SWT
SOFTWARE TOOLS

English User Manual

Duevej 23
DK-2680 Solrød Strand
Denmark
Phone: **+45) 33 33 05 56**
Mail: **swtools@swtools.com**
www: **www.swtools.com**

Calculations and subfunctions

22/11/01 / 2022-09-01 008.384

Contents

1. Introduction

The the syntax of the calculations written in RAPGEN is based on a BASIC-like language. This language allows test on field values, arithmetic statements, text processing and much more.

	Reserved words	Synonym	Description
Conditional statements			
	IF		IF expression ...
	LET		IF expression LET expression ...
	ELSE		IF expression ... ELSE expression
Block statements			
	BEGIN		start block
	END		end block
Control loop flow			
	BREAK		exit from loop
	CONTINUE		continue loop
Logical operators			
	NOT		not equal to
	AND		and
	OR		or
Arithmetic operators			
	+		addition
	-		subtraction
	*		multiply
	/		divide
	%		percentage
Relational operators			
	=		equals
	>		greater than
	<		less than
	>=		greater than or equal to
	<=		less than or equal to
	<>		not equal to
Comments			
	REM		full comment file
	/*		comment after the statement
Jump and subroutines			
	GOTO		go to label:
	GOSUB		execute subroutine label:
	RETURN		return from subroutine
	ON..GO..		conditional branch

This language syntax provides you with lots of ways to write statements. We now give some examples of this:

1.1. Examples

1.1.1. **IF..ELSE** - Conditional statements

If supplier balance (LE#6) is over 1000 subtract 100 else add 47.

```
IF LE#6 > 1000 LET LE#6 = LE#6 - 100 ELSE LET LE#6 = LE#6 + 47
```

1.1.2. **BEGIN..END** - Block statements

If supplier balance (LE#6) id over 1000 then start block where 100 is subtracted from the balance and line 7 is printed.

```
IF LE#6 > 1000 THEN BEGIN
LE#6 = LE#6 - 100
PRINT(7)
END
```

Which means, that all lines between BEGIN and END are performed only if the condition is true.

1.1.3. START/END...NEXT...REPEAT - Loops

The following loop reads all suppliers in the range 111-999. If the balance is less than 1000 the supplier is not processed.

```
START(LE),"111"
END(LE),"999"
NEXT(LE)
IF LE#6 < 1000 CONTINUE /* skip suppliers with a balance < 1000
REM *** process suppliers ***
REPEAT(LE)
```

The following loop reads all suppliers in the range 111-999. When a balance greater than 10000 is met the loop is ended.

```
START(LE),"111"
END(LE),"999"
NEXT(LE)
IF LE#6 > 10000 BREAK  /* break loop if balance > 10000
REPEAT(LE)
IF LE#6 > 10000 ....   /* supllier found with balance > 10000
```

1.1.4. **NOT, AND, OR** - Logical operators

```
IF NOT VA#5 LET VA#5=#DD /* date last purchase is set to todays date
```

If date last purchase equals 0 the purchase date is set to todays date. This statements equals

```
IF VA#5=0 LET VA#5=#DD
```

If costprice not equal to 0 and date last purchase not equal to 0, then print line 5 on the report.

```
IF VA#4<>0 AND VA#5<>0 PRINT(5) /* if cost and date set print line
```

If costprice not equal to 0 OR date last purchase not equal to 0, then print line 5 on the report.

```
IF VA#4<> OR VA#5<>0 PRINT(5) /* if cost or date not equals 0 print line 5
```

1.1.5. REM, /* - Comments

```
REM *** this report is developed by SW-Tools ApS ***
REM *** date. 07.09.1997
IF LE#6 > 1000 LET LE#6 = LE#6 - 100 /* Ajust the balance
```

1.1.6. **GOTO** Jump to label

Using the GOTO statement you can jump in the calculations usually dependent on the value of a field. A label defined as 'NAME:' decides where to jump to. In the example below line 7 is printed three times.

```
          #30 = 0                 /* Zero counter
   AGAIN:                         /* Label for later jump
          PRINT(7)                /* Do something
          #30 = #30 + 1           /* Count
          IF #30 < 3 GOTO AGAIN   /* Do it three times
```

1.1.6.1. **ON**...GOTO/GOSUB - Conditional jump / subroutinecall

Conditioanl jump to a label or calling a subroutine dependent on the value in a field can be done using ON. ON may be used both with GOTO and GOSUB.

```
        #30 = 0
        ON #7 GOTO ONE,TWO,ONE
        #30 = #30 + 1         /* #30 becomes 3 if field 7 not equals 1,2 or 3
   TWO: #30 = #30 + 1         /* #30 becomes 2 if field 7 equals 2
   ONE: #30 = #30 + 1         /* #30 becomes 1 if field 7 equals 1 or 3
```

1.1.7. **GOSUB** Calling subroutines

If the same calculations are to be done several times you may write these lines as a subroutine starting with a 'label:' and called with GOSUB

```
        #30 = 0                     /* Zero counter
        GOSUB DOIT                  /* Call the routine
        GOSUB DOIT                  /* Call the routine again
        RETURN                      /* End the normal calculations
  DOIT: #30 = #30 + 1               /* Routine DOIT, count up field 30
        PRINT(7)                    /* And print line 7
        RETURN                      /* Return from the subroutine
```

1.1.7.1. **RETURN** Returning from a subroutine

A subroutine is ended with RETURN whereafter the calculations will be continued from where the call took place. Also refer to the RETURN function descriped later where a value may be returned from the calculations.

1.2. Fields

1.2.1. #xx or kk#xx - Fields from a file

You can refer to a field from a file as:

```
     #xx = fieldnumber xx from the mainfile
   kk#xx = fieldnumber xx from the file kk
```

Note that kk, KK, Kk and kK references different records from a file, you should normaly use the lowercase kk.

1.2.1.1. #xx(from,to) - Part of fields

Part of fields are written as kk#xx(from,to) and you may use this syntax for both numeric and alphanumeric fields.

```
#30 = #2(3,4)      /* Field 30 becomes character 3 thru 4 of field 2
```

For alphanumeric textfields and only for these you may also assign a value to a part of a field:

```
#2="Sorenco and Son Ltd."
#2(9,15)="xx"      /* Field 2 becomes "Sorenco xx Ltd."
```

1.2.1.2. #xx(no) - Tabelfields

Tabelfields are referred as kk#xx(no) where no is in the range 0 until max.
A field may be defined as a table field in the data dictionary if the format contains eg. 20(003) specifying 3 extra elements in the table or it may just be a set of contigous fields with the same format which you would whish to use as a tabel in the calculations. Note that the freefields may also be defined as tabelfields with the format specification.
An example of this is the demo-supplier file where the name block #2, #3 and #4 also can be used as a table as #2(0), #2(1) and #2(2)

```
#30 = #2(#31)              /* Freefield 31 specifies name line 0,1 or 2
PRINT(7)                   /* Which is printed
#2(#31)="xx"               /* And set to "xx"
```

Note that crytical values may occur if you exeeds the maximum of a table eg. by using #2(4)

1.2.1.3. Conversion between numeric and textfields

You may just set a numericfield = a textfield as #30 = #2 to convert to numeric and calculated with the numeric value. The functions NUMBER and NUMS may be used for more advanced conversions, see these.
In case of a textfield = a numericfield as #2 = #30 the result will be a textstring of variable length dependent the number as "123". Normally #2 = #30 USING "#####" is used to specify the layout of the resulting textfield, see the USING function.

1.2.2. **SY#xx** - System fields

System fields are special fields defined in the pseudofile SY which will always be present. A few of the system fields are descriped in the following, for a complete list see your actual SY file definition.
A system field is referred either by number SY#1 or by shortname #DD as stated in the first part of the fieldname. Some of the system fields are associated to a file and must be given as kk#shortname as kk#RECNO

1.2.2.1. **#DD, #PD** - Today's date and As of date

Entered at the beginning of a report, (99.99.99).

1.2.2.2. **#PP** - Pagenumber

Is automatically assigned during page shift, (9999).

1.2.2.3. #SN - System name

May be used if RAPGEN is installed with multiple systems e.g. different companies/files sets. Also note the fields #SU containing subsystem name and #CN with company name.

1.2.2.4. #OK - Result after reading of a file

After reading of a file you may use #OK, This field will be 0 if a record was read, anything else indicates error.

1.2.2.5. **#UN** User name

You may use #UN to get the user name for this PC entered by the LICENSE module.

1.2.2.6. **#LIN** linenumber and #LOF lines on form

#LIN contains the current printline, #LOF the actual number of lines on form.

1.2.2.7. #LEVEL - Current total level

With #LEVEL you may control calculations / print dependent of the subtotal level, see the RAPGEN user manual.

1.2.2.8. kk#RECNO - Last used recordnumber from the file kk

If the used database system is connected with recordnumbers the last used for file kk can be found in kk#RECNO. Also note the fields kk#NUMBER containing relative recordnumber and kk#FILENAME

1.2.3. **WW#xx** - Freefields (Workfields)

A program will be creation be assigned 40 workfields which must be defined when first time used and which may later be changed by doubleclick on the field.
The fieldnumbers will be shown as a continuation of the fields in the mainfile but the fields are actually stored as WW#1,WW#2,... whereby a later change of the number of fields in the mainfile causes automatic renumber of the free fields in all programs.
The number of freefields may be ajusted in IQ/DATAMASTER with the programparameter function, in RAPGEN by in the calculations just using a higher number than shown in the listbox which causes the number of freefields to be extended automatically.

1.2.3.1. **#Dntext** - Input data

In RAPGEN a freefieldname beginning with #Dn defines input field 1 to 7 to be entered by start of the report.

1.2.3.2. **#Ptext** - Picturefields

A freefieldname beginning with #P and defined as textfield is a reference to a picture.

2. Aritmetic functions

This section describes functions for numeric calculations such as rounding and power.

2.1. **ABS** - The absolute value of a number

number ABS(number *par1*)
Parameters: *par1* : number to be converted to an absolute value
Description: The function returns the absolute value of the parameter *par1*. Eg. the positive value without sign.
Returnvalue: The positive value.
See also: SGN
Example: #1 = ABS(-123.45) /* *Field #1 contains the value 123.45*

2.2. **FNH** - Round number - no decimals

number FNH(number *par1*)
Parameters: *par1* : defines a number (with decimals)
Description: The function is used to round a number with decimals to a number without decimals.
Returnvalue: The number without decimals.
See also: FNR, RUN
Example: #1 = FNH(1234.56) /* Field #1 contains the value 1235

2.3. **FNR** - Round number to 2 decimals

number FNR(number *par1*)
Parameters: *par1* : defines a number (with decimals)
Description: The function is used to round a number with more than 2 decimals to a number with only 2 decimals. RAPGEN always round a result to the number of decimal digits given in the field format. You can override this by calling functions as FNH/FNR.
The rounding may be controlled by use of the RUND funktion. This defines:

Returnvalue: The rounded number.
See also: FNH, RUN, RUND
Example: #1 = FNR(123.456) */* Field #1 contains the value 123.46*

2.4. **FRA** - Calculate the fraction of a number

number FRA(number *par1*)
Parameters: *par1* : the number (with decimals)
Description: The function separates the fractional value from a number and returns it.
Returnvalue: The fraction as 0.<fractional value>.
See also: FNH, FNR, RUN
Example: #1=FRA(123.456) /* *gives 0.456* , #1=FRA(-12.345) /* *gives -0.345*

2.5. **INT** - The integer value of a number

number INT(number *par1*)
Parameters: *par1* : defines a number
Description: The function returns the integer value, it is the nearest lower value without decimals.
Returnvalue: The integer value.
See also: FRA
Example: #1=INT(1234.56) /* *gives 1234* , #1=INT(-12.34) /* *gives -13*

2.6. **NOT** - Logical negation

number NOT(number *par1*)
Parameters: *par1* : defines a number
Description: The function returns 1 if par1 equals zero, 0 if par1 is unequal to zero.
Returnvalue: 0 or 1.
See also: SGN
Example: NOT(1) *is 0*

2.7. **POW** - Raise to n'th power

number POW(number *par1*, number *par2*)

par2 : defines the exponent
Description: The function raises a number *par1* to the *par2* power.
Returnvalue: The n'th power.
See also: SQR
Example: #1=POW(8,3) */* *gives 512 (8*8*8)* , #1=POW(4,0.5) */* *gives 2*

2.8. RUN - Rounding to x decimals

number RUN(number *par1*, number *par2*)

par2 : No of decimals to round to
Description: The RUN function rounds the given figure to the given number of decimals.
Returnvalue: The rounded figure.
See also: FNH, FNR, INT
Example: #1=RUN(-123.4567,3) /* *Field 1 becomes the value -123.457*

2.9. RUND - Definition of the FNR rounding function

number RUND(number *par1*, number *par2*)

par2 : The number of decimals to round TO, eg. 2
Description: The RUND function defines how the FNR function is doing the rounding. If par1 is positive FNR will round UP, if par1 is negative FNR will round DOWN.
Returnvalue: None.
See also: FNR
Example:

```
RUND(-25,2)      /* Round DOWN to nearest 25 pence with 2 decimals
RUND(5,2)        /* Round UP to nearest 5 pence
RUND(1,3)        /* Round to 3 decimals
RUND(1,2)        /* FNR function will work as default
```

2.10. SGN - Check if number is negative, zero or positive

number SGN(number *par1*)
Parameters: *par1* : defines a number
Description: The function examines if the number is negative, zero or positive.
Returnvalue:

```
  -1       The number is negative
   0       The number is zero
   1       The number is positive
```

See also: INT, NOT
Example: #1=SGN(-123.45) */* *Field #1 then contains the value -1.*

2.11. SQR - Calculate the square root of a number

number SQR(number *par1*)
Parameters: *par1* : the number to take the square root of
Description: The function calculates the square root of the number in *par1*.
Returnvalue: The square root.
See also: POW
Example: #1=SQR(4) */* Gives 2*

3. String functions

This section describes functions for conversion of textfields and for converting numeric fields into strings.

