



natural

Version 4.1.2 for Mainframes | System Functions

This document applies to Natural Version 4.1.2 for Mainframes and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© Copyright Software AG 1979 - 2003.
All rights reserved.

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

Table of Contents

Natural System Functions	1
Natural System Functions	1
Natural System Functions for Use in Processing Loops	2
Natural System Functions for Use in Processing Loops	2
Using System Functions in Processing Loops	2
Specification/Evaluation	2
Use in SORT GIVE FUNCTIONS Statement	3
Arithmetic Overflows in AVER, NAVER, SUM or TOTAL	3
Statement Referencing (<i>r</i>)	4
Detailed Descriptions	4
AVER(<i>r</i>)(<i>field</i>)	4
COUNT(<i>r</i>)(<i>field</i>)	4
MAX(<i>r</i>)(<i>field</i>)	4
MIN(<i>r</i>)(<i>field</i>)	4
NAVER(<i>r</i>)(<i>field</i>)	4
NCOUNT(<i>r</i>)(<i>field</i>)	5
NMIN(<i>r</i>)(<i>field</i>)	5
OLD(<i>r</i>)(<i>field</i>)	5
SUM(<i>r</i>)(<i>field</i>)	5
TOTAL(<i>r</i>)(<i>field</i>)	5
Examples	5
Mathematical Functions	10
Mathematical Functions	10
Miscellaneous Functions	13
Miscellaneous Functions	13
POS - Field Identification Function	14
POS - Field Identification Function	14
RET - Return Code Function	16
RET - Return Code Function	16
SORTKEY - Sort-Key Function	17
SORTKEY - Sort-Key Function	17
*MINVAL/*MAXVAL - Evaluate the Minimum/Maximum	19
*MINVAL/*MAXVAL - Evaluate the Minimum/Maximum	19
Function	19
operand	19
result-format-length	19
Resulting Format/Length Conversion Rule Tables	20
Explicit Specification of the Resulting Format/Length	20
Implicit Specification of the Resulting Format/Length	20
Evaluating the <i>result-format-length</i>	22
Format/Length Evaluation Order	23
*TRANSLATE - Translate to Lower/Upper Case Characters	24
*TRANSLATE - Translate to Lower/Upper Case Characters	24
Function	24
operand	24
LOWER	24
UPPER	24
Example	24
Output	25
*TRIM - Remove Leading and/or Trailing Blanks	26
*TRIM - Remove Leading and/or Trailing Blanks	26
Function	26
operand	26

LEADING 26

TRAILING 26

Operand Not Followed by a Keyword 27

Examples 28

 Example Using an Alphanumeric Argument 28

 Output 30

 Example Using an Alphanumeric Argument 30

 Output 31

Natural System Functions

This document describes various Natural "built-in" functions for use in certain statements.

Note:

As of Natural Version 6 for Windows and UNIX, all new system functions are preceded by an asterisk (*) to avoid naming conflicts with, for example, user-defined variables in existing applications.

The same will apply to all new system functions in future Natural for Mainframes versions.

- System Functions Grouped by Function

- System Functions for Use in Processing Loops
- Mathematical Functions
- Miscellaneous Functions

See also:

- System Functions in the Natural Programming Guide.
- Example of System Variables and System Functions in the Natural Programming Guide.

Natural System Functions for Use in Processing Loops

This document describes those Natural system functions which can be used in a program loop context.

The following topics are covered below:

- Using System Functions in Processing Loops
 - Specification/Evaluation
 - System Functions in SORT GIVE FUNCTIONS Statement
 - Arithmetic Overflows in AVER, NAVER, SUM or TOTAL
 - Statement Referencing (*r*)
 - Detailed Descriptions
 - AVER(*r*)(field)
 - COUNT(*r*)(field)
 - MAX(*r*)(field)
 - MIN(*r*)(field)
 - NAVER(*r*)(field)
 - NCOUNT(*r*)(field)
 - NMIN(*r*)(field)
 - OLD(*r*)(field)
 - SUM(*r*)(field)
 - TOTAL(*r*)(field)
 - Examples
-

Using System Functions in Processing Loops

The following topics are covered:

- Specification/Evaluation
- Use in SORT GIVE FUNCTIONS Statement
- Arithmetic Overflows in AVER, NAVER, SUM or TOTAL
- Statement Referencing (*r*)

Specification/Evaluation

Natural system functions may be specified in

assignment and arithmetic statements:

- MOVE,
- ASSIGN,
- COMPUTE,
- ADD,
- SUBJECT,
- MULTIPLY,
- DIVIDE

input/output statements:

- DISPLAY,
- PRINT,
- WRITE

that are used within any of the following statement blocks:

- AT BREAK,
- AT END OF DATA,
- AT END OF PAGE,

that is, for all FIND, READ, HISTOGRAM, SORT or READ WORK FILE processing loops.

If a system function is used within an AT END OF PAGE statement, the corresponding DISPLAY statement must include the GIVE SYSTEM FUNCTIONS clause.

Records rejected by a WHERE clause are not evaluated by a system function.

If system functions are evaluated from database fields which originated from different levels of processing loops initiated with a FIND, READ, HISTOGRAM or SORT statement, the values are always processed according to their position in the loop hierarchy. For example, values for an outer loop will only be processed when new data values have been obtained for that loop.

