complex text (memo) field output error
2 Output of format failed (complex error, see history)

Example

```
CMP       c_genus.gennote,0
JE        notextoutput
OUT       'Note. ',|02
OUTTEXT   c_genus.gennote,|01|1
:notextoutput
```

OUTP <const>[,<format>] (Occasionally)

Output of a constant value into a file.

Parameters

<const>	Constant to be sent to the output channel.
<format>	Character [and paragraph] format

Remarks

The difference between OUT and OUTP is that OUTP accept only constant values, no expressions, table data etc.

Errors

1 Output of expression/constant failed (complex error, see history)
2 Output of format failed (complex error, see history)

Example

```
FILE      authors
RESET
OUTP      'fname'  ; set inverted comma to conserve small and large caps
OUTP      ' - '    ; set inverted comma to conserve spaces
OUTP      fname
OUT       ' - '+fname

fname - FNAME - Anonymous
---
OK.
```

OUTPL <const>[,<format>] (Occasionally)

Output of a constant value into a file.

Parameters

<const>	Constant to be sent to the output channel.
<format>	Character [and paragraph] format

Remarks

The difference to OUTP is that a new line is inserted.

Example

```
FILE      authors
RESET
OUTPL     'fname' ; set inverted comma to conserve small and large caps
OUTPL     fname
OUTL      fname

fname
FNAME
Anonymous
---
OK.
```

TXT.CR <cconst | &cvar> (Occasionally)

Creates a textfile.

Parameters

<cconst | &var>	File to be created.

Remarks

Creates a text file.

Errors

1 Macro evaluation of the first parameter failed
2 Empty file name
3 Could not create the file (? invalid name or path)

TXT.OP <cconst | &cvar>,<0 | 1> (Occasionally)

Opens an existing text file.

Parameters

<cconst | &var>	File to be opened.
<0 | 1>	Allowance whether the pool path is allowed; 0 = not allowed, 1 = allowed.

Errors

1 Macro evaluation of the first parameter failed
2 Empty file name
3 Could not open the file (? does not exist, invalid name or path)

TXT.RS (Occasionally)

Resets the text file to the beginning.

Errors

1 Textfile is not opened/created.
2 Text file reset error

TXT.AP (Rarely)

Prepares an open textfile for appending new items.

Remarks

Command TXT.OP must preceed this command to open an existing file, or TXT.CR to create a new text file.

Errors

1 Textfile is not opened/created.

Example

```
TXT.CR    names.txt
TXT.AP
TXT.WL    'This is a new file.'
```

TXT.RD <cvar> (Occasionally)

Reads one line into the variable.

Remarks

If the end of the file is reached, the variable contains the string <EOF>. The EOF flag is not set! Only 253 characters are stored; if the line has more characters than that, they get lost. In this case use the command TXT.RDL.

Errors

1 Textfile is not opened/created.
2 Text file read error
3 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    s,c
TXT.OP    names.txt
:begin
TXT.RL    s
CON       s
CMP       s,'<EOF>'
JNE       begin
TXT.CL
```

TXT.RDL <cvar>,<cvar>,<cvar>,<cvar> (Rarely)

Reads a long line in up to four variables.

Parameters

<cvar>,<cvar>,<cvar>,<cvar> Variables to store the various parts of the string.

Remarks

A string of up to 1000 characters is read and split into four variables. This command is mainly used for importing data.

Errors

1 Textfile is not opened/created.
2 Could not write value into variable (not exist? wrong type?)
3 Text file read error
4 Parameter(s) is/are lacking
5 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    s1,c
DEFINE    s2,c
DEFINE    s3,c
DEFINE    s4,c
TXT.OP    text.txt
MOV       s1,''
MOV       s2,''
MOV       s3,''
MOV       s4,''
TXT.RDL   s1,s2,s3,s4
CON       'The string has a length of '+STR(LEN(S1)+LEN(S2)+LEN(S3)+LEN(S4))+'
          characters.'
```

TXT.WR <expression> (Occasionally)

Writes the expression to a text file without a line skip.

Parameters

<expression> Expression to be written.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 Textfile is not opened/created.
4 Evaluating the expression caused an error
5 Text file write error

TXT.WL <expression> (Occasionally)

Writes the expression to a text file with a line skip.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Evaluating the first parameter caused an error
3 Textfile is not opened/created.
4 Evaluating the expression caused an error
5 Text file write error

TXT.CL (Occasionally)

Closes the text file.

Database operation : create, open and close tables

FCREA <cconst | &cvar> (Rarely)

Creates a new table.

Parameters

<cconst | &cvar> Name of the new table.

Remarks

Creates a table.

Errors

1 Macro evaluation of the first parameter failed
2 Error during file creating (see error histoy)

Example

```
FCREA     datafile
```

OPEN <cconst | &cvar>[,<iconst>] (Always)

Opens a table with all interconnected tables.

Parameters

<cconst | &var>	Table to be opened.
<iconst>	Open mode bit1 (1) = checks access index. bit2 (2) = creates access index if not available. bit3 (4) = opens cache and reads the data in the main memory. bit4 (8) = reads alias names from the form files. bit5 (16) = extended open mode, not used. bit6 (32) = opens base in the read-only mode. bit7 (64) = data pooling is not allowed. bit8 (128) = unused. To set an open mode, the values in parentheses have to be totaled. A good choise are the first three options, so the open mode (1+2+4) would be 7.

