Natural

System Functions

Version 8.3.7 for Windows

March 2016

ADABAS & NATURAL

This document applies to Natural Version 8.3.7 for Windows.

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

Document ID: NATWIN-NNATFUNCTIONS-837-20160330

Table of Contents

Preface

This documentation describes various Natural "built-in" functions for use in certain statements; see *System Functions* in the *Programming Guide*.

This documentation is organized under the following headings:

System Functions for Use in Processing Loops	Describes Natural system functions which can be used in a program loop context.
Mathematical System Functions	Describes the system functions which are supported in arithmetic processing statements and in logical condition criteria.
Miscellaneous System Functions	System functions to evaluate the minimum or maximum; system function for field identification; system function to receive the return code from a non-Natural program; system function to convert "incorrectly sorted" characters; system function for lower/upper case translation; system function to remove leading and/or trailing blanks.
Functions Supplied as Natural Objects	Describes functions which are supplied as Natural objects to support, for example, URL encoding and Base64 conversion.

See also *Example of System Variables and System Functions* in the *Programming Guide*.

I

1 Natural System Functions for Use in Processing Loops

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

Using System Functions in Processing Loops

Specification/Evaluation

Natural system functions may be specified in

- assignment and arithmetic statements:
  - MOVE
  - ASSIGN
  - COMPUTE
  - ADD
  - SUBTRACT
  - 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 Statement

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

For a reference to a system function evaluated with a SORT GIVE 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 option of the SORT GIVE statement 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 *Referencing of Database Fields Using (r) Notation* in the section *User-Defined Variables* of the *Programming Guide*).

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.

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.

Whether the result of `NCOUNT` is an array or a scalar value depends on its argument (field). The number of the resulting occurrences is the same as of field.

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

- Example 1 - AT BREAK Statement with Natural System Functions OLD, MIN, AVER, MAX, SUM, COUNT
- Example 2 - AT BREAK Statement with Natural System Function AVER
- Example 3 - AT END OF DATA Statement with System Functions MAX, MIN, AVER
- Example 4 - AT END OF PAGE Statement with System Function AVER

Example 1 - AT BREAK Statement with Natural System Functions OLD, MIN, AVER, MAX, SUM, COUNT

```
** Example 'ATBEX3': AT BREAK  (with Natural system functions)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 3
READ EMPLOY-VIEW LOGICAL BY CITY = 'SALT LAKE CITY'
  DISPLAY NOTITLE CITY NAME 'SALARY' SALARY(1) 'CURRENCY' CURR-CODE(1)
  /*
  AT BREAK OF CITY
    WRITE /  OLD(CITY) (EM=X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X)
      31T '   MINIMUM:' MIN(SALARY(1))  CURR-CODE(1) /
      31T '   AVERAGE:' AVER(SALARY(1)) CURR-CODE(1) /
      31T '   MAXIMUM:' MAX(SALARY(1))  CURR-CODE(1) /
      31T '       SUM:' SUM(SALARY(1))  CURR-CODE(1) /
      35T COUNT(SALARY(1)) 'RECORDS FOUND' /
  END-BREAK
  /*
  AT END OF DATA
```

```
    WRITE 22T 'TOTAL (ALL RECORDS):'
              T*SALARY TOTAL(SALARY(1))  CURR-CODE(1)
  END-ENDDATA
END-READ
*
END      ↩
```

Output of program ATBEX3:

```
        CITY                   NAME            SALARY   CURRENCY
-------------------- -------------------- ---------- --------

SALT LAKE CITY       ANDERSON                    50000 USD
SALT LAKE CITY       SAMUELSON                   24000 USD

S A L T   L A K E   C I T Y      MINIMUM:        24000 USD
                                 AVERAGE:        37000 USD
                                 MAXIMUM:        50000 USD
                                     SUM:        74000 USD
                                         2 RECORDS FOUND

SAN DIEGO            GEE                         60000 USD

S A N   D I E G O                MINIMUM:        60000 USD
                                 AVERAGE:        60000 USD
                                 MAXIMUM:        60000 USD
                                     SUM:        60000 USD
                                         1 RECORDS FOUND

                     TOTAL (ALL RECORDS):       134000 USD ↩
```