If system functions are evaluated from user-defined variables, the processing is dependent on the position in the loop hierarchy where the user-defined variable was introduced in reporting mode. If the user-defined variable is defined before any processing loop is initiated, it will be evaluated for system functions in the loop where the AT BREAK, AT END OF DATA or AT END OF PAGE statement is defined. If a user-defined variable is introduced within a processing loop it will be processed the same as a database field from that processing.

For selective referencing of system function evaluation for user-defined variables it is recommended to specify a loop reference with the user-defined variable to indicate in which loop the value is to be processed. The loop reference may be specified as a statement label or source code line number.

Use in SORT GIVE FUNCTIONS Statement

System functions may also be referenced when they have been evaluated in a GIVE FUNCTIONS clause of a SORT statement.

For a reference to a system function evaluated with a SORT GIVE FUNCTIONS statement, the name of the system function must be prefixed with an asterisk (*).

Arithmetic Overflows in AVER, NAVER, SUM or TOTAL

Fields to which the system functions AVER, NAVER, SUM and TOTAL are to be applied must be long enough (either by default or user-specified) to hold any overflow digits. If any arithmetic overflow occurs, an error message will be issued.

Normally, the length is the same as that of the field to which the system function is applied; if this is not long enough, use the NL parameter to increase the output length as follows:

SUM(field)(NL=nn)

This will not only increase the output length but also causes the field to be made longer internally.

Statement Referencing (*r*)

Statement referencing is also available for system functions (see also Statement Reference Notation - *r* in the section User-Defined Variables of the Natural Statements documentation).

By using a statement label or the source-code line number (*r*) you can determine in which processing loop the system function is to be evaluated for the specified field.

Detailed Descriptions

AVER(*r*)(*field*)

Format/length: Same as field.

Exception: for a field of format N, AVER(*field*) will be of format P (with the same length as the field).

This system function contains the average of all values encountered for the field specified with AVER. AVER is updated when the condition under which AVER was requested is true.

COUNT(*r*)(*field*)

Format/length: P7

COUNT is incremented by 1 on each pass through the processing loop in which it is located. COUNT is incremented regardless of the value of the field specified with COUNT.

MAX(*r*)(*field*)

Format/length: Same as field.

This system function contains the maximum value encountered for the field specified with MAX. MAX is updated (if appropriate) each time the processing loop in which it is contained is executed.

MIN(*r*)(*field*)

Format/length: Same as field.

This system function contains the minimum value encountered for the field specified with MIN. MIN is updated (if appropriate) each time the processing loop in which it is located is executed.

NAVER(*r*)(*field*)

Format/length: Same as field.

Exception: for a field of format N, NAVER(*field*) will be of format P (with the same length as the field).

This system function contains the average of all values - excluding null values - encountered for the field specified with NAVER. NAVER is updated when the condition under which NAVER was requested is true.

NCOUNT(*r*)(*field*)

Format/length: P7

NCOUNT is incremented by 1 on each pass through the processing loop in which it is located unless the value of the field specified with NCOUNT is a null value.

NMIN(*r*)(*field*)

Format/length: Same as field.

This system function contains the minimum value encountered - excluding null values - for the field specified with NMIN. NMIN is updated (if appropriate) each time the processing loop in which it is located is executed.

OLD(*r*)(*field*)

Format/length: Same as field.

This system function contains the value which the field specified with OLD contained prior to a control break as specified in an AT BREAK condition, or prior to the end-of-page or end-of-data condition.

SUM(*r*)(*field*)

Format/length: Same as field.

Exception: for a field of format N, SUM(*field*) will be of format P (with the same length as the field).

This system function contains the sum of all values encountered for the field specified with SUM. SUM is updated each time the loop in which it is located is executed. When SUM is used following an AT BREAK condition, it is reset after each value break. Only values that occur between breaks are added.

TOTAL(*r*)(*field*)

Format/length: Same as field.

Exception: for a field of format N, TOTAL(*field*) will be of format P (with the same length as the field).

This system function contains the sum of all values encountered for the field specified with TOTAL in all open processing loops in which TOTAL is located.

Examples**System Functions Example 1:**


```

/* EXAMPLE 'ATBEX4': AT BREAK USING NATURAL SYSTEM FUNCTIONS
/*****
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 NAME
    2 CITY
    2 SALARY (2)
  1 #INC-SALARY (P11)
END-DEFINE
/*****
LIMIT 4
EMPLOOP. READ EMPLOY-VIEW BY CITY STARTING FROM 'ALBU'
  COMPUTE #INC-SALARY = SALARY (1) + SALARY (2)
  DISPLAY NAME CITY SALARY (1:2) 'CUMULATIVE' #INC-SALARY
  SKIP
  AT BREAK CITY
    WRITE NOTITLE
      'AVERAGE:' T*SALARY (1) AVER(SALARY(1)) /
      'AVERAGE CUMULATIVE:' T*#INC-SALARY
      AVER(EMPLOOP.) (#INC-SALARY)
  END-BREAK
END-READ
/*****
END