3.1. CONV - Change characters in a text

text CONV(text *par1*, text *par2*, text *par3*)

par3 : the new characters to be inserted
Description: The function tests each character in the text *par1*. If the character equals one of those in *par2*, it will be changed with the new character in *par3*. If parameter 1 contains "abc" and parameter 2 the text "ABC", the function will replace a with A, b with B and c with C.
Returnvalue: The text where the requested characters are converted.
See also: LOWER, SMAA, UPPER
Example: #1 = CONV("hans", "hn", "lr") */* *Gives "lars"*

3.2. **EDIT** - Editing of an integer figure

text EDIT(number *par1*, text *par2*)

par2 : USING mask for editing
Description: The EDIT function converts an integer figure to a textfield. The USING mask determins the layout of the text.
Returnvalue: The edited textfield.
See also: NUMBER, USING
Example:

```
#1 = EDIT(-123,"&&&,&&")        /* Returns "001,23"
#1 = EDIT(123,"##&-#&&&")       /* Returns "  0- 123"
#1 = EDIT(123,"eg.# and ##")    /* Returns "eg.1 and 23"
```

3.3. **FIND** - Find text in textfield

number FIND(text *par1*, text *par2*, number *par3*, number *par4*, number *par5*)

Description: The function search for the text *par1* in the text *par2*. Both parameters has to be given in "" (quotes).
Returnvalue: Returns -1 if the text is not found, otherwise a positive number equal to the position where the text was found (origin 1).
See also:
Example:

```
#1 = "This is a text"
#2 = FIND("te", #1)          /* Field #2 contains the value 11.
```

3.4. LEN - Length of a text

number LEN(text *par1*)
Parameters: *par1* : defines a text
Description: The function calculates the length of a text.
Returnvalue: The length of the text.
See also: SPOFF
Example:

```
#1 = "SW-Tools ApS"
#2 = LEN(#1)        /* returns the length of the text
```

Field #2 then contains the value 12, because there are 12 characters in #1.

3.5. LOWER - Convert text to lower case characters

text LOWER(text *par1*)
Parameters: *par1* : defines a text to be converted
Description: The function converts a text to small letters, eg. all letters A-Z are converted to a-z.
Returnvalue: The converted text.
See also: CONV, SMAA, UPPER
Example:

```
#1 = "THIS is a TEST"
#2 = LOWER(#1)   /* Field #2 then contains the text "this is a test"
```

3.6. NAME - Extraction of Christian and lastname

text NAME(text *par1*, number *par2*)

Description: The function extracts best possible Christianname and Lastname from the given sourcename and returns the name as specified by *par2*. This value may be used for sorting.
The SSV textfile WORDS.ENG is used for this. Each line contains a specialword as Mr., Miss, Mrs and their eventual replacements (Mister;Mr.)
Returnvalue: The name as specified by *par2*.
See also: SMAA, SOGE
Example:

```
#1 = NAME("MR CHRIS HANSON",0)    /* Gives "HANSON, CHRIS Mr."
#1 = NAME("OLSEN, MICHAEL",1)     /* Gives "MICHAEL OLSEN"
```

3.7. NUMBER - Conversion of 'dirty' numbers

number NUMBER(text *par1*)
Parameters: *par1* : A text contaning a number
Description: The NUMBER function extracts a value from a textfield without looking at any charecters not being digits.
Returnvalue: The extracted integer figure, no decimals are returned.
See also: EDIT, NUMS, USING
Example:

```
#1=NUMBER("33)33 05 56")  /* Covert phonenumber to value 33330556
#1=NUMBER("31/03-1997")   /* A date is converted to the value 31031997
#1=NUMBER("ab1cd2&3.4")   /* Returns 1234
```

3.8. NUMS - Conversion of textfield to number

number NUMS(text *par1*)
Parameters: *par1* : A text contaning a number
Description: On a line containing #1=#2 where #1 is numeric and #2 is a textfield, any number in field 2 will be converted automatically. Same result could be reached using #1=NUMS(#2) but NUMS is optional.
However if you in such a line wants to calculate directly on the field values NUMS must be used to exactly specify the conversion as in: #1=NUMS(#2)+NUMS(#3)
Returnvalue: The numeric value of the textfield. Decimal point must be stated as . (point)
See also: NUMBER
Example: #1 = NUMS("aa111") + NUMS("222,22 test") + NUMS("333.33")
Field 1 becomes the sum of the numbers contained in the textfields = 555.33

3.9. **PACK** - Packing of a number

text PACK(text *par1*, number *par2*)

par2 : 0, not used, reserved for future packtype
Description: 8870 - basic call 60,A$,B$ is the same as B$=PACK(A$)
Returnvalue: The packed value of the field.
See also: UNPACK
Example: #1=PACK(#2) */* #1 becomes the packed value of #2*

3.10. SMAA - Convert text to upper/lower case - names

text SMAA(text *par1*)
Parameters: *par1* : the text to be converted
Description: The function converts the text in *par1* to upper and lower case letters. Eg. the first letter in each word will be set to upper case while the rest is set to lower case letters. The SSV file WORDS.ENG will be checked for occurence of the first and last word in the text. If found the spelling of this will be taken from here.
Note that the SMAA function may be used in DATAMASTER also for online conversion of name input fields.
Returnvalue: The converted text.
See also: CONV, LOWER, NAME, UPPER
Example:

```
#1 = SMAA("MICHAEL OLSEN") /* Gives "Michael Olsen"
#1 = SMAA("SORENCO GMBH")  /* Gives "Sorenco GmbH"
```

3.11. SOGE - Creation of a searchkey from an adress field

text SOGE(text *par1*, number *par2*)

par2 : The length of the resulting namepart.
Description: The streetname and streetnumber is isolated from the given adress field. These are then combined into a searchkey where the streetname is of fixed length *par2* followed by the street number. This field may be used for sorting or searching.
Returnvalue: The streetname length *par2* followed by 4 digit street number.
See also: LOWER, NAME, SMAA, UPPER
Example:

```
#1 = SOGE("MAIN STREET 3",12)     /* Gives "MAINSTREET_____3"
#1 = SOGE("27, Rue de Saute",8)   /* Gives "RuedeSau__27"
```

3.12. SPOFF - Remove leading and trailing blanks in text

text SPOFF(text *par1*, Bitflag *par2*)

Description: The function removes all leading and trailing blanks. Furthermore it reduces all blank positions to a maximum of only one blank character.
Returnvalue: The converted text.
See also: LEN
Example:

```
#1="   This     is    a text    "
#2=SPOFF(#1)       Field #2 then contains the value "This is a text".
```

3.13. **UNPACK** - Unpacking of a number

text UNPACK(text *par1*, number *par2*)

par2 : 0, not used, reserved for future packtype
Description: 8870 - basic call 61,A$,B$ is the same as B$=UNPACK(A$)
Returnvalue: The unpacked value of the field.
See also: PACK
Example: #1=UNPACK(#2) */* #1 becomes the unpacked value of #2*

3.14. UPPER - Convert text to upper case

text UPPER(text *par1*)
Parameters: *par1* : defines a text to be converted
Description: The function converts a text to upper case, eg. all letters a-z are converted to A-Z.
Returnvalue: The converted text.
See also: CONV, LOWER, SMAA
Example:

```
#1="This is a test"
#2=UPPER(#1)       /* Field #2 then contains the text "THIS IS A TEST"
```

3.15. USING - Editing of number

text USING(number *par1*, text *par2*)

par2 : USING mask for editing
Description: The USING function converts a number to a textfield. The USING mask determins the layout of the text.
The function may be called with the special BASIC syntax also as: textfield = number USING "mask"
Returnvalue: The edited textfield.
See also: EDIT
Example:

```
#1 = USING(-123,"&&&,&&")           /* Gives "001,23"
#1 = USING(123.45,"#####")          /* Gives "__123"
#1 = USING(1234.56,"###,###.##")    /* Gives "__1,234,56"
#1 = 123.45 USING "#####"           /* Gives "__123"
```

4. Checkdigit and validation

This section describes functions for checkdigit calculation and validation of text and numbers.

4.1. CCODE - Field checktext (DATAMASTER checkcodetext)

text CCODE(text *par1*, field *par2*)

par2 : Fieldnumber with check defined as "7", "#7", "va#7", "va07"
Description: The function reads the field definition from the Data Dictionary for the given field *par2* and finds the checkcodes defined for this. The text connected with the value given in *par1* is returned.
Returnvalue: The checktext. Blank indicates not allowed, "-" no check defined.
See also: VALID, VALCH
Example: #1 = CCODE(9,"va#7") */* Gives "Special"*

4.2. **CHECK** - OCR check

text CHECK(text *par1*)
Parameters: *par1* : is a number as customer number
Description: The function processes a number and returns a text containing an OCR checkvalue.
#47=CHECK (#19) will calculate the OCR checkdigit modulus 10 with weights 212121... for the textfield #19 and ads this as the last digit.
CHECK("123456789012345") returns a text with one character added: "1234567890123452".
Returnvalue: The text plus the OCR checkdigit.
See also: CHEX
Example: #1 = CHECK("33330556") */* *Gives "333305563"*

4.3. CHEX - Modula 11 check

text CHEX(text *par1*, text *par1*)

par2 : Weigths for calculating the checkdigit, 2 digits for each input character
Description: #47=CHEX (#15,"01020304") will as CHECK calculate a checkdigit and add this on the return field.
The checkdigit is calculated using modulus 11 with the weights 01, 02, 03, 04 according to the second parameter. Each set of 2-digits in this parameter gives the weight for one digit in the parameter 1 field.
Returnvalue: The text plus the checkdigit.
See also: CHECK
Example: #2=CHEX("330556", "010203040506") */* *Gives "3305569"*

4.4. VALCH - Check if text found in the validation range

number VALCH(text *par1*, text *par2*)

par2 : the allowed values separated with comma
Description: The function validates *par1* found among the values given in *par2*. All values given in *par2* has to be separated with , (comma).
Returnvalue: Returns 0 if *par1* not found in *par2*.
See also: CCODE, VALID
Example: #1=VALCH("Chris", "Anne,Nette,Chris,Ole,Michael") */* #1 then contains the value 2.*

4.5. VALID - Check if number found in the valid numbers

number VALID(number *par1*, number *par2*, number *par3*)

. **Description:** The function validates if the value in *par1* is allowed by checking the allowed values in *par2*. The syntax for *par2* is:
"1,2,8-10,12" It is the values 1, 2, 8 to 10 and 12 are allowed.
"-1,2,8-10,12" If a minus is the first character the values are NOT allowed.

```
#20="1-3,8-12"
VALID(15,#20,1)
```

will change the value of the range field #20 by inserting 15 so #20 becomes: "1-3,8-12,15"
Returnvalue: Returns 0 if *par1* not found in *par2*.
See also: CCODE, VALCH
Example: #1 = VALID(9, "1,2,8-10,12")
Field #1 then contains the value 3 as the value is found inside the third range.