Remarks

The command can be repeated if various tables, that are not interconnected with each other should be used.

Errors

1 Macro evaluation of the first parameter failed
2 Open mode (second parameter) is invalid (must be an integer value < 256)
3 Memory allocation problem (anything internally that should not happen)
4 Open table complex error (see history)
5 Memory allocation problem (anything internally that should not happen)

Example

```
OPEN      citation,7
```

FILE <cconst | &cvar>[,<iconst>] (Always)

Opens a table.

Parameters

<cconst | &var>	Name of the table.
<iconst>	A number between 1 and 128 as a kind of identification.

Remarks

FILE without parameter closes the current file. FILE opens only the indicated table, not interconnected tables. If a databases has been opened beforehand, FILE selects one file of the database. FILE without name has no effect; it does not close a specified table from the base because the base can only be opened o closed as a whole.

Errors

1 Macro evaluation of the first parameter failed
2 Memory allocation problem (anything internally that should not happen)
3 Error closing file
4 Memory allocation problem (anything internally that should not happen)
5 Open table complex error (see history)
6 Memory allocation problem (anything internally that should not happen)
7 Invalid file selector (second parameter)

RESET (Always)

Resets the table.

Remarks

This command sets the current record at the beginning of the table (physical beginning or according to an index). This command should always be applied after using FILE and before any operation that goes through the whole table.

Errors

1 Task is nil or table name is empty (= no file)
2 It was not possible to go to the beginning of the table (empty?)
3 Error reading the record

Example

```
FILE      genera
RESET
:begin
CMP       gauthor,0
JE        noauthor
```

SLN (Rarely)

Forces the file command to use a new task and not the present one.

Remarks

This command can only be used immediately before the FILE command, unless the table is opened with interconnected tables.

Example

```
FILE      test,1   ; open table as task 1
FILE      new,2    ; open table as task 2
            ; but close table test
; therefore :
FILE      test,1
SLN
FILE      new,2
SLF       1  ; use table test
SLF       2  ; use table new
```

SLF <iconst | &ivar> (Very rarely)

Selects a table by a personal task number.

Parameters

<iconst | &ivar> File number (see command FILE)

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Task is nil or table name is empty (= no file)
4 Memory allocation problem (anything internally that should not happen)

Example

```
FILE     test,1  ; open table as task 1
FILE     new,2   ; open table as task 2
           ; but close table test
; therefore :
FILE     test,1
SLN
FILE     new,2
SLF      1 ; use table test
SLF      2 ; use table new
```

TSK <iconst | &ivar> (Very rarely)

Selects table by global task number.

Parameters

<iconst | &ivar> Task number.

Remarks

This is a very rare command that is only used for internal or test purposes.

Errors

1 Macro evaluation of the first parameter failed
2 Invalid numerical expression
3 Task is nil or table name is empty (= no file)
4 Memory allocation problem (anything internally that should not happen)

RLD (Very rarely)

Save and reload a database.

Remarks

This command is used when complex modifications in the database were realized. The data are saved to the harddisk and the base is loaded again into the main memory.

Errors

1 Complex reload error (see history)

POB <cconst | &cvar> (Occasionally)

Request whether a specified table forms part of the database and is opened.

Parameters

<cconst | &cvar> Name of the table.

Remarks

The purpose of this command is mainly to guarantee in programmes that are called by edit forms, the necessary tables are opened. If a table is opened by the user in the file mode from the command line application, many programmes that fill listboxes etc. cannot be executed properly because not all necessary files are opened.

Errors

1 Macro evaluation of the first parameter failed

Example

```
POB       citation
JNE       exit
FILE      citation
RESET
; ...
:exit
EXIT
```

POPL (Rarely)

Request whether the current table is located in the pool path.

Remarks

The command should be follow by a JE or JNE. The command is mostly applied for internal checks.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed

REIDX (Very rarely)

Invalidates the access file for this table.

Remarks

This is more an internal command that improves data consistency. It should be only applied when numerous changes in key fields were untertaken.

CLB (Occasionally)

Close database.

Remarks

CLB is used when a table was beforehand opened with OPEN.

Errors

1 Complex close base error (see history)

CLA (Regularly)

Closes all tables.

Remarks

The command CLB closes only a database, e.g. tables that were opened using OPEN, but not other files opened using the FILE command. CLA closes (and saves) all tables.

Errors

1 Complex close base error (see history)
2 Complex close all error

CAO (Very rarely)

Clears all variables created by the DMS.

Remarks

To be used only after CLB, otherwise a crash of the programme may occur. The use of this command is rather internal. Be careful in its application.

Database operation : read / show / modify / write records

GO <iconst | ivar> (Regularly)

Moves the file pointer to the specific record and read it.

Parameters

<iconst | ivar> The value must be an integer.