```

NAME	CITY	ANNUAL SALARY	CUMULATIVE
HAMMOND	ALBUQUERQUE	22000 20200	42200
ROLLING	ALBUQUERQUE	34000 31200	65200
FREEMAN	ALBUQUERQUE	34000 31200	65200
LINCOLN	ALBUQUERQUE	41000 37700	78700
AVERAGE :		32750	
AVERAGE CUMULATIVE :			62825

System Functions Example 3:

```

/* EXAMPLE 'AEDEX1S': AT END OF DATA (STRUCTURED MODE)
DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 PERSONNEL-ID
    2 NAME
    2 FIRST-NAME
    2 SALARY (1)
    2 CURR-CODE (1)
END-DEFINE
/*
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME SALARY (1) CURR-CODE (1)
/*****
  AT END OF DATA
    IF *COUNTER (EMP.) = 0
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-IF
    WRITE NOTITLE / 'SALARY STATISTICS:'
                  / 7X 'MAXIMUM:'  MAX(SALARY(1)) CURR-CODE (1)
                  / 7X 'MINIMUM:'  MIN(SALARY(1)) CURR-CODE (1)
                  / 7X 'AVERAGE:'  AVER(SALARY(1)) CURR-CODE (1)
  END-ENDDATA
/*****
END-FIND
END

```

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
11100328	BERGHAUS	ROSE	70800	€
11100329	BARTHEL	PETER	42000	€
11300313	AECKERLE	SUSANNE	55200	€
11300316	KANTE	GABRIELE	61200	€
11500304	KLUGE	ELKE	49200	€
SALARY STATISTICS:				
	MAXIMUM:	70800	€	
	MINIMUM:	42000	€	
	AVERAGE:	55680	€	

System Functions Example 4:

```

/* EXAMPLE 'AEPEX1S': AT END OF PAGE (STRUCTURED MODE)
/*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
/*****
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
    NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
/*****
  AT END OF PAGE
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
/*****
END-READ
/*****
END
    
```

NAME	CURRENT POSITION	SALARY	CURRENCY CODE
CREMER	ANALYST	34000	USD
MARKUSH	TRAINEE	22000	USD
GEE	MANAGER	39500	USD
KUNEY	DBA	40200	USD
NEEDHAM	PROGRAMMER	32500	USD
JACKSON	PROGRAMMER	33000	USD
AVERAGE SALARY: ...		33533	USD

Mathematical Functions

The following mathematical functions are supported in arithmetic processing statements (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) and in logical condition criteria:

Function	Format/Length	Explanation
ABS (<i>field</i>)	same as <i>field</i>	Absolute value of <i>field</i> .
ATN (<i>field</i>)	F8 (*)	Arc tangent of <i>field</i> .
COS (<i>field</i>)	F8 (*)	Cosine of <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , COS (<i>field</i>) will be "1".
EXP (<i>field</i>)	F8 (*)	Exponentiation of exponent <i>field</i> to base e , i.e. e^{field} , where e is Euler's number.
FRAC (<i>field</i>)	same as <i>field</i>	Fractional part of <i>field</i> .
INT (<i>field</i>)	same as <i>field</i>	Integer part of <i>field</i> .
LOG (<i>field</i>)	F8 (*)	Natural logarithm of <i>field</i> .
SGN (<i>field</i>)	same as <i>field</i>	Sign of <i>field</i> (-1, 0, +1).
SIN (<i>field</i>)	F8 (*)	Sine of <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , SIN (<i>field</i>) will be "0".
SQRT (<i>field</i>)	(*) (**)	Square root of <i>field</i> . A negative value in the argument field will be treated as positive. On mainframe computers, the maximum number of digits before the decimal point of the argument is 22.
TAN (<i>field</i>)	F8 (*)	Tangent of <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , TAN (<i>field</i>) will be "0".
VAL (<i>field</i>)	same as target field	Extract numeric value from an alphanumeric <i>field</i> . The content of the <i>field</i> must be the character representation of a numeric value. Leading or trailing blanks in the <i>field</i> will be ignored; decimal point and leading sign character will be processed. If the target field is not long enough, decimal digits will be truncated (see also Field Truncation and Field Rounding in the section Rules for Arithmetic Assignment of the Natural Statements documentation). Note: VAL cannot be used in conjunction with the COMPUTE statement.

* On UNIX and Windows platforms, these functions are evaluated as follows: The argument is converted to format/length F8 and then passed to the operating system for computation; the result returned by the operating system has format/length F8, which is then converted to the target format.

** On mainframe computers, the following applies:
If *field* has format/length F4, format/length of **SQRT**(*field*) will be F4;
if *field* has format/length F8 or I, format/length of **SQRT**(*field*) will be F8;
if *field* has format N or P, format/length of **SQRT**(*field*) will be Nn.7 or Pn.7 respectively (where *n* is automatically calculated to be large enough).

A *field* to be used with a mathematical function - except VAL - may be a constant or a scalar; its format must be numeric, packed numeric, integer, or floating point (N, P, I or F).

A *field* to be used with the VAL function may be a constant, a scalar, or an array; its format must be alphanumeric.