5. Date manipulation functions

Date calculation is a sience for itself and is descriped in this chapter.

5.1. **DATE** - Current date YYYYMMDD

number DATE()
Returnvalue: The current date as YYYYMMDD.

5.2. DATECALC - Calculate a date

Date DATECALC(Date *par1*, number *par2*, number *par3*, number *par4*, number *par5*)

par5 : day(s) DD

Description: The function can is used to at set a date, or add to/subtract from a date. If *par2* is set to 0 a date can be set using the parameters *par3-par5*. If parameter 3, 4 and 5 are set, the parameter 1 will be ignored. To set the month only, the function uses the the date in *par1* and changes the month to the one in *par4*.

Returnvalue: The calculated date as YYYYMMDD.

See also: DAY, FNA, FNB, FND, FNU, FNV, FNY,MONTH, WDAY, WORKD

Example:

```
#1=DATECALC(0, 0, 1997, 10, 16)   /* set the date 16.october 1997 (19971016)
#1=DATECALC(19970101, 1, 0, 2, 0) /* add 2 months to the date (19970301)
#1=DATECALC(19971016, 2, 1, 2, 3) /* subtract 1 year, 2 months and
                                     3 days from the date (19960813)
```

5.3. DAY - Description of a date - textform

text DAY(Date *par1*)
Parameters: *par1* : a date as YYYYMMDD
Description: The function creates a text with the date as: <?> <weekday> the. <day> <month> <year>
If the day is a 'free-day' the first character will be a *, if only a 'half free-day' a /, otherwise blank. The same calender as descriped for WORKD is used.
Returnvalue: Returns a text.
See also: DATECALC, FNA, FNB, FND, FNU, FNV, MONTH, WDAY, WORKD
Example: #1 = DAY(19931016) */* *create text for 16. october 1993*
Field #1 contains the value "*Saturday The 16 october 1993"

5.4. FNA - Convert date to number of days from year 0

number FNA(Date *par1*, number *par2*)

Description: The function calculates the given date to the number of days since the year 0. This value can be used to add or subtract days or to calculate the difference between dates.
Returnvalue: The number of days since the year 0.
See also: FNB, FND, FNU, FNV, DATECALC, DAY, MONTH, WDAY, WORKD
Example:

```
#1 = 19931215           /* the date 15. december 1993
#2 = FNA(#1)            /* how many days since 0 ?
#3 = #2 - FNA(19931202) /* how many days since 2. december ?
```

Field #2 contains the value 728277 and field #3 the value 13

5.5. FNB - Convert number of days from year 0 to date

Date FNB(number *par1*, number *par2*)

Description: The function calculates a date YYYYMMDD on basis of a value. Eg. a number returned from the function FNA() can be parsed as parameter to this function and hereby return a valid date.
Returnvalue: Returns the value as a date YYYYMMDD.
See also: DATECALC, DAY, FNA, FND, FNU, FNV, MONTH, WDAY, WORKD
Example:

```
#1 = FNA(19931215) /* convert the date 15. december 1993
#2 = FNB(#1 + 9)   /* add 9 days and convert to date YYYYMMDD
```

Field #2 contains the value 19931224, eg. 24. december 1993

5.6. FND - Date conversion

Date FND(Date *par1*)
Parameters: *par1* : defines a date as YYYYMMDD
Description: This function may be used to convert dates from one format to another, and is normally used with sorting and selections. Ex.

970101 is greater than 961231 but 311296 is greater than 010197

You can see the need for using the FND function if you try similar comparisons with a datefield defined DDMMYY.
Returnvalue: Returns the value as a date YYMMDD or DDMMYY.
See also: DATECALC, DAY, FNA, FNO, FNU, FNV, FNY, MONTH, WDAY, WORKD
Example:

```
#1 = FND(310395)        /* Gives 950331
#1 = FND(950331)        /* Gives 310395
#1 = FND(19950331)      /* Gives 310395
```

5.7. FNE - Convert date to month number

number FNE(Date *par1*)
Parameters: *par1* : A date as YYYYMMDD or YYMMDD
Description: This function may be used to calculate date differences in months.
Returnvalue: The function calculates the month number as Year*12 + Month (YY*12+MM)
See also: DATECALC, DAY, FNA, FNB, FND, FNV, MONTH, WDAY, WORKD
Example: #1 = FNE(19950331) */* *gives 1143 = 95*12+03*

5.8. FNF - Convert date to daynumber, 360 days/year

number FNF(Date *par1*)
Parameters: *par1* : A date as YYYYMMDD or YYMMDD
Description: This function calculates the daynumber from year 0 using 360 days/year. Same as FNA(date,360)
Returnvalue: Number of days from year 0.
See also: FNA
Example:

```
#1 = FNF(19950331) /* gives 1718290
#1 = FNF(950331)   /* gives 34290
```

5.9. **FNO** - Convert date to DDMMYY

Date FNO(Date *par1*)
Parameters: *par1* : Date given as DDMMYY, YYMMDD or YYYYMMDD
Description: Nomatter how the input date is turned the date will be returned as DDMMYY. This can then be used in subsequent printouts.
Returnvalue: DDMMYY
See also: FND, FNY
Example:

```
#1 = FNY(310395)        /* Returns 310395
#1 = FNY(950331)        /* Returns 310395
#1 = FNY(19950331)      /* Returns 310395
```

5.10. FNU - Convert date to weekday

number FNU(Date *par1*)
Parameters: *par1* : defines a date as YYYYMMDD
Description: The function is used to at calculate the weekday of a date.

See also: DATECALC, DAY, FNA, FNB, FND, FNV, MONTH, WDAY, WORKD
Example: #1 = FNU(19931215) */* which day is 15. december 1993 ?*
Field #1 contains the value 4 (=Wednesday)

5.11. FNV - Convert date to weekno or weekno to date

number FNV(number *par1*)
Parameters: *par1* : defines a date as YYYYMMDD, or a weeknumber as YYYYWW
Description: The function converts a date to a weeknumber YYYYWW, if *par1* is a date. If *par1* on the other hand is a weeknumber YYYYWW the function will return a date equal to the last sunday before the given week. Same as WEEK(date)
Returnvalue: Returns a number YYYYWW, where YYYY = year and WW = ugenr, or a date YYYYMMDD.
See also: DATECALC, DAY, FNA, FNB, FND, FNU, MONTH, WDAY, WEEK, WORKD
Example:

```
#1 = FNV(19931016) /* calculate weeknumber of the date 16. oktober 1993
#2 = FNV(#1)       /* calculate the last sunday before weeknumber 41
```

Field #1 then contains the value 199341, equal to weeknumber 41. Field #2 contains the date 19931010.

5.12. **FNY** - Convert date to YYYYMMDD

Date FNY(Date *par1*)
Parameters: *par1* : Date given as DDMMYY, YYMMDD or YYYYMMDD
Description: Nomatter how the input date is turned the date will be returned as YYYYMMDD. This can then be used in subsequent calculations.
Returnvalue: YYYYMMDD
See also: FND, FNO
Example:

```
#1 = FNY(310395)        /* Returns 19950331
#1 = FNY(950331)        /* Returns 19950331
#1 = FNY(19950331)      /* Returns 19950331
```

5.13. MONTH - Generate text describing a month

text MONTH(Date *par1*)
Parameters: *par1* : defines a date as YYYYMMDD
Description: The function generates a text equal to the name of the requested month.
Returnvalue: Returns the name of the month.
See also: DATECALC, DAY, FNA, FNB, FND, FNU, FNV, WDAY, WORKD
Example: #1 = MONTH(19931016) */* date is 16. oktober 1993*
Field #1 then contains the value "october".

5.14. **TIME** - Current time TTMMSS

number TIME()
Returnvalue: The current time as TTMMSS.

5.15. **WDAY** - Describe weekday of date

text WDAY(Date *par1*)
Parameters: *par1* : defines a date as YYYYMMDD
Description: The function generates a text as: <?> weekday
If the day is a free-day the first character will be a * and is it a / if only a half free-day. Otherwise blank. The same calender as descriped for WORKD is used.
Returnvalue: A text with the day.
See also: DATECALC, FNA, FNB, FND, FNU, FNV, MONTH, WDAY, WORKD
Example: #1 = WDAY(19931016) */* *Field #1 contains the value "*Saturday"*

5.16. WEEK - Convert date to weekno or weekno to date

number WEEK(number *par1*)
Parameters: *par1* : defines a date as YYYYMMDD, or a weeknumber as YYYYWW
Description: The function converts a date to a weeknumber YYYYWW, if *par1* is a date. If *par1* on the other hand is a weeknumber YYYYWW the function will return a date equal to the last sunday before the given week. Same as FNV(date)
Returnvalue: Returns a number YYYYWW, where YYYY = year and WW = ugenr, or a date YYYYMMDD.
See also: FNV
Example:

```
#1 = WEEK(19931016) /* calculate weeknumber of the date 16. oktober 1993
#2 = WEEK(#1)       /* calculate the last sunday before weeknumber 41
```

Field #1 then contains the value 199341, equal to weeknumber 41. Field #2 contains the date 19931010.

5.17. WORKD - Calculate number of workdays between dates

number WORKD(Date *par1*, Date *par2*)

par2 : defines a date as YYYYMMDD
Description: The function calculates the number of workdays between two dates.
#47 = WORKD (#15,#PD) calculates the number of actual workdays from the date in field 15 to the date entered in 'As of date'.
The function starts by calculating the number of days between the two dates. All Saturdays and Sundays will then be subtracted. As the final step the functions searches a 'workday tablefile', where holidays are listed, and then subtracts a full or half day per day found.
This tablefile can if necessary be adjusted individually. The function uses the file RAPDAY.ENG. This file is a SSV tekstfile where each line contains a holyday as YYYYMMDD. For half holidays follows the percentage of freedom as eg. 19960630;50
Returnvalue: Returns the number of workdays between to dates.
See also: DATECALC, FNA, FNB, FND, FNU, FNV, MONTH, WDAY, WORKD
Example: #1 = WORKD(19930420, 19930430) */* Field #1 then contains the value 19.*

6. Handling of multiple fields

This chapter decripes functions for handling a bunch of fields, especially the LET function.

6.1. LET - Calculating several fields at a time

number LET(fields *par1*)
Parameters: *par1* : defines one or more fields
Description: The function is used to at calculate one or more fields using one statement. The fields can calculated with the expression fields **XX** constant/field, where **XX** may be

Operator	Meaning
=	set fields equal to
+=	add value to the fields
-=	subtract value from the fields
*=	multiply fields with the value
/=	divide fields with the value
%=	set fields to the mod. value from the division
&=	perform logical and operation on fields
|=	perform logical or operation on fields

Returnvalue: Returns 0 if the calculation was successful.
See also: CLEAR, ZERO
Example:

Letexpression	Function
LET("#1-10=12")	Field 1 to 10 is set equal to 12
LET("#20,25=3,7")	#20=3 and #25=7
LET("#20-25=le#1-10")	Field 20-25 is set to the file le field 1-6
LET("#20-25=le#1-2")	#20=#22=#24=le#1, #21=#23=#25=le#2
LET("le#1,3,va#7=#1,ku#3")	Several files may be mixed
LET("#20-25+=1")	Add 1 to all the fields 20-25

6.1.1. LET - Assign values to fields between IQ programs (IQ)

number LET(fields *par1*)
Parameters: *par1* : defines one or more fields
Description: The LET assignment of multiple fields has been extended to work with multiprograms and between lines in list/transaction programs.
Returnvalue: Returns 0 if the calculation was successful.
See also:
Example:

Letexpression	**Function**
LET (20.#1-3=#1-3)	Sets field 1-3 for program 20 = this program #1-3
LET (#1-3=20.#4-6)	Sets field 1-3 in this program to #4-6 from program 20
LET (#10=#3.4)	Sets field 10 equal to field 3 from line 4

6.1.2. LET - Creating new files (RAP)

number LET(fields *par1*)
Parameters: *par1* : defines one or more fields
Description: The LET function may be used to build new files.
Returnvalue: Returns 0 if the calculation was successful.
See also: INSERT, UPDATE, *Rapgen Manual*
Example:

Letexpression	**Function**
LET (aa=#1-3,87,le#2)	Define file aa, key=aa#1, type=1.database driver
LET (aa=#1-3,6K,15D)	Keys aa#4 and aa#5 (duplicates)
LET (aa=#1-3,6,15:2,NP)	Keys aa#2 and rel.recno (duplicates)
LET (aa=#1-3),12000	12000 records (default is 1000 if needed)
LET (aa=#1-3),-1	File should be builded eachtime
LET (aa=#1-3),1000,xnet	File is a XNET file
LET (aa=#1-3) -acc	File is an access file, build always
LET (07/aa=#1-3),25	Lu may be given for basic files

6.2. CLEAR - Clear all fields in a file (RAP)

number CLEAR(file *par1*)
Parameters: *par1* : the shortname of the file
Description: The function sets all fields for a file to zero.
Returnvalue: Returns 0 if ok.
See also: ZERO
Example:

```
UPDATE(1)       /* the report updates the file
CLEAR(VA)       /* zero all fields from article file
VA#1 = "1234"   /* article number
INSERT(VA)      /* insert new record in article file
```

The example inserts a new record in the article file. Due to the function CLEAR() all other fields than the article number are set to zero.