Remarks

In the case a record number that points to another file is taken from a table, it should be tested whether the value is valid, e.g. above zero and within the range of the table size. See the second example below.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : the parameters is invalid
3 Evaluating the first parameter caused an error
4 Invalid numerical expression
5 Error reading the record

Example

```
FILE      authors
RESET
GO        500
CON       fname
EXIT

Náprstek
---
OK.

; --- second example, testing the value of the record number

DEFINE    i,i
OPEN      base
FILE      data1
RESET
```

```
:BEGIN
MOV       i,rcdata
CMP       i,0      ; !!!
JE        skip
FILE      data2
CMP       i,reccount
JA        invalid  ; !!!
GO        i
; ...
:invalid
FILE      data1
:skip
SKIP
JNEOF     begin
CLA
EXIT
```

SKIP (Always)

Moves the file pointer one record forward.

Remarks

If the file end is reached the EOF flag is set. After SKIP always the question should follow whether the end of the file has been reached (JNEOF). The JNEOF command resets the EOF flag.

Errors

1 Cannot obtain next record (mostly a problem of the index file)
2 Error reading the record

Example

```
FILE      authors
RESET
:begin
CON       fname+', '+cname
SKIP
JNEOF     begin
```

APR (Rarely)

Appends the current record as a new record to the table.

Errors

1 Task is nil or table name is empty (= no file)
2 Record writing error
3 Error reading the record

CLR (Rarely)

Clears the current record.

Errors

1 Task is nil or table name is empty (= no file)

APB (Occasionally)

Appends an empty record to the table.

Remarks

Identical with the commands APR and CLR.

Errors

1 Task is nil or table name is empty (= no file)
2 Record writing error
3 Error reading the record

Example

```
FILE      authors
APB
PUT       fname,'Einstein'
PUT       cname,'A.'
FLSH      ; without FLSH, the new record will not be saved
```

EDT (Occasionally)

Edits one record.

Remarks

The command FLSH must be used after editing.

Errors

1 Task is nil or table name is empty (= no file)
2 Complex edit record error

Example

```
; The error check should be preferably switched off and the result controlled:
#error 2
EDT
#error 0
CMP       lasterror,0
JNE       error
FLSH
JMP       ok
:error
CON       'Edit was canceled...'
:ok
```

EDTM (Rarely)

Edits multiple records.

Errors

1 Task is nil or table name is empty (= no file)
2 Complex edit record error

EDM <cconst> (Rarely)

Edits the text of a text (Memo) field of the current record.

Parameters

<cconst> Name of the data field.

Errors

1 Task is nil or table name is empty (= no file)
2 Field is unknown
3 Field is not a text (memo) field
4 Edit text field error

DSP (Rarely)

Shows a record in the edit mask but the record cannot be modified.

Errors

1 Task is nil or table name is empty (= no file)

BRW <cconst | &cvar>[,<ivar>] (Occasionally)

Shows selected records / fields as a table.

Parameters

<cconst | &var>	Caption.
<ivar>	Optional variable for storing the selected record.

Remarks

The number of the selected (by double click) record is optionally stored in a variable.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed
3 Complex browse error (see history)
4 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    i,i
FILE      authors
RESET
SFLT      recno<50
BRW       "Authors",i
CON       i
```

PUT <dbfield>,<const | var | expression> (Regularly)

Stores a value into a data field.

Parameters

<dbfield>	Valid field name. This can be also the name of a field of an interconnected table.
<const | var | dbfield | expression>	Any expression, but it must have the same data type as the data field.

Remarks

Stores a value into a data field. Any PUT command should be followed by a FLSH to write the record back to the disk. Several PUT commands can be followed by a terminating FLSH, not each PUT needs a separate FLSH. The macro operator (&) is not allowed here, please compare to MPUT.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Could not write value into variable (because it is probably a constant)
4 Could not write value into variable (because the data types are not compatible)

Example

```
OPEN      species,4
FILE      genera
RESET
:beginreset
PUT       numspec,0
FLSH      ; !!!
SKIP
JNEOF     beginreset
FILE      species
RESET
:begin
PUT       c_genus.numspec,c_genus.numspec+1
; in interconnected tables no FLSH necessary
SKIP
JNEOF     BEGIN
CLA
EXIT
```

MPUT <dbfield | &cvar>,<const | var | expression | &var> (Rarely)

Stores a value into a data field with involved macro operators.

Parameters

<dbfield | &cvar> Valid field name. This can be also the name of a field of an interconnected table.

<const | var | dbfield | expression | &var> Any expression, but it must have the same data type as the data field.

Remarks

Stores a value into a data field. Any MPUT command should be followed by a FLSH to write the record back to the disk. Several MPUT commands can be followed by a terminating FLSH, not each MPUT needs a separate FLSH. The macro operator (&) is allowed here.

Errors

1 Memory allocation problem (anything internally that should not happen)
2 Parsing : one of the parameters is invalid (variable or data field probably unknown) or expression simply invalid
3 Could not write value into variable (because it is probably a constant)
4 Could not write value into variable (because the data types are not compatible)
5 Evaluating the expression caused an error
6 Could not write value into variable or data field
11 Macro evaluation of the first parameter failed
12 Macro evaluation of the second parameter failed

Example

```
; given table has numerous fields such as data1 to data30
; to reset this values to zero, 30 commands (PUT data1,0)
```