Mathematical Functions Example:

```

/* EXAMPLE 'MATHEX': MATHEMATICAL FUNCTIONS
/*****
DEFINE DATA LOCAL
1 #A (N2.1) INIT <10>
1 #B (N2.1) INIT <-6.3>
1 #C (N2.1) INIT <0>
1 #LOGA (N2.6)
1 #SQRTA (N2.6)
1 #TANA (N2.6)
1 #ABS (N2.1)
1 #FRAC (N2.1)
1 #INT (N2.1)
1 #SGN (N1)
END-DEFINE
/*****
COMPUTE #LOGA = LOG(#A)
WRITE NOTITLE '=' #A 5X 'LOG' 40T #LOGA
/*****
COMPUTE #SQRTA = SQRT(#A)
WRITE '=' #A 5X 'SQUARE ROOT' 40T #SQRTA
/*****
COMPUTE #TANA = TAN(#A)
WRITE '=' #A 5X 'TANGENT' 40T #TANA
/*****
COMPUTE #ABS = ABS(#B)
WRITE // '=' #B 5X 'ABSOLUTE' 40T #ABS
/*****
COMPUTE #FRAC = FRAC(#B)
WRITE '=' #B 5X 'FRACTIONAL' 40T #FRAC
/*****
COMPUTE #INT = INT(#B)
WRITE '=' #B 5X 'INTEGER' 40T #INT
/*****
COMPUTE #SGN = SGN(#A)
WRITE // '=' #A 5X 'SIGN' 40T #SGN
/*****
COMPUTE #SGN = SGN(#B)
WRITE '=' #B 5X 'SIGN' 40T #SGN
/*****
COMPUTE #SGN = SGN(#C)
WRITE '=' #C 5X 'SIGN' 40T #SGN
/*****
END

```

#A:	10.0	LOG	2.302585
#A:	10.0	SQUARE ROOT	3.162277
#A:	10.0	TANGENT	0.648360
#B:	-6.3	ABSOLUTE	6.3
#B:	-6.3	FRACTIONAL	-0.3
#B:	-6.3	INTEGER	-6.0
#A:	10.0	SIGN	1
#B:	-6.3	SIGN	-1
#C:	0.0	SIGN	0

Miscellaneous Functions

The following topics are covered:

- POS - Field Identification Function
- RET - Return Code Function
- SORTKEY - Sort-Key Function
- *MINVAL/*MAXVAL - Minimum/Maximum Value of a Field
- *TRANSLATE - Translate Operand1 to Lower/Upper Case Character
- *TRIM - Remove Leading and/or Trailing Blanks from Operand1

Note:

As of Natural Version 6 for Windows and UNIX, all new system functions are preceded by an asterisk (*) to avoid naming conflicts with, for example, user-defined variables in existing applications. The same will apply to all new system functions in future Natural for Mainframes versions.

POS - Field Identification Function

Format/length: I4

The system function `POS(field-name)` contains the internal identification of the field whose name is specified with the system function.

`POS(field-name)` may be used to identify a specific field, regardless of its position in a map. This means that the sequence and number of fields in a map may be changed, but `POS(field-name)` will still uniquely identify the same field. With this, for example, you need only a single REINPUT statement to make the field to be MARKed dependent on the program logic.

Example:

```

DECIDE ON FIRST VALUE OF ...
  VALUE ...
    COMPUTE #FIELDX = POS(FIELD1)
  VALUE ...
    COMPUTE #FIELDX = POS(FIELD2)
  ...
END-DECIDE
...
REINPUT ... MARK #FIELDX

```

If the field specified with POS is an array, a specific occurrence must be specified; for example, "POS(FIELDX(5))". POS cannot be applied to an array range.

POS and *CURS-FIELD

The system function `POS(field-name)` may be used in conjunction with the Natural system variable `*CURS-FIELD` to make the execution of certain functions dependent on which field the cursor is currently positioned in.

`*CURS-FIELD` contains the internal identification of the field in which the cursor is currently positioned; it cannot be used by itself, but only in conjunction with `POS(field-name)`. You may use them to check if the cursor is currently positioned in a specific field and have processing performed depending on that condition.

Example:

```

IF *CURS-FIELD = POS(FIELDX)
  MOVE *CURS-FIELD TO #FIELDY
END-IF
...
REINPUT ... MARK #FIELDY

```

Note:

The values of `*CURS-FIELD` and `POS(field-name)` serve only as internal identifications of the fields and cannot be used for arithmetic operations.

The value returned by `POS(field-name)` for an occurrence of an X-array (an array for which at least one bound in at least one dimension is specified as expansible) may change after the number of occurrences for a dimension of the array has been changed using the EXPAND, RESIZE or REDUCE statements.

Note for Natural RPC:

If `*CURS-FIELD` and `POS(field-name)` refer to a context variable, the resulting information can only be used within the same conversation.

See also Dialog Design, Field Sensitive Processing and Simplifying Programming, in the Natural Programming Guide.

RET - Return Code Function

Format/length: I4

The system function `RET(program-name)` may be used to receive the return code from a non-Natural program called via a `CALL` statement.

`RET(program-name)` can be used in an `IF` statement and within the arithmetic statements `ADD`, `COMPUTE`, `DIVIDE`, `MULTIPLY` and `SUBTRACT`.