6.3. CLRFLAG - Clear options for fields on screen (IQ)

`CLRFLAG(fields par1, number par2, number par3)`

Description: Each screenfield is associated with parameters (bits) defining the use. The SETFLAG function may be used to set these flags, CLRFLAG to clear them.
See also: SETFLAG, GETFLAG
Example: CLRFLAG("#12,44",7,0)

6.4. COLOR - Set background box color for a number of fields

COLOR(fields *par1*, ColorRed *par2*, ColorGreen *par3*, ColorBlue *par4*)

par4 : Blue colorvalue (0-255)
Description: The background color for the given fields is set to the RGB value, it is the field box is filled with the given color.
Returnvalue: None
See also: COLORF
Example:

```
COLOR("#3-4",255,0,0)      /* Field 3 and 4 becomes a red box around
COLOR("#3-4",-1)           /* No background color for the fields
```

6.5. COLORF - Set forground text color for a number of fields

COLORF(fields *par1*, ColorRed *par2*, ColorGreen *par3*, ColorBlue *par4*)

par4 : Blue colorvalue (0-255)
Description: The forground color for the given fields is set to the RGB value, it is the field text is printed in the given color.
Returnvalue: None
See also: COLOR
Example: COLORF("#3-4",0,0,255) */* Field 3 and 4 are printed in blue*

6.6. DIALOG - Function for additional input

Number DIALOG(Fields *par1*)
Parameters: *Par1*: Fields to show in the dialog
Description: The DIALOG function enables the user to pop up dialogboxes with a selected set of fields at any point of a report execution or in an IQ program for example by click on a field. DIALOG("#1,7-8,le#3") defines a dialog with the given fields. The fields documentation is used as floating online help when the mouse cursor is moved over the leading text.
Together with a field you may state one of the following additional options:

```
Lxxxx   Line     (dialog units)
Pxxxx   Position (dialog units)
Hxxxx   Height    (dialog units)
Wxxxx   Width    (dialog units)
N       No leading text
N1      Add fieldnumber to leading text
N2      Display leadingtext above field instead of left of field
C       COMBOBOX, Field check definitions shown as values
O       LISTBOX, Field check definitions shown as values
:xx     Skip to next column fieldline xx
+xx     Skip xx fieldlines down
```

.
Returnvalue: OK=0, CANCEL=1
See also: PARAMS
Example:

```
DIALOG("#1-3,11")             /* Make a dialog with the given fields
```

6.7. GETFLAG- Get options for fields on screen (IQ)

number GETFLAG(fields *par1*, number *par2*, number *par3*)

Description: Each screenfield is associated with parameters (bits) defining the use. The SETFLAG function may be used to set these flags, CLRFLAG to clear them. The GETFLAG function may be used to read these flags.
Returnvalue: None
See also: SETFLAG, CLRFLAG
Example: GETFLAG("#12,44",7,0)

6.8. SETFLAG- Set options for fields on screen (IQ)

SETFLAG(fields *par1*, Bitflag *par2*, number *par3*)

Description: Each screenfield is associated with parameters (bits) defining the use. The SETFLAG function may be used to set these flags, CLRFLAG to clear them.
For the type parameter 0 only should normally be used.
Returnvalue: None
See also: GETFLAG, CLRFLAG
Example: SETFLAG("#12,44",7,0)

6.9. **ZERO** - Zero a number of fields

ZERO(fields *par1*)
Parameters: *par1* : Field specification
Description: The given fields are zeroed. ZERO is working just like the LET function.
Returnvalue: None
See also: LET, CLEAR
Example: ZERO("3,19") */* Zeroes field 3 and field 19*

7. Report control

The chapter describes functions to control the flow of report calculations/print in RAPGEN. The functions CHAIN, MESS and RETURN may also be used in IQ and DATAMASTER, the other functions is of no interest for screenprograms.

7.1. CHAIN - Start next report or another program (RAP)

number CHAIN()

par3 : Blank or Index,Totallevel,Companynumber
Description: CHAIN(7) starts report number 7 when this report is finished. The same start parameters as for this report will be used.
CHAIN(7,",310395,-,9999","1") sets Asofdate to 310395, Startkey to nothing, Stopkey to 9999 and lowest total level to 1. The other startparameters remains unchanged.
CHAIN(2007) starts report number 7 in subsystem 2.
CHAIN(-1,"c:/windows/write.exe") will start this (windows)program.
Each time CHAIN is executed a new runnumber is given starting from 1 and onwards. A report is started from the menu has runnumber 0. You can use #20=CHAIN() without parameters for CHAIN to get this runnumber and make a report run a number of times, eg. to print a number of copies.
CHAIN("c:/windows/write.exe") may be used in IQ/DATAMASTER programs to start another windows program.
Returnvalue: CHAIN() returns the actual runnumber.
See also: EXIT , CHAINR
Example:

```
#20=CHAIN()          /* This is report number 7.
IF #20<3 CHAIN(7)    /* The same report will be started 4 times.
```

7.1.1. CHAINR - Chain program or external command directly (RAP)

CHAINR(number *par1*, text *par2*, text *par3*)

par3 : Blank or Index,Totallevel,Companynumber
Description: The CHAIN command will always be placed LAST it is the next program will be started after this is finished.
Use CHAINR instead of CHAIN to interrupt this program and call-up another program immediately.
Returnvalue: None
See also: EXIT , CHAIN
Example: CHAINR(-1,"Notepad") */* Start notepad right now*

7.1.2. CHAIN - Chain IQ program or external command (IQ)

CHAIN(text *par1*, text *par2*)

par2 : Optional key for record to display
Description: Activate a program number or a windows command string.
Returnvalue: None
See also: EXIT, ISACTIVE, WAIT
Example:

```
CHAIN ("20")    starts program 20.
CHAIN ("+5")    starts program 5 and activates this.
CHAIN (">5")    starts program 5, the current record will not be transmitted
CHAIN ("$5")    starts program 5, activates it and waits until this finishes.
CHAIN ("+5",#1) starts program 5 which will read a record using #1

#20="notepad"
#20="command.com /C edit myfile.txt"
CHAIN(#20)        starts the specified windows program
CHAIN("rapwin &") & as last character lets IQ continue
                    while the newstarted program is running.
```

7.2. **WAIT** - Wait for program to finish (IQ)

WAIT(programno *par1*)
Parameters: *par1* : Programnumber
Description: Wait for given program to finish (see EXIT). Calculations will continue when the program window is closed.
Returnvalue: None.
See also: CHAIN , EXIT
Example: WAIT(20) */* Do not continue before program ready*

7.3. COMPILE - Compile a report (RAP)

COMPILE(number *par1*)

Prerequirements: It is only possible to use this function if a C compiler is installed and RAPGEN is bougth with licens for compiling.
Description: Instead of selecting 'Compile' from the 'Parameter' menu whenever the report is started after amendments this can be fixed in the calculations.
See also: INSTALL
Example: COMPILE *\/* The report will be compiled*

7.4. EXIT - Exit the report (RAP)

number EXIT(number *par1*)

Description: The function terminates the report or the current pass (sort/print).
Returnvalue: None
See also: CHAIN , CHAINR , MESS
Example:

```
READ(le) /* Read supplier data
IF #OK THEN BEGIN /* terminate the report if supplier not found
#12="Supplier ", le#1, " not found:"
MESS(#12)
EXIT(0)
END
```

7.4.1. EXIT - Close IQ program or window (IQ)

EXIT(number *par1*)
Parameters: *par1* : Program number to close
Description: EXIT(0) closes the current IQ program.
Returnvalue: None
See also: CHAIN , MESS, WAIT
Example:

```
EXIT(20) closes program 20 if this is open, 1020 gives subsystem 1.
EXIT(-1) closes the program selection window.
EXIT(-2) closes the field   selection window.
EXIT(-3) closes and exits all IQ.
```

7.5. KEYS - External start/stop ranges (RAP)

number KEYS()

par2 : Eventual fixed name for .KEY definition file
Description: Using the KEYS function you can make a report to run with a number of start/stop ranges defined as lines in an external textfile. KEYS then replaces the entering of START/STOP keys by start of the report and may also replace the INDEX specification.
The keysfile can be created with any texteditor and may contain lines like:

```
0001
1000-1999
0005-0099,0200,0155-0157
2:205-271
47/2000-2500
```

Each line can contain single keys or key ranges for print. 2: specifies use of index 2, 47/ states a calculation code which you can read in the calculations with #20=KEYS() and use for individual calculations.
Use of KEYS(0) produces one list containing all records specified in the keysfile, KEYS(1) produces one seperate list for each line in the keysfile and the ENDSUM routine may be used to get a total of these reports.
You may control a report with a keysfile also without placing a KEYS calculation. By start of any list, in START FROM, you may enter:

```
(aa)                          Start with keysfile aa
(1000,1100-1200,0004          Run over these key ranges
```

If path/extension is omitted for the keysfile this will be taken from the normal reportdirectory with the extension .KEY, eg. c:/rapfil/rap/aa.key
Returnvalue: KEYS() returns the calculation code (47 of 47/111-222) for the current range.
See also: ENDSUM, INDEX
Example:

```
KEYS(0,"c:/mydir/enfil.min")  /* The report is controlled from this file.
#20=KEYS()                    /* A calculation code is read.
```

7.6. INDEX - Set index and start/stop value for report (RAP)

number INDEX(index *par1*, text *par2*, text *par3*)

par3 : value that the user normally enteres in the field Stop at
Description: The function is used to enforce an index and start/stop range for a report. If *par1* >= 1 the index is set for the mainfile, eg. in which order the report must read the records. If *par2* contains something the function will set the start range and which applies also for *par3*.
If the start/stop parameters have the first character as plus (+) the value is placed in front of any input done in the start/stop fields by start of the report.
INDEX(-2) locks the report to use index 2 but in descending order. The database driver must support descending read.
Returnvalue: Returns the index the mainfile will use.
See also: KEYS
Example: INDEX(2,"D","D") */* the reports mainfile is KU (currency file)*
The example enforces index 2 for the report, so that the currency's are sorted accoreding to the currency name and not the currency code. Furthermore it only reads the records, where the currency name start with the letter "D".
INDEX(1,"+02","+02") */* Print 024711 upon entering 4711*

7.7. LTOT - Lowest total level (RAP)

number LTOT(level *par1*)
Parameters: *par1* : the lowest total level requested for the report
Description: If *par1* >= 0 the function sets the lowest total level for the report. This level equals the one the user normally selects when starting a report.
Returnvalue: Returns the reports lowest total level.
See also: MTOT
Example: LTOT(1) */* print totals only, suppress all specification*

7.8. MTOT - Highest total level (RAP)

number MTOT(level *par1*)
Parameters: *par1* : the highest total level for the report
Description: The function enforces the highest total level for the report. If *par1* equals 0, the report will not print any totals.
Returnvalue: Returns det highest total level.
See also: LTOT
Example: MTOT(1) */* A non-relevant grande total is being suppressed*

7.9. MESS - Display message on screen

number MESS(text *par1*)
Parameters: *par1* : the message to be displayed
Description: MESS displayes the text in a Windows messagebox. Dependent on the last character in the text the following symbol and buttons are used:

text	Symbol	Buttons	Defaultbutton
text	Info	OK	OK
text?	!	OK, CANCEL	CANCEL
text??	?	YES, NO, CANCEL	YES
text!	!	YES, NO	YES
text!!	STOP	OK	OK
text?!	STOP	OK, CAN	OK

Returnvalue: 0=OK or YES, 1=NO, -1=CANCEL
See also: EXIT
Example:

```
#1=MESS("Stop the report !")
IF #1=0 EXIT(0)               /* exit the report
```