```
; would be needed, but with the MPUT command
; this can be shorter
DEFINE    ic,i
FILE      data
RESET
:begin
MOV       i,1
:begin2
MPUT      &(data+str(i)),0
MOV       i,i+1
CMP       i,31     ; > max
JNE       begin2
SKIP
JNEOF     begin
; the programme is shorter, but the execution time is longer
; because MPUT is more complex than PUT
```

FLSH (Regularly)

Writes the current data buffer back to the cache or HDD.

Errors

1 Task is nil or table name is empty (= no file)
2 Record number is zero or no current record (forgotten RESET command?)
3 Record writing error

Example

```
FILE      test
RESET
:begin
PUT       name,''
FLSH        ; if FLSH is lacking, the record will not be saved to the disk
SKIP
JNEOF     begin
EXIT
```

SETD (Rarely)

Marks the current record as being deleted.

Remarks

The record is not modified, just one character is set as mark.

Errors

1 Task is nil or table name is empty (= no file)
2 Record writing error

Example

```
FILE      authors
RESET
:begin
SETD
; no FLSH needed
SKIP
JNEOF     begin
```

CLRD (Rarely)

Undelete the current record.

Errors

1 Task is nil or table name is empty (= no file)
2 Record writing error

QDEL (Rarely)

Requests whether a record is marked as deleted.

Remarks

QDEL should be followed by a jump command (JE, JNE).

Errors

1 Task is nil or table name is empty (= no file)

Example

```
FILE      authors
RESET
:begin
QDEL
JNE       skip
con       'Deleted : '+fname
:skip
SKIP
JNEOF     begin
```

WRMEMO <cconst | &cvar>,<cconst | &cvar>[,<0 | 1 | 2>] (Very rarely)

Writes the content of a text (memo) field into a textfile.

Parameters

<cconst | &cvar>	The name of the data field.
<cconst | &cvar>	The name of the text file.
<0 | 1>	Optional order to convert the text into (1) ASCII, into (2) ANSI, or (0) no conversion at all.

Remarks

The command is rather rarely used, mainly for data conversion purposes.

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the first parameter failed
3 Macro evaluation of the second parameter failed
4 Invalid numerical expression
5 Complex text (memo) field output error

RDMEMO <cconst | &cvar>,<cconst | &cvar>[,<0 | 1 | 2>] (Very rarely)

Reads a text file into a text (memo) field.

Parameters

<cconst | &cvar>	The name of the data field.
<cconst | &cvar>	The name of the text file.
<0 | 1>	Optional order to convert the text into (1) ASCII, into (2) ANSI, or (0) no conversion at all.

Remarks

The command is rather rarely used, mainly for data conversion purposes.

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the first parameter failed
3 Macro evaluation of the second parameter failed
4 Invalid numerical expression
5 Complex text (memo) field output error

RSEL <cconst | &cvar>,<ivar>[,<iconst>] (Occasionally)

Selects an item from the current table.

Parameters

<cconst | &cvar>	Text
<ivar>	Variable to store the result.
<iconst>	Optional value that indicates whether it is allowed to append new items (1) or not (0)

Remarks

Table must be opened via OPEN not via FILE.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed
3 Key of the specified table is not defined (probably the table is not opened using OPEN)
4 Key of the specified table is invalid (nil)
5 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    ia,i
OPEN      authors,4
RSEL      'Select author',ia,0
```

RSUB <cconst | &cvar>,<dbfield>,<ivar>[,<iconst>] (Occasionally)

Selects an item from an interconnected table.

Parameters

<cconst | &cvar>	Text
<dbfield>	Data field that should be selected.
<ivar>	Variable to store the result.
<iconst>	Optional value that indicates whether it is allowed to append new items (1) or not (0)

Remarks

The data field must be an interconnection to another table.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed
3 Field is unknown
4 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    an,i
OPEN      publicat,4
;RSUB     "Select the year...",year,an    ; not allowed
RSUB      "Select an author...",author,an
RESET
:begin
CMP       author,an
JNE       skip
CON       year+#32+title
:skip
SKIP
JNEOF     begin
```

CRDB (Very rarely)

Creates a data buffer for the current table.

Remarks

This command saves the current record in a temporary buffer. This command is often necessary when during appending a record to a table the same table should be revised for any reason. If the current record is not saved, its data can get lost and it is better to save it using this command.

Errors

1 Task is nil or table name is empty (= no file)
2 Memory allocation problem (anything internally that should not happen)

Example

```
DEFINE    c,c
DEFINE    k,i
DEFINE    j,i
MOV       c,spsampl.smsite.smplpnt
CRDB      ; current buffer is saved
RESET     ; same file is used
:begin
CMP       substr(spmnno,1,5),c
JNE       skip
MOV       j,trunc(val(substr(spmnno,6,3)))
MOV       k,IFF(j>k,j,k)
:skip
SKIP
JNEOF     begin
MOV       k,k+1
MOV       c,c+IFF(k>99,'',IFF(k>9,'0','00'))+str(k)
DLDB      ; buffer is restored
PUT       spmnno,c ; one field modified
:exit
```