Example:

```
DEFINE DATA LOCAL
1 #RETURN (I4)
...
END-DEFINE
...
...
CALL 'PROG1'
IF RET('PROG1') > #RETURN
    WRITE 'ERROR OCCURRED IN PROGRAM 1'
END-IF
...
```

SORTKEY - Sort-Key Function

Format/length: A253

Several national languages contain characters (or combinations of characters) which are not sorted in the correct alphabetical order by a sort program or database system, because the sequence of the characters in the character set used by the computer does not always correspond to the alphabetical order of the characters.

For example, the Spanish letter "CH" would be treated by a sort program or database system as two separate letters and sorted between "CG" and "CI" - although in the Spanish alphabet it is in fact a letter in its own right and belongs between "C" and "D".

Or it may be that, contrary to your requirements, lower-case and upper-case letters are not treated equally in a sort sequence, that letters are sorted after numbers (although you may wish them to be sorted before numbers), or that special characters (for example, hyphens in double names) lead to an undesired sort sequence.

In such cases, you can use the system function SORTKEY(*character-string*) to convert "incorrectly sorted" characters (or combinations of characters) into other characters (or combinations of characters) that are "correctly sorted" alphabetically by the sort program or database system.

The values computed by SORTKEY are then only used as sort criterion, while the original values are used for the interaction with the end-user.

You can use the SORTKEY function as an arithmetic operand in a COMPUTE statement and in a logical condition.

As *character-string* you can specify an alphanumeric constant or variable, or a single occurrence of an alphanumeric array.

When you specify the SORTKEY function in a Natural program, the user exit NATUSK nn will be invoked - nn being the current language code (that is, the current value of the system variable *LANGUAGE).

You can write this user exit in any programming language that provides a standard CALL interface. The *character-string* specified with SORTKEY will be passed to the user exit. The user exit has to be programmed so that it converts any "incorrectly sorted" characters in this string into corresponding "correctly sorted" characters. The converted character string is then used in the Natural program for further processing.

For details on the user exit, see your Natural Operations documentation.

Example:

```

DEFINE DATA LOCAL
  1 CUST VIEW OF CUSTOMERFILE
    2 NAME
    2 SORTNAME
END-DEFINE
...
*LANGUAGE := 4
...
REPEAT
  INPUT NAME
  SORTNAME := SORTKEY(NAME)
  STORE CUST
  END TRANSACTION
...
END-REPEAT
...
READ CUST BY SORTNAME
  DISPLAY NAME
END-READ
...

```

Assume that in the above example, at repeated executions of the INPUT statement, the following values are entered: "Sanchez", "Sandino" und "Sancinto" .

At the assignment of SORTKEY(NAME) to SORTNAME, the user exit NATUSK04 would be invoked. This user exit would have to be programmed so that it first converts all lower-case letters to upper-case, and then converts the character combination "CH" to "CX" - where X would correspond to the last character in the character set used, i.e. hexadecimally H'FF' (assuming that this last character is a non-printable character).

The "original" names (NAME) as well as the converted names to be used for the desired sorting (SORTNAME) are stored. To read the file, SORTNAME is used. The DISPLAY statement would then output the names in the correct Spanish alphabetical order:

Sancinto Sanchez Sandino

*MINVAL/*MAXVAL - Evaluate the Minimum/Maximum

This system function can only be used on Windows and UNIX.

$\left\{ \begin{array}{l} *MINVAL \\ *MAXVAL \end{array} \right\} \left([(IR=result-format/length)] \right. \\ \left. operand, \dots \right)$
--

Format/length: Format and length may be specified explicitly using the IR clause or evaluated automatically using the Format/Length Conversion Rule Tables below.

Function

The *MINVAL/*MAXVAL system function evaluates the minimum/maximum value of all given operand values. The result is always a scalar value. If an array is specified as operand, the minimum/maximum of all array fields is evaluated.

The *MINVAL/*MAXVAL system function may be specified as an operand in any position of a statement wherever *MINVAL/*MAXVAL is allowed. However, *MINVAL/*MAXVAL must not be used where a target variable is expected.

*MINVAL/*MAXVAL may not be nested in a system function.

When using alpha or binary data as an argument, if the data is the same (e.g. *MINVAL('AB', 'AB')), then the result is the argument with the smallest/largest length value.

operand

Operand	Possible Structure					Possible Formats										Referencing Permitted	Dynamic Definition		
<i>operand</i>	C	S	A	G		A	N	P	I	F	B	D	T					yes	no

result-format-length

The compiler tries to determine the *result-format-length* of the whole function. If the compiler cannot determine a format/length in a way that no loss of precision is guaranteed, the *format-length* must be set by the programmer using the IR operand extension.

$\left\{ \begin{array}{l} \textit{format-length} \\ \left(\left\{ \begin{array}{l} A \\ B \end{array} \right\} \right) \text{DYNAMIC} \end{array} \right\}$
--

The IR (Intermediate Result) clause may be used in order to specify explicitly the resulting format/length of the whole *MINVAL/*MAXVAL system function. For an assortment of valid result-format/lengths, refer to the Format/Length Conversion Rule Tables below.