7.10. NOPAS - No password/username on this report (RAP)

NOPAS()
Parameters: None
Description: The function removes any password protection from the report. Normally an updating report automatically gets the password CARE. Using NOPAS() or PAS() this password may be removed or changed.
See also: PAS, UPDATE
Example:

```
UPDATE(1)
NOPAS()              /* no password on this report
```

7.11. **PAS** - Set password/username (RAP)

number PAS(text *par1*)
Parameters: *par1* : the requested password/username
Description: The function enforces a password/username for a report. This password is then required in order to start the report.
See also: NOPAS
Example: PAS("SWTOOLS") */* set password to SWTOOLS*

7.12. PARAMS - Function for additional report start parameters (RAP)

PARAMS(Fields *par1*)
Parameters: *Par1:* Fields to show in start parameter dialog
Description: PARAMS("#1,7-8,le#3") is a variant of the dialog function where the input is done by start of the report not during report execution.
Use of PARAMS in a report will add a button <Extra parameters> to the startup screen which then activates the dialog.
Returnvalue: None.
See also: DIALOG
Example:

```
PARAMS("#1-3,11")              /* Make a dialog with the given fields
```

7.13. RETURN - Return from calculations

number RETURN(number *par1*)
Parameters: *par1* : the value to be returned
Description: The function is used to exit from the calculations performed for the current main file record. If no parameter is given or *par1* equals 0, the report will print the defined print lines for the record. If the value is non-zero the record will not be processed any further or printed.
Returnvalue: None.
See also: GOSUB
Example: IF LE#6 < 1000 RETURN(1) */* no print if balance < 1000*

7.14. SORTKEY - Inserting additional sortkey (RAP)

number SORTKEY(fileid *par1*)
Parameters: *par1* : 0, -1 or fileid
Description: In some special cases a list should be sorted printing the same record multiple times on the output. For example an article list where the article is to be found with the normal supplier number and the alternative supplier number if any.
In such case you should sort using a workfield which then is calculated and an extra sortkey is released whenever the SORTKEY function is called.
Several files may also be merged using this function. The sortworkfile contains a number normally pointing to a record from the report mainfile. With SORTKEY(le) a record is inserted pointing to the file le and with #20=SORTKEY(-1) the filenumber of the file currently being the mainfile is returned which can be used to control futher calculations.
Returnvalue: Mainfilenumber, normally 1.
See also: MERGE
Example:

```
#11=#9                  /* Sortworkfield = Alternative supplier
IF #11<>0 SORTKEY(0)    /* Extra sortkey with this
#11=#6                  /* Normal sortkey with normal supplier
```

7.15. SORTWORK - Using a specific sortfile (RAP)

SORTWORK(number *par1*)
Parameters: *par1* : Sortworkfile number
Description: During sort RAPGEN creates the workfiles: c:/tmp/SIN00000.000 and c:/tmp/SUD00000.000 where c:/tmp/ is the normal TMP directory. These sortfiles are not deleted after use as you by start of the next report by entering

START AT: SORT or SORTD

can avoid the sorttime and use the same sorting as for last run. If you intend to use these function SORTWORK(47) can ensure that the sortfiles are not overwritten by other lists as the filenames then becomes: c:/tmp/SIN00000.047 and c:/tmp/SUD00000.047.
Returnvalue: None.
See also:
Example: SORTWORK(47)

7.16. **WHEN** - When to perform calculations (RAP)

WHEN(number *par1*, number *par2*)

Description: The command WHEN is used to define when calculations may be performed, i.e. before/after sorting or accumulating totals.

8. Printer control

This section describes functions for report printer control.

8.1. COPIES- Number of print copies (RAP)

COPIES(number *par1*, Printer *par2*)

par2 : optional printer number
Description: COPIES(1) gives one additional copy of the print output. A maximum of 30 copies can be stated and the must be room for all Windows spoolfiles.
COPIES(1,7) produces one additional copy on the printer defined as no.7 in the printer setup. Note however that unexpected pageshift will occur if the copyprinter has a smaller form than the original.
Returnvalue: None.
See also: PRINTER
Example: COPIES(1) */* Print 2 times*

8.2. PAGE - Change report layout page (RAP)

number PAGE(number *par1*)
Parameters: *par1* : the requested report page
Description: A report normally uses page 0 when printing. This is the page that you normally use when defining a layout. A report may use different layout pages eg. to allow supplier letters to be printed in different languages (max. 9 layout pages). These pages are numbered from 0 to 9 and can be reached from the 'file' menu, 'page layout' when editing the form.
Returnvalue: Returns the page current selected as active print page.
See also: PRINT
Example:

```
PAGE(le#5)   /* select print page according to the suppliers language
PRINT(1-10)  /* print text
```

8.3. PRINT - Print lines from report layout (RAP)

PRINT(text *par1*)
Parameters: *par1* : the lines to be printed
Description: The function is used to print lines from the report layout, or to set a print command that is performed for each page or print of total lines. The syntax is:

Function	Description
PRINT(1-10)	Print the lines 1 to 10
PRINT(1,+2,2)	Print line 1, then 2 blank lines and finally line 2
PRINT(1,:60,2)	Print line 1, goto line 60 and print line 2
PRINT(:1003,1,3)	Goto 3 lines before bottom and print line 1 and 3
PRINT(1-10,:1,20)	Print line 1 to 10, make form feed and print line 20
PRINT(*H)	The lines defined using H= is printed

The function is also used to set print commands controlling which lines to print in different situations:

Function	Description
PRINT(H=1-4)	When new page, print lines 1 to 4
PRINT(L=8)	The lines printed for each record in the main file is print line 8
PRINT(T=10)	The total line is line 10 (Applies also for grand total)
PRINT(D=9)	As heading for the Detaillines(READH) line 9 is printed
PRINT(B=:1002,17)	As Bottom on every page line 17 is printed
PRINT(N=3,:1,1-4)	Newpage 3 lines befor pageend, heading line 1-4
PRINT(A=10)	Line 10 is printed before a totalblock
PRINT(C=11)	Line 11 if printed after a totalblock

Note that a textfield may be used in the printcommand as

```
#11="1-4,15"
PRINT(#11)
```

PRINT(>2) switches to printer 2, see PRINTER.
Returnvalue: None.
See also: PAGE , PRINTER
Example: PRINT(:60,1-10) */* goto line 60 and print line 1 to 10*

8.3.1. **PRINT** - Print output control (RAP.)

PRINT(text *par1*)
Parameters: *par1* : option=value
Description: The PRINT command is expanded with the command syntax PRINT(xx=value yy), where xx,value and yy can be one of the following:

	Function	**Description**
xx=	ml	Left margin
	mr	Right margin
	mt	Top margin
	mb	Bottom margin
	eh	Empty line height
	ce	Close report windows on exit
	fh	Standard font height for all lines
	cd	Close printer document and start new
yy=	cm	Centimetre
	in	Inches
	pt	Points
	<none>	Device pixels

8.3.2. PRINT(?= - Printer characteristics inquiry (RAP.)

PRINT(?=text *par1*)

Description: The PRINT command is also expanded with a query function in order to receive some information from the internal print handler.
The return value yy is reported in pixels except when xx is 5, 8, 9, 15 or 16.

8.4. PRINT(LAB= - Label function (RAP)

PRINT(LAB=Text *par1*, Text *par2*, Text *par3*, Text *par4*, Text *par5*, Text *par6*)

par6 : Copies

Description: The width and height of any label on the sheet can be given in centimetres or inches by using the following syntax:

```
7cm equals 7 centimetres
2in equals 2 inches
```

The below sample produces labels printed from left to right on a label sheet with 21 labels, 3 on each row, 7 rows, where each label has the width/height of 7 centimetres. Each label is printed in 2 copies.

Returnvalue: None.

See also: PRINT

Example:

```
FIRST
PRINT(LAB=1,3,7,7cm,7cm,2) /* Define label print
NORMAL
```

8.5. **PRINTER**- Printer selection (RAP.)

PRINTER(Printer *par1*)
Parameters: *par1* : Printernumber
Description: This function is used in connection with the printer dialogue. In order to set the default printer for a report the following line can be added in the calculations:
Returnvalue: None.
See also: COPIES, PRINT
Example: PRINTER(7) */* default printer for this report is printer 7*

8.5.1. PRINTER - Multiple printer output (RAP)

PRINTER(number *par1*, Printer *par2*)
Parameters: *par1* : Printernumber *par2* : PrinterID
Description: PRINTER(2,7) will open a secondary printer defined as printer number 7 in the printer setup. No output is printed on this until a

PRINT(>2)

is found in the calculations whereafter all print goes to this printer. PRINT(>1) switches back to the default printer.
Each printer has its own pagenumbers and may differ in paper size. A maximum of 30 concurrent printers or copies can be used.
Returnvalue: None.
See also: COPIES, PRINT
Example: PRINTER(2,7) */* Open secondary printer 7*

8.6. PRTTOTAL - Manual control of total printout (RAP)

PRTTOTAL(Level *par1*)
Parameters: *par1* : Total level number
Description: RAPGEN normally produces a subtotal when a part of the sortkey changes value. With the use of PRTTOTAL you can manually control all print of subtotals and instead print these when a field changes value.
Returnvalue: None.
See also: ENDSUM
Example:

```
IF #7=1 PRTTOTAL(1)             /* Print subtotal if field 7 is 1
LAST
PRTTOTAL(2)                     /* Print grandetotal last
```

8.7. SCRPRT - Recall screen print (IQ)

SCRPRT(Filename *par1*)
Parameters: *Par1*: Filename to show using the screen printer
Description: SCRPRT("filename") calls up the screen printer with the saved print from filename. This may for example be used in IQ by click on a field.
Returnvalue: None.
See also: PRINT
Example:

```
SCRPRT("c:/w/ab.cde")     /* Show this file using the screen printer
```

9. Reading files

This chapter describes the READ function for reading one record from a secondary file and the START/NEXT/REPEAT functions for loop over a range of records.
The principles of handling of more files and there connections are decriped in RAPGEN Usermanual, section Using multiple files.

9.1. READ - Read record from file

number READ(file *par1*, index *par2*) ,connection *par3*

par3 : Optional connection if standard connection not present or suitable
Description: The function reads a record from a file.
READ(le) reads the file le using the standard connection defined in the Data Dictionary.
READ(le),#9 reads the file le using field 9 as key for index 1, nomatter if and how a standard connection is defined.
READ(va.02),#6 reads the file va using field 6 as key for index 2, nomatter if and how a standard connection is defined.
READ(le),"1",#9(3,4),#7 forms the key as a combination of the constant "1" and character 3-4 of field 9 followed by field 7.
READ(le.00),#6 reads the le file using the recordnumber (index 0) as given in field 6.
Returnvalue: 0 if record is read.
See also: START, NEXT, REPEAT, END , PRIOR, READR, READX
Example: READ(le) */* read the supplier*

9.2. READH - Read record with optional print of heading

number READH(file *par1*, index *par2*) ,connection *par3*

par3 : Optional connection if standard connection not present or suitable
Description: The function reads a record from a file just as READ. If another record is read than last time READH was used, eg. when the supplier number changes, the heading given for READH will be printed.
Returnvalue: 0 if record is read.
See also: READ
Example: READH(le) */* read the supplier with optional heading*

9.3. READR - Read record using recordnumber

number READR(file *par1*) ,connection *par2*

par2 : Optional connection if standard connection not present or suitable
Description: This function reads a record from a file using recordnumber as key. READR can be used only on database systems working with recordnumbers and are included only for compability with previus releases.
READ(le.00),#6 is the same as READR(le),#6
See also: READ , READX

9.4. READX - Read record using relative recordnumber

number READX(file *par1*) ,connection *par2*

par2 : Optional connection if standard connection not present or suitable
Description: This function reads a record from a file using relative recordnumber as key. READX can be used only on database systems working with recordnumbers and are included only for compability with previus releases.
READ(le.00),#6+N is the same as READX(le),#6
See also: READ , READR

9.5. START - Set index and range for a file

number START(file *par1*, index *par2*) ,connection *par3*

par3 : Optional connection if standard connection not present or suitable
Description: The function prepares reading with the NEXT function by setting the range of keys for this.
The standard file connection can be used or the key may be specified just as decriped for READ.
By START you will normally just specify a part of the key. The subsequent reading with NEXT will retreive all records where the first part of the record key matches with the keypart given in START.
Returnvalue: Returns 0 if range ok.
See also: READ, NEXT, REPEAT, END , PRIOR
Example:

```
#47=0                    /* Zero total field
START(va)                /* Start reading of articles
NEXT(va)                 /* Read next article
#47=#47+va#3             /* Totalize all articles
REPEAT(va)               /* Continue until end of range
```