```
EXIT
```

DLDB (Very rarely)

Restores the data buffer created by CRDB.

Errors

1 Task is nil or table name is empty (= no file)

SAVEREC (Very rarely)

Saves the content of the current record.

Remarks

The current record of a table is saved into an internal buffer. This operation is only necessary for complicated database operation.

Errors

1 Task is nil or table name is empty (= no file)

RESTREC (Very rarely)

Restores the content of the current record.

Errors

1 Task is nil or table name is empty (= no file)

Database operation : index and find

INDEX <var | expression>,<cconst> [, <O | U | Q> [,<iconst>[,<iconst>]]] (Regularly)

Creates an index on the expression and saves it under the given filename.

Parameters

<var | expression>	Expression for the index.
<cconst>	Index file name.
<O | U | Q>	The optional mode stands for O(verwrite), U(se) or Q(uestion) and is applied if the index file exists and is valid.
<iconst>	Optional length of the index expression.
<iconst>	If this value ist set to 1, the index is set to the unique mode.

Remarks

An index is created when data should be classified for output or analyse. Calling the comand without parameter closes the current index.

Errors

1 Macro evaluation of the first parameter failed
2 Error closing current index
3 Complex index creation error (see history for details)
4 Cannot find first record
5 Error reading the record

Example

```
FILE      authors
RESET
INDEX     fname+cname,author
STRM      liste.txt
:begin
OUTL      fname+', '+cname
SKIP
JNEOF     begin
STRM
INDEX
```

FIND <const | var | expression> (Occasionally)

Searches an activated index for an expression.

Parameters

<const | var> The search expression. This can be also a variable.

Remarks

Before using FIND, an index must be created.

Errors

1 There is no index activated
2 Search expression could not be resolved (scanning or parsing error)
3 Error in the find subroutine (index corrupted or not up to date?)
4 Error reading the record

Example

```
FILE      publicat
INDEX     year,yr
FIND      '1980'
JNF       notfound
:begin
CMP       year,'1980'
JNE       notfound
CON       title
:skip
JNEOF     begin
:notfound
CON       'Not (more) found.'
INDEX
FILE
```

JNF <label> (Occasionally)

Continues the programme at the label when the expression was not found.

Parameters

<label> A label in the source code

Remarks

Can only be used directly after FIND. There is an option (which is generally set) which says that the condition is also fulfilled by any item larger than the one searched for. So it is necessary to control whether the found item fulfils the condition. If there is no index activated, the result is equal to 'not found'.

Example

```
FILE      publicat
INDEX     year,yr
FIND      '1980'
JNF       notfound
:begin
CMP       year,'1980'         ; here it is checked whether the condition is
          fulfilled
JNE       notfound
CON       title
:skip
JNEOF     begin
:notfound
CON       'Not (more) found.'
INDEX
FILE
```

USEIND [<cconst>] (Rarely)

Closes an index or reopen an index.

Parameters

<cconst> Index file name

Remarks

The use of the command without parameter just closes the current index. Errors may occur if the index file is not consistent with the table.

Errors

1 Error closing current index
2 Error opening (new) index
3 It was not possible to go to the beginning of the table (empty?)
4 Error reading the record

Example

```
FILE      authors
INDEX     fname,fn
...
USEIND
EXIT
```

XCHI (Occasionally)

Saves index data when a programme is called from the database.

Remarks

XCHI is a command that is very often applied in programmes that form part of a application. It should avoid that an index created within the application is overwritten by the programme called. XCHI should be called always twice: at the beginning and at the end of the programme.

Errors

1 Task is nil or table name is empty (= no file)

Database operation : data fields and structure

FLDEXS <cconst | &cvar> (Occasionally)

Checks whether a specified field exists in a table.

Parameters

<cconst | &cvar> Fieldname.

Remarks

The command should be followed by a JE or JNE command.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed

Example

```
FILE     authors
FLDEXS   fullname      ; check whether the fieldname exists
JE       nonewfield
ADDFLD   'fullname,c,30'
:nonewfield
```

GFN <cconst | &cvar>,<ivar> (Rarely)

Stores the current number of a data field in a variable.

Parameters

<cconst | &cvar> Name of the data field.
<ivar> Variable where the number has to be stored.

Remarks

If the field does not exist, a zero is written in the variable.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed
3 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE   ifld,i
FILE     authors
GFN      fname,ifld
CON      ifld

1
---
OK.
```

FLDD <iconst | &ivar>,<cvar> (Rarely)

Stores the data of a data field in a variable.

Parameters

<iconst | &ivar>	The field number.
<cvar>	A character variable to store the result.

Remarks

It is the same format as required by the ADDFLD command. If the field number is zero or above the number of data fields, the resulting string is empty.

Errors

1 Task is nil or table name is empty (= no file)
2 Macro evaluation of the first parameter failed
3 Invalid numerical expression
4 The value could not be assigned to the variable (possibly a data type error)

Example

```
DEFINE    s,c
FILE      authors
FLDD      1,s
CON       s

FNAME,C,25,
---
OK.
```

ADDFLD <cconst | &cvar> (Rarely)

Adds a new data field to a table.