Example:

```

DEFINE DATA LOCAL
1 #RESULTI      (I4)
1 #RESULTA      (A20)
1 #RESULTADYN   (A) DYNAMIC
1 #A(I4)         CONST <1234>
1 #B(A20)        CONST <'H'30313233'> /* '0123' stored
1 #C(I2/1:3)     CONST <2000, 2100, 2200>
END DEFINE
*
#RESULTA      := *MAXVAL((IR=A20)      #A, #B)      /*no error, I4->A20 is allowed!
#RESULTADYN   := *MAXVAL((IR=(A)DYNAMIC) #A, #B)      /*result is (A) dynamic
/* #RESULTI    := *MAXVAL((IR=I4)       #A, #B)      /*compiler error, because conv. A20->I4 is not allowed!
#RESULTI      := *MAXVAL((IR=I4)       #A, #C(*))    /*maximum of the array is evaluated
DISPLAY #RESULTA #RESULTADYN (AL=10) #RESULTI
END

```

Resulting Format/Length Conversion Rule Tables

There are two different ways to define the resulting format/length of the whole *MINVAL/*MAXVAL system function:

- Explicit specification of the resulting format/length
- Implicit specification of the resulting format/length which is evaluated automatically using all the operands used as argument of the *MINVAL/*MAXVAL system function.

Explicit Specification of the Resulting Format/Length

The resulting format/length of the whole *MINVAL/*MAXVAL system function may be specified by the IR clause. All operands specified will be converted into this resulting format/length, if this is possible without any loss of precision. Afterwards the minimum/maximum of all the converted operands will be evaluated and one single scalar value with the evaluated format/length will be set as result of the whole system function.

Implicit Specification of the Resulting Format/Length

If no IR clause is used inside the *MINVAL/*MAXVAL system function, the resulting format/length will be evaluated regarding the format/length of all operands specified as arguments inside the *MINVAL/*MAXVAL system function. The format/length of each operand is taken and combined with the format/length of the next following operand of the argument list. The resulting format/length of two single operands are then evaluated using the Format/Length Conversion Rule Tables below.

The Format/Length Conversion Rule Table is separated into two different subtables. The first table covers all the numeric combinations of two different operands. The second table covers all other formats and lengths which may be used for *MINVAL/*MAXVAL system function operands.

All combinations not shown in the two tables below are invalid and must not be applied inside the argument list of the *MINVAL/*MAXVAL system function. The keyword "FLF" indicates that the IR clause must be used in order to define the resulting format/length, because there otherwise may be a loss of precision.

		Second Operand				
First Operand	Format-length	I1	I2	I4	Pa,b, Na,b	F4, F8
	I1	I1	I2	I4	Pmax(3,a).b	F8
	I2	I2	I2	I4	Pmax(5,a).b	F8
	I4	I4	I4	I4	Pmax(10,a).b	F8
	Px.y, Nx.y	Pmax(3,x).y	Pmax(5,x).y	Pmax(10,x).y	if max(x,a) + max(y,b) <= 29 Pmax(x,a).max(y,b) else <i>FLF</i>	if y=0 and x <=15; F8 else <i>FLF</i>
	F4, F8	F8	F8	F8	if b=0 and a <=15 F8 else <i>FLF</i>	F8

Legend:

<i>FLF</i>	Format-length declaration forced. The resulting format must be specified using the IR clause.
Ix	Format/length is Integer. x specifies the number of bytes which are used to store the Integer value.
Fx	Format/length is Float. x specifies the number of bytes which are used to store the Float value
Px.y Pa,b	Packed format with corresponding number of digits before the decimal point (x,a) and the precision (y,b).
Nx.y Na,b	Numeric format with corresponding number of digits before the decimal point (x,a) and the precision (y,b).
Pmax(c,d).e	The resulting format is packed. The length is evaluated by the information following. The number of digits before the decimal point is the maximum value of c and d. Ther precision value is e.
Pmax(c,d).max(e,f)	The resulting format is packed. The length is evaluated by the information following. The number of digits before the decimal point is the maximum value of c and d. Ther precision value is the maximum value of e and f.

		Second Operand			
First Operand	Format-length	D	T	Aa, A dynamic	Ba, B dynamic
	D	D	T	NA	NA
	T	T	T	NA	NA
	Ax, A dynamic	NA	NA	A dynamic	B dynamic
	Bx, B dynamic	NA	NA	B dynamic	B dynamic

Legend:

<i>FLF</i>	Format-length declaration forced. The resulting format must be specified using the IR clause.
<i>NA</i>	This combination is not allowed.
<i>D</i>	Date format
<i>T</i>	Time format
<i>Bx</i>	Binary format with length x.
<i>Ax</i>	Alphanumeric format with length x.
<i>B dynamic</i>	Binary format with dynamic length.
<i>A dynamic</i>	Alphanumeric format with dynamic length.

Evaluating the *result-format-length*

Using the rules described above, the compiler is able to process the source operands by regarding pairs of operands and calculating an intermediate result for each pair. The first pair consists of the first and the second operand, the second pair of the intermediate result and the third operand, etc. After all operands have been processed, the last result shows the comparison of format and length which will be used to compare all operands in order to evaluate the minimum/maximum. When you use method of format-length evaluation, the operand *format-lengths* can appear in any order.