9.6. NEXT - Get next record in range

number NEXT(file *par1*)
Parameters: *par1* : shortname of the file
Description: The function is used in connection with START/NEXT/REPEAT loops. The functions START() and END() set the requested range for the loop. NEXT() then reads one record from the file. When the calculation REPEAT() is performed the function NEXT() will be performed once again until no more records exists in the given range.
Returnvalue: Returns 0 as long as records exists in the range.
See also: READ, START, REPEAT, END , PRIOR
Example:

```
PRINT                    /* Take over complete print control
PRINT(4,6,5)             /* Print supplier heading
START(va)                /* Start reading of articles
NEXT(va)                 /* Read next article
PRINT(7)                 /* Print all articles
REPEAT(va)               /* Continue until end of range
```

9.7. REPEAT - Repeat reading NEXT

number REPEAT(file *par1*)
Parameters: *par1* : shortname of the file
Description: The function is used in connection with START/NEXT/REPEAT loops. The functions START() and END() set the requested range for the loop. NEXT() then reads one record from the file. When the calculation REPEAT() is performed the function NEXT() will be performed once again until no more records exists in the given range.
Returnvalue: None
See also: START, NEXT , PRIOR
Example:

```
#47=0                    /* Zero total field
START(va)                /* Start reading of articles
NEXT(va)                 /* Read next article
#47=#47+va#3             /* Totalize all articles
REPEAT(va)               /* Continue until end of range
```

9.8. GETKEY - Get current key value

text GETKEY(fileid *par1*)
Parameters: *par1* : Fileid
Description: #20=GETKEY(va) returns the index key for the last read record in the file va. The function is designed especially for database systems where the key not nessesary has to be stored as a field in the data record.
Returnvalue: The key value as text.
See also:
Example: #20 = GETKEY(va)

9.9. **END** - Set end range for a file after START

number END(file *par1*) ,connection *par2*

par2 : end range specification
Description: The START function defines the start key and the end key equal as the first part of the complete key. For example all postings with matching debitornumber is read.
Normally you do not have to use END, this is nessesary only if you need a special end range.
Returnvalue: Returns 0 if range ok.
See also: READ, START, REPEAT, NEXT , PRIOR
Example:

```
UPDATE(1)                  /* Empty workfile before use
START(xx),"0000"           /* Start reading from the very first
END(xx),"9999"             /* And go until the last one
NEXT(xx)                   /* Read next record
DELETE(xx)                 /* Delete all records
REPEAT(xx)                 /* Continue until file is empty
```

9.10. PRIOR - Get previus record in range

number PRIOR(file *par1*)
Parameters: *par1* : shortname of the file
Description: PRIOR works just like NEXT but the previus record is retreived. Note that not all database interfaces support reading records in 'reverse' order.
Returnvalue: Returns 0 as long as records exists in the range.
See also: READ, START, REPEAT, NEXT , END
Example:

```
PRINT                       /* Take over complete print control
#47=0                       /* Zero counter
START(va)                   /* Start reading of articles
PRIOR(va)                   /* Read prior article
#47=#47+1                   /* Count the articles
IF #47=1 PRINT(4,6,5)       /* Print supplier heading first time
PRINT(7)                    /* Print all articles in reverse order
REPEAT(va)                  /* Continue until end of range
IF #47>0 PRINT(7)           /* Print trailer if any articles
```

9.11. SPEED- Optimizing read strategi

SPEED()
Parameters: none
Description: The SPEED() function may be used to optimize the read strategi on a report as a record will not be read again when the same key is given but taken from memory. You should be carefull with this on updating reports.
Returnvalue: None.
See also: READ
Example: SPEED() */* Optimize the report read*

10. Writing to files

This chapter describes the different ways of updating files. Use of these functions requires that the system is installed allowing update of files, that the used database has functions for this and that the user has write permission on the server.
Any program doing file update should be tested before use. It will be

the total responcibility of the user

that the update really has been tested and is working correctly.

10.1. UPDATE - Allow update of files

number UPDATE(number *par1*, fields *par2*)

par2 : Optional file/fields allowed to update.
Description: UPDATE(1) must be placed in an updating report before any of the write functions are used in order to activate these.
The update command has been extended with specification of fields to update.

```
UPDATE(1,"va#6")    /* causes the program to update field 6 in va only.
UPDATE(1,"le#3-4") /* when more files are involved each file must be separate
UPDATE(0)           /* can now be used in DATAMASTER to switch all update off
```

Returnvalue: None.
See also: DELETE, INSERT, REWRITE, WRITE, NOPAS
Example:

```
UPDATE(1)               /* the report updates
NOPAS()                 /* no password
#6=#6+10                /* Do the field modifications
REWRITE(le)             /* update the supplier in the file
```

10.2. REWRITE - Rewrite record in file

number REWRITE(file *par1*)
Parameters: *par1* : shortname of the file
Description: The function updates a record in the given file which must have been read. Indexfields can be mofifyed only if the used database system supports this. The calculation UPDATE(1) must have been executed to activate this function.
Returnvalue: 0 if the record has been updated.
See also: DELETE, INSERT, WRITE, NOPAS, UPDATE
Example:

```
UPDATE(1)               /* the report updates
NOPAS()                 /* no password
AFTER                   /* JUST AFTER SELECTIONS DONE
#6=#6+10                /* Do the field modifications
REWRITE(le)             /* update the supplier in the file
```

10.3. INSERT - Insert new record in file

number INSERT(file *par1*)
Parameters: *par1* : shortname of the file
Description: The function inserts a new record in a file. ALL fields in the file must be assigned a value prior to INSERT. The calculation UPDATE(1) must have been executed to activate this function.
Returnvalue: 0 if record is inserted.
See also: DELETE, REWRITE, WRITE, NOPAS, UPDATE, CLEAR, LET
Example:

```
UPDATE(1)               /* the report updates
NOPAS()                 /* no password
CLEAR(le)               /* zero all fields in supplier record
LET("le#1,3=#7,17")     /* fill the fields
INSERT(le)              /* insert new supplier in supplier file
```

10.4. DELETE - Delete a record in a file

number DELETE(file *par1*)
Parameters: *par1* : the shortname of the file
Description: The function deletes a record in the requested file. The record must have been read before DELETE can be done. The calculation UPDATE(1) must have been executed to activate this function.
Returnvalue: 0 if record is deleted.
See also: INSERT, REWRITE, WRITE, NOPAS, UPDATE
Example:

```
UPDATE(1)               /* the report updates
NOPAS()                 /* no password
AFTER                   /* JUST AFTER SELECTIONS DONE
DELETE(va)              /* the selected articles are removed
```

10.5. WRITE - Write a record to file

number WRITE(file *par1*)
Parameters: *par1* : shortname of the file
Description: The function updates or inserts a record in det given file. If the last READ on this file has found a record the function issues a REWRITE, if no record was found by READ an INSERT is used. The calculation UPDATE(1) must have been executed to activate this function.
Returnvalue: 0 if record is updated/inserted.
See also: INSERT, REWRITE, DELETE, NOPAS, UPDATE
Example:

```
UPDATE(1)                /* the report updates
NOPAS()                  /* no password
READ(le),#6              /* Read supplier for this article
IF #OK THEN BEGIN        /* If supplier not present
le#1=#6                  /* Set suppliernumber
le#2="I made this"       /* and name
END
le#6=le#6+#3             /* Update supplier fields
WRITE(le)                /* insert or update supplier record
```

11. Export / Import from external files

This chapter describes the functions for read/write of textfiles with data for transfer to other systems.

11.1. EXPORT - Export of data to a textfile

number EXPORT(fields *par1*, filename *par2*, text *par3*, text *par4**6, text *par5*, text *par6**6)

Description: EXPORT exports data to a textfile. The function can be used to transfer data between systems, to spreadsheets and wordprocessing systems.
The fields that are given in *par1* has to be entered as text, eg. "#1-99" (in quotes).
The filename in *par2* is defaulted to TMP if a directory is omitted, the default extension is .OUT and if the filename is completely omitted the report name as c:/tmp/DM1007.OUT for report number 7.
With *par3* and *par5* you can control the recordlength and lineseperators for the file.
par4 is normally used only with fixedlength files for transport to mainframe systems.
par6 consists of 6 characters used to control the layout of a commaseparated file. Note that " in this string must be written as the two characters \". As standard all alphanumeric fields are written as "xxxx", where the character " (quote) is converted to a ' (single quote). The numeric fields are written as 99.99, where . (dot) is the decimal point. All fields are separated with , (comma).
The export file may now be closed using EXPORT("CLOSE"). This may be useful if you want to CHAIN notepad to view the file.
Returnvalue: None.
See also: IMPORT
Example:

```
AFTER                          /* AFTER selections
EXPORT("LE#1-99","le.csv")     /* all fields are exported (CSV)
EXPORT("#1-6","le.csv","","","","--,\"'.")  /* Same as above
```

This example will create the file le.csv with the following lines:

```
"100","HUMBER LTD.","HUMBER STREET 223","4711 COPENHAGEN S"
"102","AX & AX LTD.","SEA PARK ROAD 43","2100 COPENHAGEN",,25000
"105","WEBB'S SUPPLIERS LTD.","EAST STREET 373","4711 COPENHAGEN F",,500
```

Example:

```
EXPORT("#1-6","le.ssv","000001","","","--;. ,")  /* all fields exported (SSV)
```

This example will create the file le.ssv with the following lines:

```
SW-Tools
100;HUMBER LTD.;HUMBER STREET 223;4711 COPENHAGEN S;;123,25
```

Example:

```
EXPORT("#1-2,5-6","a","-80","1")  /* fixed fieldlength and no crlf
```

11.2. IMPORT - Import data from textfile (RAP)

IMPORT(fields *par1*, filename *par2*, text *par3*, text *par4**6, text *par5*, text *par6**6)

Description: The function reads data from a textfil.
The fields that are given in *par1* has to be entered in "" (quotes). It is possible to give simple calculations together with the field specification as IMPORT("#1-5,+6")

Operator	Function
+	Add up these fields
-	Subtract from these fields
&	Skip these fields
=	Set fields equal
:xx	Skip to position xx in the record

The physical filename given in *par2* may contain a path, e.g. "c:\\export\\le.csv".
Returnvalue: None.
See also: EXPORT
Example:

```
                              /* the reports mainfile is le (supplier file)
   UPDATE(1)                  /* the report updates
   NOPAS()                    /* no password
   IMPORT("#1-6","le.csv")    /* read one record from textfile
   READ(LE),#1                /* Check if supplier LE present
   LE#6=LE#6+#6               /* Add amount read to old balance
   IF #OK LET("LE#1-6=#1-6")  /* If new supplier move all fields
   WRITE(LE)                  /* Modify/Create LE supplier
```

Example:

```
   IMPORT("#1-6","le.ssv","","","","--;- -")  /* import fra (SSV) tekstfil
```

11.2.1. **IMPOCONT** - Continuation of import (RAP)

IMPOCONT(fields *par1*)
Parameters: *par1* : the fields to be set by read of data from the textfile
Description: IMPOCONT continues import of more fields from the same record and recordposition as last IMPORT reached. Used when a recordtype by start of the record may cause import of different fields.
See also: IMPORT, IMPONEXT, IMPOTHIS

11.2.2. **IMPONEXT** - Import of next record (RAP)

IMPONEXT(fields *par1*)
Parameters: *par1* : the fields to be set by read of data from the textfile
Description: IMPONEXT reads a new record from the import file and imports the fields from here. Used when a field indicates that one or more records follows with related informations for this mainrecord.
See also: IMPORT, IMPOCONT, IMPOTHIS

11.2.3. **IMPOTHIS** - Reimport this record (RAP)

IMPOTHIS(fields *par1*)
Parameters: *par1* : the fields to be set by read of data from the textfile
Description: IMPOTHIS imports the current record again. Used when a recordtype by start of the record may cause import of different fields.
See also: IMPORT, IMPOCONT, IMPONEXT

11.3. FTP - File Transfer Processor

number FTP(Number *par1*, Text *par2*)

Par2: FTP command
Description: The FTP function has been build in to enable advanced users to transfer files for example in a report based on a SSV file containing filenames. For a complete set of commands you should consult a FTP manual. Note that freefields may be used for the command and that the 32 bit version supports long filenames.
The example shows the transfer of a file from a Quattro system with the special command QUATTRO for transfer with headerblock and XQUAT to remove the additional FTP informations from the so transferred file.
Returnvalue: For OPEN: FTP Handle, all other: FTP error code, 0=OK
Example:

```
#10=FTP(0,"open 200.0.0.9")          /* Connect to Server
#11=FTP(#10,"user cms mypas")        /* Log in as user cms password mypas
#11=FTP(#10,"binary")                /* Switch on binary transfer
#11=FTP(#10,"quattro")               /* Switch on Quattro backup mode
#11=FTP(#10,"get /X.BASIC/0/AFIL c:/mydir/myfil") /* Get the file
if #11<>0 FTP(#10,"error")           /* Display error message
#11=FTP(#10,"xquat c:/mydir/myfil")  /* Convert from Quattro
#11=FTP(#10,"quit")                  /* Thats it
```

12. Multiple companies and merge of files

The functions descriped below are intended for use with multiple companies with separate tables/database systems and for merging different files with same record layout.