Parameters

<cconst | &cvar >	Format description of the new field in the form name,type,length[,decimals]. Valid types are C (character), N (numerical), M (text), and L (logical). Decimals are only required in case of a float number. The field description must be in apostrophs. Valid description would be the following: name,c,100 year,n,4 dtext,m,10 mark,l,1

Remarks

The file should not be opened via OPEN, only via FILE. An error occur when the field already exists.

Errors

1 Task is nil or table name is empty (= no file)
2 Table is opened via OPEN not via FILE
3 Macro evaluation of the first parameter failed
4 Memory allocation problem (anything internally that should not happen)
5 Complex modify structure error (see history)
6 Memory allocation problem (anything internally that should not happen)

Example

```
FILE      authors
```

```
FLDEXS    fullcname           ; check whether the fieldname exists
JE        nonewfield
ADDFLD    'fullcname,c,30'
:nonewfield
```

CPS <cconst | &cvar>[,<O | Q>] (Rarely)

Copies the structure of a table into new table.

Parameters

<cconst | &cvar>	Target file name.
<O | Q>	Optional mode in the case the target file already exists, O for over write, Q for question.

Errors

1 Task is nil or table name is empty (= no file)
2 Target file name is empty
3 Macro evaluation of the first parameter failed
4 Complex table copy error (see history)

CFL (Rarely)

Clears the field list.

Remarks

Clears the list of data fields. See the commands FFL and AFL.

Example

```
OPEN      publicat,4
CFL               ; clear the list
AFL       author.key          ; fill the list
AFL       year
AFL       title
BRW               ; show as table
```

FFL (Rarely)

Fills the field list for copy or browse procedures with all fields.

Remarks

In contrary to CLF, FFL fills the list with all data fields.

Errors

1 Task is nil or table name is empty (= no file)

Example

```
OPEN      publicat,4
FFL         ; show all fields
BRW         ; as table
```

AFL <cconst | expression | var > (Rarely)

Adds a field (or expression) to the fields list.

Parameters

<cconst | expression | var | dbfield> Any expression to be shown.

Remarks

Adds an expression to the field list. This must not be a data field, it can be also a complex expression with variables and constants involved. The expression is not checked by the interpreter but by the function applied (CPY, BRW). Expressions cannot be used with CPY.

Errors

1 Parameter(s) is/are lacking
2 Maximum of the field list is reached

Example

```
FILE      authors
AFL       fname
AFL       cname
AFL       fname+', '+cname
BRW
```

DFL <cconst | expression | var > (Rarely)

Removes a field (or expression) from the fields list.

Errors

1 Parameter(s) is/are lacking

Database operation : whole table operations

CPY <cconst | &cvar>[,<A | B | O | Q>] (Rarely)

Copies records to a new table from the field list.

Parameters

<cconst | &cvar>	Target file name.
[,<A | B | O | Q>]	Optional mode when the file already exists: O for overwrite), B for backup, A for append) or Q for question.

Errors

1 Task is nil or table name is empty (= no file)
2 Parameter(s) is/are lacking
3 Macro evaluation of the first parameter failed
4 Complex table copy error (see history)

Example

```
FILE      authors
FFL
CPY       authors2,Q
```

ADDF <cconst | &cvar> (Rarely)

Adds records to an existing table from the field list.

Parameters

<cconst | &cvar> Target file name.

Remarks

Field names have priority. The routines only adds fields that exist in both tables.

Errors

1 Task is nil or table name is empty (= no file)
2 Parameter(s) is/are lacking
3 Macro evaluation of the first parameter failed
4 Complex table copy error (see history)

ADDR <cconst | &cvar> (Rarely)

Adds records to an existing table from the field list.

Parameters

<cconst | &cvar> Target file name.

Remarks

The record size has priority. If both tables do not have the same record size, no records will be added. Fields are not taken into account.

Errors

1 Task is nil or table name is empty (= no file)
2 Parameter(s) is/are lacking
3 Macro evaluation of the first parameter failed
4 Complex table copy error (see history)

CMPR <cconst | &cvar> (Very rarely)

Writes a table, but compresses the table according to an index.

Parameters

<cconst | &cvar> Target file name.

Remarks

Records that have the same content of an index impression, are only written once. This command is mainly used during importation processes, to remove double items.

Errors

1 Task is nil or table name is empty (= no file)
2 Parameter(s) is/are lacking
3 Macro evaluation of the first parameter failed
4 Complex table copy error (see history)

PCK (Rarely)

Packs a table. Records marked as being deleted are removed.

Remarks

This is a very dangerous command and should not used of tables that are interconnected with other tables.

Errors

1 Task is nil or table name is empty (= no file)
2 Complex table copy error (see history)

CHBI (Very rarely)

Initiation of the search for unused records.

Remarks

CHBI marks all records of all tables of the currently opened database as deleted. The following CHB command – wisely used – then recalls used records. Unused records keep marked as being deleted. Both commands are rather for internal use.

Errors

1 Complex check base error

CHB (Very rarely)

Search for unused records.

Errors

1 Task is nil or table name is empty (= no file)
2 Table is opened via FILE not via OPEN
3 Complex check base error

System options

GSYS <ivar | iconst> , <cvar> (Very rarely)

Stores an internal system value in a variable.