Example:

```

DEFINE DATA LOCAL
1 A (I2)      INIT <34>
1 B (P4.2)    INIT <1234.56>
1 C (N4.4)    INIT <12.6789>
1 D (I1)      INIT <100>
1 E (I4/1:3)  INIT <32, 6745, 456>
1 #RES-MIN (P10.7)
1 #RES-MAX (P10.7)
END-DEFINE
*
MOVE *MINVAL(A, B, C, D, E(*)) TO #RES-MIN
MOVE *MAXVAL(A, B, C, D, E(*)) TO #RES-MAX
DISPLAY #RES-MIN #RES-MAX
END

```

Output:

```

#RES-MIN          #RES-MAX
-----
12.6789000      6745.0000000

```

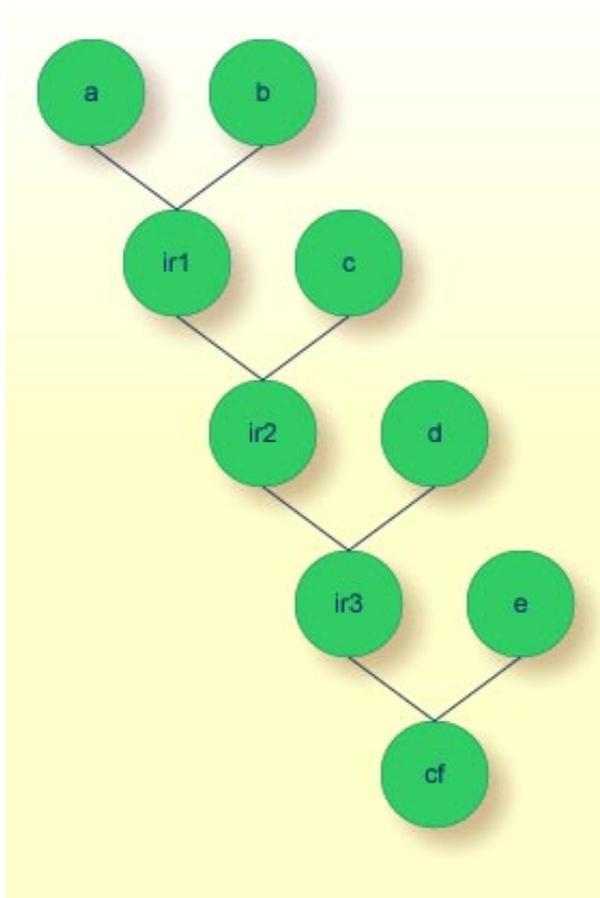
The following table shows the single steps evaluating the format/length of the example automatically. It shows the intermediate result (ir) of all steps and the comparison format/length (cf) which is used as result-format/length.

Evaluation Order	Name of First Operand	Format/Length of First Operand or Intermediate Result	Name of Second Operand	Format/Length of Second Operand or Intermediate Result	Format/Length of the Intermediate Result (ir)
1.	A	I2	B	P4.2	ir1 = P5.2
2.	ir1	P5.2	C	N4.4	ir2 = P5.4
3.	ir2	P5.4	D	I1	ir3 = P5.4
4.	ir3	P5.4	E	I4	cf = P10.4

During runtime, all operands are converted into the cf format/length. Then, all converted values are compared, and the corresponding minimum/maximum is evaluated.

Format/Length Evaluation Order

The following graphic represents the order in which format and length are evaluated:



Legend:

- ir1, ir2, ir3 Intermediate result 1, 2, 3
- cf Comparison of format and length

*TRANSLATE - Translate to Lower/Upper Case Characters

This system function can only be used on Windows and UNIX.

```
*TRANSLATE(operand [ , { LOWER } ] )
```

Format/length: same as the operand.

Function

The system function *TRANSLATE converts the characters of an alpha or binary operand to upper case or lower case characters. The content of operand is not modified.

*TRANSLATE may be specified as an operand in any position of a statement wherever an operand of Format A or B is allowed. However, *TRANSLATE must not be used where a target variable is expected. You may not nest *TRANSLATE in a system function.

operand

Operand	Possible Structure			Possible Formats												Referencing Permitted	Dynamic Definition			
<i>operand</i>	C	S	A			A	B												yes	no

LOWER

Translates operand to lower case characters when using the keyword LOWER as a second argument.

```
*TRANSLATE (... , LOWER)
```

UPPER

Translates operand to upper case characters when using the keyword UPPER as a second argument.

```
*TRANSLATE (... , UPPER)
```

Example

*TRANSLATE - Translate to Lower/Upper Case Characters

Output

```
DEFINE DATA LOCAL
1 #SRC (A)DYNAMIC INIT <'aBcDeFg !$$%&/()=?'>
1 #DEST (A)DYNAMIC
END-DEFINE

PRINT 'Source string to be translated:.....' #SRC

MOVE *TRANSLATE(#SRC, UPPER) TO #DEST
PRINT 'Source string translated into upper case:' #DEST

MOVE *TRANSLATE(#SRC, LOWER) TO #DEST
PRINT 'Source string translated into lower case:' #DEST
END
```

Output

```
Source string to be translated:..... aBcDeFg !$$%&/()=?

Source string translated into upper case: ABCDEFG !$$%&/()=?

Source string translated into lower case: abcdefg !$$%&/()=?
```

*TRIM - Remove Leading and/or Trailing Blanks

This system function can only be used on Windows and UNIX.

```
*TRIM(operand [ , { LEADING } TRAILING ] )
```

Format/length: same as operand (A or B)/DYNAMIC

Function

The system function *TRIM removes all leading and/or trailing blanks from an alphanumeric or a binary string. The content of operand is not modified. When using a dynamic variable like operand, the length of this variable is adapted according to the result.