12.1. ACCESS- Check if file exists

number ACCESS(Filename *par1*)
Parameters: *par1* : Filename
Description: Check if the given file is present, returns 0 if file is found.
Returnvalue: 0 if file is found.
See also: OPEN
Example: IF ACCESS("myfile.ssv")=0 MESS("Ok ? ")

12.2. **COMNO** - Get current company id

text COMNO(Fileid *par1*)
Parameters: *par1* : Blank of fileid
Description: This function retreives the current company id of the given file, if no file id is given for the mainfile.
Returnvalue: Company id.
See also: OPCOM
Example: #1 = OPCOM() */* Get current company id, eg. "001"*

12.3. ENDSUM - Additional grande total when using more mainfiles

ENDSUM()
Parameters: None
Description: On a report with more seperate lists caused by the use of either the MERGE or the OPCOM function with totals for each list you may obtain an additional total of all printed records by placing an ENDSUM() calculation line.
The system fields #CO and #CN will be printed as *** at the ENDSUM page.
Returnvalue: None
See also: KEYS, MERGE, OPCOM
Example: ENDSUM() */* Print additional end total*

12.4. **FILENAME** - Current filename for an open file

text FILENAME(fileid *par1*)
Parameters: *par1* : Fileid
Description: This function returns the filename for the file which are currently opened with the given fileid.
Returnvalue: Real filename.
See also: OPEN
Example: #1 = FILENAME(va) */* Gives "c:/rapfil/ssv/isa/va.ssv"*

12.5. OPEN - Open a file with a specific name

number OPEN(fileid *par1*, Filename *par2*, Driver *par3*)

par3 : 0 or database interface number

Description: With the use of this function you may open a specific file instead of the one allready opened and associated with this fileid. The former opened file will be closed.
An error message is given if the filename is not present or the file cannot be opened for any other reason.
If *par3* is given the file is forced open with this database interface type as defined in BASIS.SSV by database driver installation.

Returnvalue: 0=ok, <>0=error.

See also: ACCESS, FILENAME, MERGE, OPCOM

Example:

```
FIRST
OPEN(va,"c:/swtools/demo/va.ssv")  /* Use this specific file

OPEN(va,#50)                       /* Enter filename by start report
```

12.5.1. OPEN - Temporary close of files

OPEN(fileid *par1*, Constant *par2*)

par2 : "-"
Description: Files may be closed to allow CHAINED programs to access these NOTE that the MAIN file must not be closed in this way.
Returnvalue: 0=ok, <>0=error.
See also: FILENAME, MERGE, OPCOM
Example:

```
OPEN("ku","-")          /* will temporary close file to allow:
CHAIN("command.com /c edit c:\\windows\\system\\ku.ssv")

OPEN("ku","+")          /* will reopen file again
```

12.6. MERGE - Merging of more mainfiles in one report (RAP)

number MERGE(fileid *par1*, Filename *par2*, Driver *par3*)

par3 : 0 or database interface number

Description: With the use of this function you can merge multiple files into one list. For the MERGE routine you may either give a fileid in *par1* if the file is defined seperately or a filename in *par2* as in OPEN. The involved files must have the same structure.
An error message is given if the filename is not present or the file cannot be opened for any other reason.
If *par3* is given the file is forced open with this database interface type as defined in BASIS.SSV by database driver installation.
A report using MERGE will normally be sorted to gain the merge effect, eg. sorted on articlenumber. If MERGE is used without sorting you will first get a list from the normal mainfile followed by a list from each of the merged files. The ENDSUM function may be used to get a grande total of all printed records.
If MERGE is called without parameters at all a MERGENUMBER is return as 1 for the mainfile, 2 for the first merged file, 3 for the next and so on.

Without parameters: MERGENUMBER from 1 and onwards.

See also: ENDSUM, OPCOM, OPEN

Example:

```
MERGE(0,"c:/swtools/demo/va.ssv")   /* Merge this file
MERGE(le)                           /* And the le file
#12=MERGE()                         /* Get mergenumber 1,2 or 3
```

12.7. OPCOM - Open files in different companies

number OPCOM()

par3 : 0 or database interface number
Description: The OPCOM function enables you to access more companies on one report.
A report can be made to run once for each stated company by placing OPCOM("111,777-888") or OPCOM(#50) where field 50 is a startdata inputfield. Such a report can then be extended with a total for all companies using ENDSUM or can be sorted eg. to collect all informations of one article in all companies.
The system fields #CO and #CN may be used to print the company id and company name in the heading.
An article list defined on the file va can be made to collect informations of the article from another company also by placing OPCOM(VA,"555") folloved by READ(VA). By this va#8 contains the holding for the actual company whereas VA#8 is the holding for company 555.
A statistics report where each record in the statistics file contains a company number can open this company files by use of OPCOM(0,#47)
If *par3* is given the file is forced open with this database interface type as defined in BASIS.SSV by database driver installation.
If OPCOM is used without parameters the current company number is returned.
From the COMPANY.SSV file the company names are read. If the company id's in *par3* contains ranges the valid companies herein are taken from this file.

Without parameters: MERGENUMBER from 1 and onwards.
See also: COMNO, ENDSUM, MERGE, OPEN
Example:

```
OPCOM("001,777-888")            /* Run the report with these companies
OPCOM("*")                      /* Run the report with all companies
OPCOM(va,"123")                 /* Use the article file company 123
OPCOM(0,"777")                  /* Company 777 for all other but mainfile
OPCOM(-1,"888")                 /* Use company 888 for all files

OPCOM(#50)                      /* Enter companies by start
```

13. IQ/DATAMASTER functions

The functions are designed for DATAMASTER and can not be used in reports. Some of the functions are usefull also in IQ which will then be indicated in the text.

13.1. DISABLE- Disable input for a program (IQ)

DISABLE(programno *par1*)
Parameters: *par1* : Programnumber to disable.
Description: Disables all input for the given program number.
Returnvalue: None.
See also: ENABLE , FOCUS
Example: DISABLE(20)

13.2. **DISP** - Display of changed fields (IQ)

DISP(fields *par1*)
Parameters: *par1* : "" or fields to redisplay
Description: DISP is to be used when you in a calculation for a field changes the value of other fields and these fields are shown on the screen. If DISP is not present you cannot be sure that the newly calculated value really is shown.
The DISP() command displaying all fields is extended to possibility of stating just selected fields as DISP("#1,4")
Returnvalue: None.
See also:
Example: DISP()

13.3. DOFUNCTION - Execute external function (IQ)

DOFUNCTION(Function *par1*, text *par2*, programno *par3*)

par3 : Optional programno
Description: DOFUNCTION sends the message <functionno> to the running IQ-Program or to the open <program>. A key may be passed to the READ functions.
The list of valid function numbers is found in the calculations listbox for 'Calculations by selection of function'.
Returnvalue: None.
See also: CHAIN, PLSNEXT, TRANSMIT
Example:

```
DOFUNCTION(505,#1,20) /* ask program 20 to read a record using key #1
DOFUNCTION(550)       /* Zooms the current screen
```

13.4. **ENABLE**- Enable input for a program (IQ)

ENABLE(Programno *par1*)
Parameters: *par1* : Program number to enable
Description: Enables all input for the given program number after DISABLE.
Returnvalue: None.
See also: DISABLE , FOCUS
Example: ENABLE(20) /* *Enable program 20*

13.5. **FOCUS** - Activate program (IQ)

FOCUS(Programno *par1*)
Parameters: *par1* : Programnumber to activate.
Description: Activates input and sets focus to the given program number.
Returnvalue: None.
See also: DISABLE, ENABLE
Example: FOCUS(20) */* Program 20 becomes active*

13.6. FUNC - Current update mode for a record (IQ)

number FUNC(fileid *par1*)
Parameters: *par1* : Fileid
Description: Dependent of the users input DATAMASTER decides if update of a certain record is nessesary and how this should be performed. FUNC is then used in the write calculation to branch to the proper routine.
Returnvalue:

Mode	Function
0	No update nessesary
1	An existing record should be modified
2	A new record should be inserted
3	An existing record should be deleted

See also: SETUPD, ON
Example:

```
ON FUNC(cu) GOSUB MAINWRT,MAININS,MAINDEL
IF FUNC(va)!=3 LET #27=#27+va#4
```

13.7. GETINFO - Get additional program information (IQ/DM)

number GETINFO(number *par1*, text *par2*)

par2 : Field reference

Description: This function allows you to get some special information from an IQ/DM program. The type 0 and 1 will return the unique id of the window, which may be used by other functions to manipulate the window. A sample of this is present in the OLE manual.
When the type is 2 to 5 the function requires a field reference in *par2*. For example, to get the start column for article field number 7 *par2* should equal "va#7". The coordinates of a field is here given in the actual size of the field defined in IQ/DM.
If you require the actual coordinates of a field according to the scale factor currently used, e.g. zoom in/out, use the type 6 to 9 instead.

Type 2-9 returns a field coordinate. The value can be held in a 9,T2 field format.

Example:

```
GETINFO(0)            /* Get the IQ program window id
GETINFO(2,"va#7");    /* Get the start x coordinate of va field 7
```

13.8. HELP - Display box with help for field (IQ)

HELP(field *par1*)
Parameters: *par1* : Fieldreference
Description: HELP("#31") displays a messagebox with help for the given field
See also: MESS
Example: HELP("#31")

13.9. **ISACTIVE** - Ask if program is active (IQ)

number ISACTIVE(Programno *par1*)
Parameters: *par1* : Programno
Description: Test if <program> is active.
Returnvalue: Returns 1 if <program> is active, 0 else.
See also: CHAIN, EXIT, WAIT
Example: IF ISACTIVE(20)=0 CHAIN(20) */* Start program 20 if not done*

13.10. KEYON - Switch key input field ON/OFF (IQ)

KEYON(number *par1*)

Description: KEYON(0) removes the key input field, (1) reactivates this.
Returnvalue: None.
See also:
Example: KEYON(0) */* Remove the key input field*

13.11. **LINE** - Retrieve or set the current line number (IQ/DM)

number LINE(number *par1*)
Parameters: *par1* : Type of information to get
Description: The function will retrieve or set the IQ/DM line number. The line number is the counter for lines defined in a program defined as **va#1-6l** or **le#1-6/va#1-6**.
If *par1* equals 0 the function returns the line number of the currently active line.
if *par1* equals -1 the function returns the number of lines defined for the programs. If the programs was defined as **va#1-6l,t5** the return value would be **5**
if *par1* is greater than 0 the function sets the active line to *par1*.
Returnvalue: A line number/count or zero if the functions sets the line number.
Example:

```
#20=LINE()  /* Get current active line number
```

13.12. LOOP - Call a routine for all records in the linebuffer (IQ)

LOOP(label *par1*)
Parameters: *par1* : Label (the routinename) to be called
Description: For each record read in a list program and for each transaction in a transaction program the internal linebuffer is filled with the read field values together with the result of the calculations for that line (non-global workfields).
In the writeroutine of such program LOOP is used to call a writeroutine for each single line. Also LOOP is used to recalculate SUM of all transaction lines.
Returnvalue: None.
See also: GOSUB, ON
Example:

```
LOOP(MAIN)          /* Writing the lines in a LIST-program
LOOP(TRANS)         /* Writing transaction lines
LOOP(SUMIT)         /* Recalculation of transaction SUM
LOOP(TRANSDEF)      /* Change of keyvalue for all transactions
```

13.13. **MENUCH** - Flip menu checked flag (IQ)