Parameters

<ivar>	Number of the internal system variable.
<cvar>	Char variable to store the result.

Remarks

The Hdb2Win kernel system and the application library have about 130 internal system variables that can be read and partly modified. This command is for reading the variable. Unless of their real data type, these variables are always kept as char variables. For setting se SSYS.

Errors

1 Evaluating the first parameter caused an error
2 First parameter is beyond allowed limits
3 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    csys,c
GSYS      26,csys  ; 26 = language, 1 = English
```

```
CON       csys
DEFINE    i,i
MOV       i,1
GSYS      i,csys
CON       csys

1
Times New Roman
---
OK.
```

SSYS <iconst | ivar>,<cconst | &cvar> (Rarely)

Sets an internal system variable.

Parameters

<iconst | ivar>	Number of the intern symbol.
<cconst | &cvar>	New value.

Remarks

The value to be set ist always a character.

Errors

1 Evaluating the first parameter caused an error
2 Invalid numerical expression
3 Macro evaluation of the second parameter failed
4 Could not set system variable

Example

```
SSYS      36,''
```

NSYS <icont | ivar>,<svar> (Rarely)

Returns the name of an internal system variable.

Parameters

<icont | ivar>	Number of the internal variable.
<svar>	Return value is written to this variable.

Errors

1 Evaluating the expression caused an error
2 Variable does not exist
3 Could not write value into variable (not exist? wrong type?)

Example

```
DEFINE    sy,c
NSYS      1,sy
CON       sy

HDFRM.Font0
---
OK.
```

REG.RINT <cconst | &cvar>,<ivar> (Rarely)

Reads an integer value from the internal registry.

Parameters

<cconst | &cvar>	Name of the key
<ivar>	Return value of the key

Remarks

Reads an integer value from the internal registry. If the key is unknown to the registry, the value zero is returned. If the data type of the variable to be read is not an integer, the value zero is returned. Please compare to REG.WRITE for more details.

Errors

1 Macro evaluation of the first parameter failed
2 Could not write value into variable (not exist? wrong type?)

Example

```
define   i,i
reg.rint hdb2win.user.runcount,i
con      'Run count = '+str(i)
define   rr,n
reg.rint hdb2win.user.runcount,rr
; rr is zero because the expected data type is integer
```

REG.RREAL <cconst | &cvar>,<rvar> (Rarely)

Reads a real value from the internal registry.

Parameters

<cconst | &cvar>	Name of the key
<rvar>	Return value of the key

Remarks

Reads a real value from the internal registry. If the key is unknown to the registry, the value zero is returned. If the data type of the variable to be read is not a real, the value zero is returned. Please compare to REG.WRITE for more details.

Errors

1 Macro evaluation of the first parameter failed
2 Could not write value into variable (not exist? wrong type?)

Example

```
define   r,n
reg.rreal hdb2win.user.change,r
con      'Change = '+str(r)
```

REG.RSTR <cconst | &cvar>,<cvar> (Rarely)

Reads a string value from the internal registry.

Parameters

<cconst | &cvar>	Name of the key

<cvar> Return value of the key

Remarks

Reads a string value from the internal registry. If the key is unknown to the registry, an empty string returned. If the data type of the variable to be read is not a string, an empty string returned. Please compare to REG.WRITE for more details.

Errors

1 Macro evaluation of the first parameter failed
2 Could not write value into variable (not exist? wrong type?)

Example

```
define    c,n
reg.rstr  hdb2win.user.name,c
con       'Name = '+c
```

REG.WRITE <cconst | &cvar>,<var> (Rarely)

Writes a value into the internal registry.

Parameters

<cconst | &cvar> Name of the key
<var> Value of the key

Remarks

The internal registry allows to keep values of user defined variables. They can be red and writen only by the interpreter. Keys defined by the user should always start with the text hdb2win.user. to separate them from internal keys. The registry data are not written into the Windows Registry but a text file in the user data area. Please take care that assignation of values to a key or reading a key from the registry is only by the means of variables. This is necessary to conserve type comptability.

Errors

1 Macro evaluation of the first parameter failed
2 Variable does not exist
3 Evaluating the variable caused an error
4 The variable has not the required datatype (you cannot use data fields)
5 The variable could not be written to the registry

Example

```
define    iarea,i
define    q,i
file      LocAreas
reset
reg.rint  hdb2win.user.area,iarea   ; read from registry
cmp       iarea,0             ; not found in the registry or not defined
je        SelectArea
go        iarea
req       &('Do you want to use this area : '+areaname+' ?'),4,q
cmp       q,6     ; yes
je        SelectAreaDone

:SelectArea
rsel      Select area,iarea,0
reg.write hdb2win.user.area,iarea   ; write into registry
```

```
:SelectAreaDone
```

External libraries

EXEC [<cconst | &cvar>] | [,<cconst | &cvar>] (Occasionally)

Calls a external programme, optionally with a parameter.

Parameters

<cconst | &cvar>	Name of the programme. If no programme is given, Windows is looking for the most convenient programme, depending on the parameter.
<cconst | &cvar>	Optional parameter. The parameter is normally a file that should be opened or an Internet page that should be visisted.

Remarks

There must be at least one parameter (the programme name). A programme parameter is optional.

Errors

1 Parameter(s) is/are lacking
2 Macro evaluation of the first parameter failed
3 Macro evaluation of the second parameter failed
4 Complex execute error

Example