The *TRIM system function may be specified as an operand in any position of a statement wherever an operand of Format A or B is allowed.

However, *TRIM must not be used where a target variable is expected. You may not nest *TRIM in a system function.

Note:

The operand behaves differently when removing trailing blanks depending on whether it used as a static or dynamic variable. If the operand is a static variable, it is not possible to remove trailing blanks using *TRIM. This is because for static variables, the remaining trailing positions of the variable memory are filled with space characters. However, it is possible to remove trailing blanks when using dynamic variables.

operand

Operand	Possible Structure				Possible Formats										Referencing Permitted	Dynamic Definition		
<i>operand</i>	C	S	A		A	B											yes	no

LEADING

Removes leading blanks from operand, when using the keyword LEADING as a second argument.

```
*TRIM( . . . , LEADING )
```

TRAILING

```
*TRIM( . . . , TRAILING )
```

Removes leading blanks from operand, when using the keyword TRAILING as a second argument.

Operand Not Followed by a Keyword

Removes leading and trailing blanks from operand, when not using a keyword as a second argument.

*TRIM(. . .)

Examples

Example Using an Alphanumeric Argument

```
END-DEFINE
DEFINE DATA LOCAL
/*****
/* STATIC VARIABLE DEFINITIONS
/*****
1 #SRC (A15) INIT <' ab CD '>
1 #DEST (A15)

/* FOR PRINT OUT WITH DELIMITERS
1 #SRC-PRN (A20)
1 #DEST-PRN (A20)

/*****
/* DYNAMIC VARIABLE DEFINITIONS
/*****
1 #DYN-SRC (A)DYNAMIC INIT <' ab CD '>
1 #DYN-DEST (A)DYNAMIC

/* FOR PRINT OUT WITH DELIMITERS
1 #DYN-SRC-PRN (A)DYNAMIC
1 #DYN-DEST-PRN (A)DYNAMIC

END-DEFINE

PRINT 'static variable definition:'
PRINT '-----'
COMPRESS FULL ':' #SRC ':' TO #SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 123456789012345      123456789012345'

MOVE *TRIM(#SRC, LEADING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '-----'
COMPRESS FULL ':' #DYN-SRC ':' TO #DYN-SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 1234567890      12345678'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT '":' := delimiter character to show the start and ending of a string!'
```

Output

```

#SRC-PRN          #DEST-PRN
-----

static variable definition:
-----

123456789012345      123456789012345
:  ab CD           :      :ab  CD           :      *TRIM(#src, leading)
:  ab CD           :      : ab   CD           :      *TRIM(#src, trailing)
:  ab CD           :      :ab   CD           :      *TRIM(#src)

dynamic variable definition:
-----

1234567890          12345678
:  ab CD  :          :ab  CD  :          *TRIM(#src, leading)
:  ab CD  :          : ab CD:          *TRIM(#src, trailing)
:  ab CD  :          :ab   CD:          *TRIM(#src)

':' := delimiter character to show the start and ending of a string!

```

Example Using an Alphanumeric Argument

```
DEFINE DATA LOCAL
/*****
/* STATIC VARIABLE DEFINITIONS
/*****
1 #SRC (B10) INIT <H'2020FFFF2020FFFF2020'>
1 #DEST (B10)

/*****
/* DYNAMIC VARIABLE DEFINITIONS
/*****
1 #DYN-SRC (B)DYNAMIC INIT <H'2020FFFF2020FFFF2020'>
1 #DYN-DEST (B)DYNAMIC
END-DEFINE

FORMAT LS=100

PRINT 'static variable definition:
'PRINT '-----'
MOVE *TRIM(#SRC, LEADING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC)'

PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '-----'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST ' *TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST ' *TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST ' *TRIM(#SRC)'

PRINT ' '

PRINT 'hex."20" := space character'
END
```

Output

Output

*TRIM - Remove Leading and/or Trailing Blanks

static variable definition:

```
2020FFFF2020FFFF2020 0000FFFF2020FFFF2020 *TRIM(#src, leading)
2020FFFF2020FFFF2020 00002020FFFF2020FFFF *TRIM(#src, trailing)
2020FFFF2020FFFF2020 00000000FFFF2020FFFF *TRIM(#src)
```

dynamic variable definition:

```
2020FFFF2020FFFF2020 FFFF2020FFFF2020 *TRIM(#src, leading)
2020FFFF2020FFFF2020 2020FFFF2020FFFF *TRIM(#src, trailing)
2020FFFF2020FFFF2020 FFFF2020FFFF *TRIM(#src)
```

hex.'20' := space character