Example 2 - AT BREAK Statement with Natural System Function AVER

```
** Example 'ATBEX4': AT BREAK  (with 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
EMPL. 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 1
  /*
  AT BREAK CITY
```

```
    WRITE NOTITLE
      'AVERAGE:'            T*SALARY (1)  AVER(SALARY(1)) /
      'AVERAGE CUMULATIVE:' T*#INC-SALARY AVER(EMPL.) (#INC-SALARY)
  END-BREAK
END-READ
*
END                                                          ↩
```

Output of program ATBEX4:

```
NAME                 CITY           ANNUAL    CUMULATIVE
                                               SALARY
-------------------- -------------------- ---------- ------------

HAMMOND              ALBUQUERQUE             22000        42200
                                             20200

ROLLING              ALBUQUERQUE             34000        65200
                                             31200

FREEMAN              ALBUQUERQUE             34000        65200
                                             31200

LINCOLN              ALBUQUERQUE             41000        78700
                                             37700

AVERAGE:                                     32750
AVERAGE CUMULATIVE:                                       62825
```

Example 3 - AT END OF DATA Statement with System Functions MAX, MIN, AVER

```
** Example 'AEDEX1S': AT END OF DATA
************************************************************************
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                                                          ↩
```

Output of program AEDEX1S:

```
PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

11100328  BERGHAUS             ROSE                    70800 DM
11100329  BARTHEL              PETER                   42000 DM
11300313  AECKERLE             SUSANNE                 55200 DM
11300316  KANTE                GABRIELE                61200 DM
11500304  KLUGE                ELKE                    49200 DM

SALARY STATISTICS:
       MAXIMUM:      70800 DM
       MINIMUM:      42000 DM
       AVERAGE:      55680 DM
```

Example 4 - AT END OF PAGE Statement with System Function AVER

```
** 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
```

Output of program AEPEX1S:

```
        NAME                  CURRENT             SALARY   CURRENCY
                             POSITION                        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
```

2 Mathematical System 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(field)`	same as *field*	Absolute value of *field*.
`ATN(field)`	F8 (*)	Arc tangent of *field*.
`COS(field)`	F8 (*)	Cosine of *field*.
`EXP(field)`	F8 (*)	Exponentiation of exponent *field* to base e , that is, e^{field}, where e is Euler's number.
`FRAC(field)`	same as *field*	Fractional part of *field*.
`INT(field)`	same as *field*	Integer part of *field*.
`LOG(field)`	F8 (*)	Natural logarithm of *field*.
`SGN(field)`	same as *field*	Sign of *field* (-1, 0, +1).
`SIN(field)`	F8 (*)	Sine of *field* .
`SQRT(field)`	F8 (*)	Square root of *field*. A negative value in the argument field will be treated as positive. The maximum number of digits before the decimal point of the argument is 22.
`TAN(field)`	F8 (*)	Tangent of *field*.
`VAL(field)`	same as target field	Extract numeric value from an alphanumeric *field*. The content of the *field* must be the alphanumeric (code page or Unicode) character representation of a numeric value. Leading or trailing blanks in the *field* will be ignored; decimal point and leading sign character will be processed.

Function	Format/Length	Explanation
		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 *Programming Guide*).

* 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.

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

Output of program MATHEX:

```
#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       ↩
```

II Miscellaneous System Functions

The following topics are covered:

3 *MINVAL/*MAXVAL - Evaluate the Minimum/Maximum

```
{ *MINVAL } ([(IR=result-format/length)] operand,...)
{ *MAXVAL }
```

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 Natural system function `*MINVAL/*MAXVAL` 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.

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

Restrictions

When using the system function `*MINVAL/*MAXVAL`, the following restrictions apply:

- `*MINVAL/*MAXVAL` must not be used where a target variable is expected.
- You may not nest `*MINVAL/*MAXVAL` in a system function.