```
EXEC      notepad.exe,data.txt
EXEC      www.paleotax.de
EXEC      C:\Programme\MSOffice\Office10\EXCEL.EXE
EXEC      helloworld.txt
EXEC      data.rtf
EXEC      index.htm
```

CVT <cconst | &cvar>,<cconst>[,<cconst | &cvar>[,<cconst>[,<0 | 1>]]] (Occasionally)

Text conversion.

Parameters

<cconst | &cvar>	Source file.
<cconst>	Format description file (style sheet).
<cconst | &cvar>	Optional targetfile.
<cconst>	An optional letter stands for (A)ppend, (B)ackup, (O)verwrite, and (Q)uest, if the file already exists. The default value is Q.
<0 | 1>	If the optional parameter is set to '1', the command will not display any messages (quiet mode).

Remarks

The commands converts an ASCII file into a RTF file, taking into account the format specification.

Errors

1 Macro evaluation of the first parameter failed
2 Macro evaluation of the third parameter failed

3 Target cannot be the console
4 Target file name is empty
5 Invalid option for the decision taken when the target file exists
6 Text conversion failed (complex, often a format error, see history)

GRAPH <cconst | &cvar> [,1] (Rarely)

Call of the external PaleoTax/Graph module.

Parameters

<cconst | &cvar>	Name of the input file. This must be a text file in the PaleoTax/Graph (PGR) format.
[,1]	Auto run. If this optional parameter is set, the WMF is created and the application is closed. Not implemented for all functions.

Remarks

The command calls the PaleoTax/Graph module. The programme of the interpreter is stopped as long the window of the PaleoTax/Graph is open. For the file format compare to the documentation on PaleoTax/Graph. Any error occurring in PaleoTax/Graph are shown after calling the command, so the success of the execution has no influence on the programme execution of the interpreter. – From Hdb2Win Version 2.5 on.

Errors

1 Empty file name
2 Macro evaluation of the first parameter failed
3 Screen size insufficient for the external library
4 PaleoTax/Graph initial error (cannot read error definitions)
5 PaleoTax/Graph initial error (cannot read messages)

Example

```
GRAPH     mychart.pgr
```

VECDRAW <cconst | &cvar>[,<cconst>] (Rarely)

Calls the Vector programme.

Parameters

<cconst | &cvar>	Name of the input file. This must be a text file in the PaleoTax/Graph vector format.
<cconst>	Optional configuration file. This file will be saved with the changes made in the PaleoTax/Graph application and can be used again to apply the same options.

Remarks

The vector graphic module (that forms part of PaleoTax/Graph) can be called directly from Hdb2Win. The programme of the interpreter is stopped as long the window of the VectorDraw module is open. For the commands compare to the documentation on PaleoTax/Graph. – Only from Hdb2Win Version 2.4.2 on. Please use from version 2.5 only GRAPH.

Errors

1 Macro evaluation of the first parameter failed
2 Textfile not found or could not be opened
3 Screen size insufficient for the external library
4 PaleoTax/Graph initial error (cannot read error definitions)
5 PaleoTax/Graph initial error (cannot read messages)

Example

```
strm      vectest.pgr,o
outl      'goto   10,10'
outl      'rect   100,100'
outl      'goto   20,50'
outl      'text   "Hello, World",24,Arial,red,1'
strm
vecdraw   vectest.pgr
exit
```

VECCHRT <cconst | &cvar>,<cconst | &cvar>[,<cconst>] (Rarely)

Call of the external PaleoTax/Graph Chart module.

Parameters

<cconst | &cvar>	Name of the input file. This must be a text file in the PaleoTax/Graph chart format.
<cconst | &cvar>	Name of the raster. This must be a text file in the PaleoTax/Graph raster chart format.
<cconst>	Optional configuration file. This file will be saved with the changes made in the PaleoTax/Graph application and can be used again to apply the same options.

Remarks

The command calls the Chart module of PaleoTax/Graph. The programme of the interpreter is stopped as long the window of the VectorDraw module is open. For the file format compare to the documentation on PaleoTax/Graph. – Only from Hdb2Win Version 2.4.2 on. Please use from version 2.5 only GRAPH.

Errors

1 Macro evaluation of the first parameter failed
2 Textfile not found or could not be opened
3 Macro evaluation of the second parameter failed
4 Textfile not found or could not be opened
5 Screen size insufficient for the external library
6 PaleoTax/Graph initial error (cannot read error definitions)
7 PaleoTax/Graph initial error (cannot read messages)

Example

```
VECCHRT   mychart.pgr,raster.psc,mychart.cfg
```

Index

Hdb2Win / PaleoTax © H. Löser 1993-2021
Interpreter

Version	2.1
Published	May 2021
Internet	http://www.paleotax.de
E-Mail	info@paleotax.de
Document	E:\T\DOKUM\INTERPR\IP-2-1b.DOC