MENUCH(Menuno *par1*)
Parameters: *par1* : Menunumbers
Description: Flip checkflag on the given menu numbers (see MENUS) and update the according internal flag for program control.
Returnvalue: None.
See also: MENUUPD, MENUS
Example: MENUCH("31-32") */* Flip talk and listen menu*

13.14. MENUS - Menu control (IQ)

MENUS(Menuno *par1*)
Parameters: *par1* : -xxx=Deactivate, +xxx=Activete the menupoints xxx

Menunumber	Function
1/11	Insert new record in mainfile/transactions
2/12	Amend a record in mainfile/transactions
3/13	Delete a record in mainfile/transactions
4/14	Superindex search on mainfile/transactions
5/15	Selections on mainfile/transactions
6/16	Superindex fielddefinition on mainfile/transactions
20	Search, list must match input
21/22/23/24/25	Transactions, Next/Previus/First/Last/Direction
26	Display key during search
27	Case sensitive search
31/32	Talk/Listen to other programs
41/42/43/44	Mainfile, Next/Previus/First/Last
51/52/53	Calculations/Amend form/Save program
54/55	Parameter menus
61/62/63/64	New program, Delete program, Print program, Start program
100-149	Index locked and index number
999	Activate everything

Description: The MENUS function may be used both in DATAMASTER and IQ to deactivate certain menupoints.
MENUS can also be activated all from start by calling IQ from Windows with the -m+xxx or -m-xxx parameter. Especially to amend the calculations for a program with the calculations deactivated you will have to select IQ as eg: C:\SWTOOLS\IQWIN -m999
Returnvalue: None.
See also: MENUCH, MENUUPD
Example: MENUS("-51-55") */* Deactivate amendments of this program*

13.15. MENUUPD - Add/Control menu (IQ)

MENUUPD(Menuno *par1*, number *par2*, number *par3*)

par3 : Text
Description: Add / Control menu manually.
MENUUPD(1,2000,"My &Own menu") Adds function 2000 to menu number 1.
By selection of this new menupoint the user calculations labelled FU2000: in the function section will be performed.
Returnvalue: None.
See also: MENUCH, MENUS
Example: MENUUPD(1,2000,"My &Own menu") */* Add function 2000 to menu number 1.*

13.16. NEXTFLD - Jump to input field (IQ)

NEXTFLD(field *par1*)
Parameters: *par1* : Field number for next input
Description: NEXTFLD can be used to overwrite the fixed input sequence dependent on the calculations.
Together with the field specifications you may give program number or line number.
Returnvalue: None.
See also: NEXTFLDSEQ, SEQ
Example:

```
IF #4<#3 NEXTFLD(#3)
NEXTFLD("#10")       /* sets next input field to field 10.
NEXTFLD("#10.2")     /* jumps to field 10 on line 2
NEXTFLD("5.#10")     /* jumps to program 5 field 10
```

13.17. NEXTFLDSEQ - Jump to input field in sequence (IQ)

NEXTFLDSEQ(number *par1*, number *par2*)

par2 : Field number
Description: Jump to a distinct field in one of the field sequences.
Returnvalue: None.
See also: SEQ , NEXTFLD
Example: NEXTFLDSEQ(2,1) */* Jump to the first field given in input sequence 2*

13.18. OBJECTADDSTRING - Add string to object (IQ)

OBJECTADDSTRING(fields *par1*, text *par2*, text *par3*)

par3 : Text to use as index
Description: The function inserts a text in an object. The function result varies depending on the object type. In order to use the function correctly, please have the following rules in mind:

Object	**Meaning**
BUTTON	The function sets the text displayed for the button
COMBOBOX	The function adds a new element to the list
EDITBOX	The function set the text in the editbox. If the flag for multiple edit lines has been set the text will be added to the previous text
LISTBOX	The function adds a new element to the list

Parameter *par3* is only used if the object type is COMBOBOX or LISTBOX. The parameter must contain the normal value of the field.
Returnvalue: None.
See also: OBJECTCLEAR
Example: OBJECTADDSTRING("va#7",gr#2,gr#1) */* Display name and use no. as index*

13.19. OBJECTCLEAR - Clear contents of object (IQ)

OBJECTCLEAR(fields *par1*)
Parameters: *par1* : Field on the form, e.g. va#7
Description: The function clears the contents of an object.
Returnvalue: None.
See also: OBJECTADDSTRING
Example:

```
OBJECTCLEAR("va#7")     /* clear all previous values
START(gr),""            /* read all values from Article group table
NEXT(gr)
   OBJECTADDSTRING("va#7",gr#2,gr#1) /* Display name and use no. as index
REPEAT(gr)
```

13.20. OBJECTGETSTRING- Get index of an objects selected item (IQ/DM)

text OBJECTGETSTRING(field *par1*)
Parameters: *par1* : Field on the form, e.g. va#7
Description: The function retrieves the normal value of the combobox/listbox field. It returns the value equal to the *par3* used when calling OBJECTADDSTRING.
The use of this function will normally be by click on the combo/listbox field.
Returnvalue: The normal (index) value of the current selected item.
See also: OBJECTADDSTRING
Example:

```
#20=OBJECTGETSTRING("va#6")  /* Get current selected supplier number
```

13.21. **PLSNEXT** - Prepare and read mainfile (IQ)

PLSNEXT(number *par1*, text *par2*, number *par3*,)

Description: Prepare and perform read of mainfile according to the given mode. Used by the menus and by page down/up etc. If inputflag is set, key is used, otherwise read is next/prior/direct.
Returnvalue: None.
See also: DOFUNCTION, TRANSMIT
Example: PLSNEXT(0,#1,1) */* read the next record using #1 as key*

13.22. SEQ - Change of input sequence (IQ)

SEQ(number *par1*, fields *par2*)

par2 : Fieldnumbers in the new sequence
Description: The parameterpage informations of field sequence is overwritten by use of this function.
Returnvalue: None.
See also: NEXTFLD, NEXTFLDSEQ
Example:

```
SEQ(2,"va#2-3,5")          /* Set the normal amendment sequence
IF #7=1 SEQ(2,"va#4,3")    /* Special for this article group
```

13.23. SETUPD - Mark a file on a line for updating (IQ)

SETUPD(fileid *par1*)
Parameters: *par1* : Fileid for update
Description: When 'critical' fields in the main file are changed this may cause change of all transaction records. Normally modified transactions only will be written.
Returnvalue: None.
See also: LOOP
Example: SETUPD(va)

13.24. **SHOW**- Enable/Disable/Show/Hide a field (IQ/DM)

number SHOW(field *par1*, number *par2*)

3 = Hide field
Description: This function allows you to enable/disable a field or show/hide a field.
Returnvalue: None.
Example:

```
SHOW("va#7",1)        /* Disable field va#7
```

13.25. SUPER - Prepare superindex search (IQ)

SUPER(fileid *par1*) , text *par2*

par2 : Key
Description: The SUPER function initialises the NEXT read for use of superindex
Returnvalue: None.
See also: NEXT, START
Example:

```
SUPER(va),#21     /* NEXT uses superindex search for the text in #21
NEXT(va)          /* Must be follow to actually read the record
SUPER(va)         /* Superindex is switched off
SUPER(va),"#1-3"  /* Superindex fields is set to field 1-3
```

13.26. TRANSMIT- Update other IQ programs (IQ)

TRANSMIT(number *par1*, text *par2*, text *par3*)

par3 : Optional Connection

Description: Transmit the current records to one or more programs using the automatic connections or if given the connection stated.

```
    Progid=""   Send to all other
           "20" Just update program 20 if active
           "le" Send to all programs using the file le as mainfile

    Connection = ""     Use automatic connections between files
                 "1,2P" Use field 1 and 2 packed as connection
                 "va.01.6" Use va as transmitting file to the other program
                           Read the other mainfile index 1 using field 6.
```

Returnvalue: None.

See also: PLSNEXT, DOFUNCTION

Example: TRANSMIT(0,"","") */* Update all other programs using auto connections*

13.27. TRANSSEL- Define IQ transaction selections (IQ)

TRANSEL(text *par1*, number *par2*)

Description: Scan the given input if any and define transaction selections if input contains formulas as #15>0. Used by arrows in the keyfield
Returnvalue: None.
See also:
Example: TRANSSEL("#15>20",1) /* *Define selection*

14. SYSTEM functions

These functions are designed for use in special programs where you for example requires direct access to files / paths.

14.1. DEBUG- Switch on debug window (IQ)

DEBUG(number *par1*)

Description: DEBUG(1) will open a window which lists all calculated expressions and their program number/label when these are carried out.
The DEBUG window is closed when IQ is closed.
Returnvalue: None
See also: WIF, WIFS
Example: DEBUG(1) */* Switch on debug window*

14.2. **EXEC**- Execute text as calculation line

EXEC(text *par1*, Programno *par2*)

par2 : **(IQ/DM)**program number

Description:

```
#20="#2=17"
EXEC(#20)
```

executes the textstring stored in field 20 as a calculation.
When using freefields in the EXEC function you must use the WW#nn references which you may obtain from a print of the program definitions.
In general the string passed to the EXEC function is not pretranslated and checked as normal calculation lines. This has especially importance when used in RAPGEN where the C-Syntax of the calculations must be followed. We strongly advise non-programmes to keep the use of EXEC in RAPGEN simple without involving function calls. Invalid function parameters may lead to general protection faults.
One point should be especially noticed for RAPGEN: #15=2 sets field 15 equal to 2 ALSO when used as IF #15=2 LET #16=3. You must double the equal sign in such a statement following the C-Syntax giving: IF (#15==2) LET #16=3
IQ: EXEC(#20,15) switches to the active program 15 and executes the given calculation.

Returnvalue: None

See also:

Example: EXEC(#20) */* Execute a calculation entered by start report*

14.3. GETFLD- Set SY structure pointers (IQ)

GETFLD(text *par1*)
Parameters: *par1* : Field specification
Description: This function sets system variables (SY#..) to point to the definition of the given field. The field definition may then be read/changed. Special and programmers use only.
Returnvalue: None

14.4. INSTALL- Aktivation of external functions

INSTALL(text *par1*, text *par2*, text *par3*, text *par4*)

par4 : Optional my function name
Description: Programmers knowing function definitions from other DLL's may now include these as IQ functions.

NOTE: Improper use of this function may cause system breakdown.

Returnvalue: None
See also:
Example:

```
INSTALL("a.dll","b","3,[ss]")
   activates #20=B(#21) from a.dll, #20 and #21 being short variables

INSTALL("some.dll","aname","3,[sCl]","FUNNY")
   activates #30=FUNNY(#31,#32) as function aname from some.dll
   return value #30 short, parameters #31 as char pointer, #32 as long.
```

14.5. SYSPAR - Get systemparameter

text SYSPAR(number *par1*)

Description: SYSPAR reads the given system parameter. Only the above mentioned values are usefull.
Returnvalue: The systemparameter.
See also: SYSPARSET
Example: #1 = SYSPAR(4) */* Get the current TMP path*

14.6. SYSPARSET - Set value of a systemparameter

SYSPARSET(number *par1*, text *par2*)

par2 : New value of this system parameter
Description: SYSPARSET changes the value for the given system parameter.
Returnvalue: None.
See also: SYSPAR
Example: SYSPARSET(4,"c:/mytmp/") */* Set a new TMP directory*

14.7. USERINFO - Get information about user

text USERINFO(number *par1*)

17=User defined
Description: This function gets the requested user information.
The number in *Par1* refers to the fieldnumber in the system file US where you may define field 11 to 17 individually for each installation, just be careful if later upgrading the version of TRIO.
Returnvalue: String containing the user information.
Example:

```
#11=USERINFO(6)  /* Get the user first remark
```

14.8. **WIF** - Testprint (IQ)

WIF(text *par1*)
Parameters: *par1* : Text to print
Description: WIF gives testprint without disturbing the screenlayout to the file c:/wif
Returnvalue: None
See also: WIFS, DEBUG
Example: WIF("Here I am") */* Output text*

14.9. **WIF**- Testprint (RAP)

WIF(text *par1* , text *par2*)

. **Description:** WIF gives testprint to the file c:/wif
Returnvalue: None
See also: WIFS, DEBUG
Example: WIF("Field equals %s.",#2) /* *Testprint*

14.10. **WIFS**- Testprint of fields (IQ)

WIFS(fields *par1*)
Parameters: *par1* : Fields to print
Description: WIFS gives testprint of the given field values to the file c:/wif
Returnvalue: None
See also: WIF, DEBUG
Example: WIFS("va#1-3,le#2") */* Output field values*

Index