Syntax Description

Operand Definition Table:

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

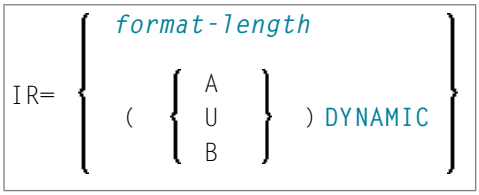
Syntax Element Description:

Syntax Element	Description
*MINVAL	Evaluates the minimum value of all given operand values.
*MAXVAL	Evaluates the maximum value of all given operand values.
operand	The operand(s) whose minimum/maximum values are to be evaluated by the *MINVAL/*MAXVAL system function.
result-format-length	Intermediate Result clause for **explicit specification** of the resulting format/length. See *IR Clause* below.

IR Clause

The IR (Intermediate Result) clause may be used in order to specify explicitly the *result-format/length* of the whole *MINVAL/*MAXVAL system function.

```
IR=result-format/length
```



For an assortment of valid result-format/lengths, refer to the ***Format/Length Conversion Rule Tables*** below.

Syntax Element Description:

Syntax Element	Description
format-length	The compiler tries to determine the resulting 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.
A, U or B	Format: Alphanumeric, Unicode or Binary for dynamic variable.
DYNAMIC	Instead of specifying a fixed format/length, you may specify an alphanumeric, Unicode or binary format with dynamic length.

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 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

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. 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.

Table 1

Covers all the numeric combinations of two different operands.

		Second Operand				
	Format- length	**I1**	**I2**	**I4**	**P*a*.*b*, N*a*.*b***	**F4, F8**
First Operand	**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
	P*x*.*y*, N*x*.*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 FLF	if *y*=0 and *x* <=15; F8 else FLF
	F4, F8	F8	F8	F8	if *b*=0 and *a* <=15 F8 else FLF	F8

Legend:

FLF	Format-length declaration forced. The resulting format must be specified using the IR clause.
I*x*	Format/length is Integer. *x* specifies the number of bytes which are used to store the Integer value.
F*x*	Format/length is Float. *x* specifies the number of bytes which are used to store the Float value.
P*x*.*y* **P***a*,*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*. The 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*. The precision value is the maximum value of *e* and *f*.

Table 2

Covers all other formats and lengths which may be used for `*MINVAL/*MAXVAL` system function operands.

	Second Operand					
	Format-length	**D**	**T**	**A*a*, A dynamic**	**B*a*, B dynamic**	**U*a*, U dynamic**
First Operand	**D**	D	T	NA	NA	NA
	T	T	T	NA	NA	NA
	A*x*, A dynamic	NA	NA	A dynamic	A dynamic	U dynamic
	B*x*, B dynamic	NA	NA	A dynamic	B dynamic	U dynamic
	U*x*, U dynamic	NA	NA	U dynamic	U dynamic	U dynamic

Legend:

NA	This combination is not allowed.
D	Date format.
T	Time format.
B*x*, B*a*	Binary format with length *x*, *a*.
A*x*, A*a*	Alphanumeric format with length *x*, *a*.
U*x*, U*a*	Unicode format with length *x*, *a*.
B dynamic	Binary format with dynamic length.
A dynamic	Alphanumeric format with dynamic length.
U dynamic	Unicode 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 this 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.

Notes:

1. If only a single operand is specified, *result-format-length* will be the format/length of this operand.
2. If a binary operand with a length in the range 1- 4 is specified as an argument inside the *MINVAL/*MAXVAL system function along with an alphanumeric or Unicode operand, the intermediate result (*result-format-length*) is evaluated to alphanumeric or Unicode format with dynamic length.

In this case, the value of the binary operand is considered to be a numeric value, which is converted to the *result-format-length* according to the data transfer rules (the binary numeric value is converted to unpacked format) before the minimum/maximum is evaluated.

Example:

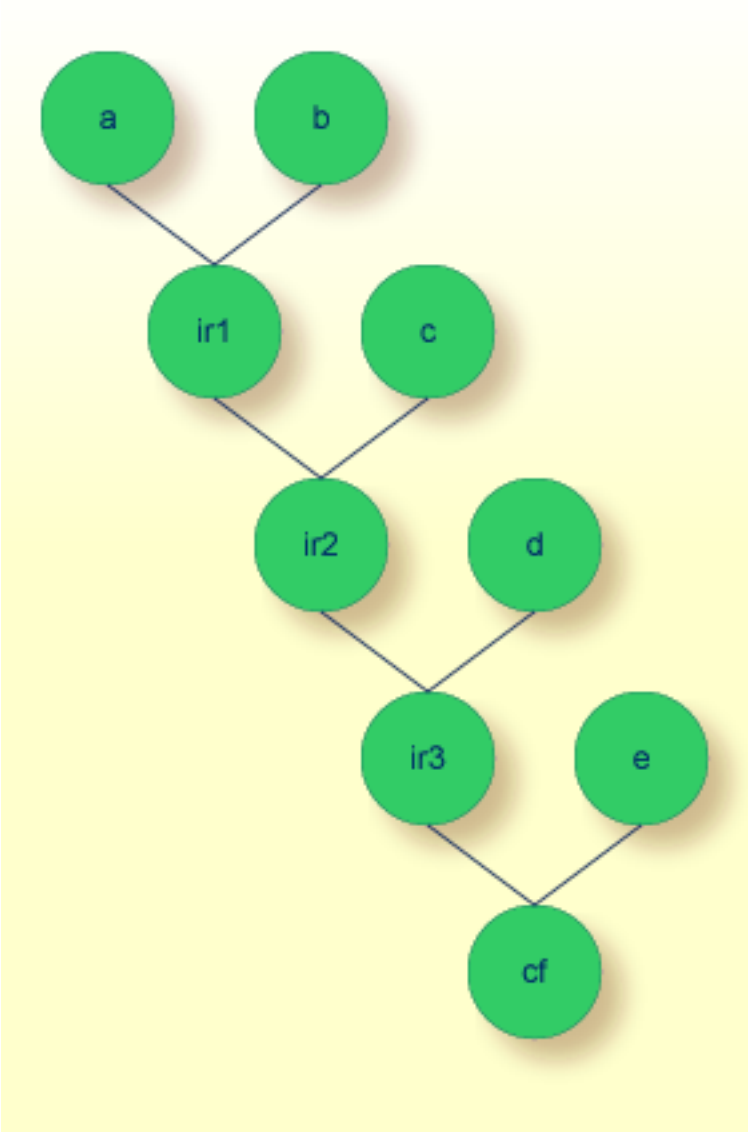
```
DEFINE DATA LOCAL
1 #B4 (B4)  INIT <1>
1 #A10(A10) INIT <"2">
END-DEFINE
WRITE "=" *MAXVAL(#A10, #B4) (AL=60) /* RESULT FORMAT-LENGTH IS (A)DYNAMIC: "2"
WRITE "=" *MINVAL(#A10, #B4) (AL=60) /* RESULT FORMAT-LENGTH IS (A)DYNAMIC: "1"
END
```

Intermediate *result-format-length* (#A10, #B4) is A dynamic.

So first #A10 is converted into A dynamic as well as #B4 is converted into A dynamic (considering data transfer rules), before the intermediate result of both operands 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	Resulting comparison format-length.

4 *TRANSLATE - Translate to Lower/Upper Case Characters

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

Format/length: same as *operand*.

Function

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

*TRANSLATE may be specified as an operand in any position of a statement wherever an operand of format A, U or B is allowed.

Restrictions

When using the system function *TRANSLATE, the following restrictions apply:

- *TRANSLATE must not be used where a target variable is expected.
- You may not nest *TRANSLATE in a system function.

Syntax Description

Operand Definition Table:

Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
operand	C	S	A			A	U	B									yes	no

Syntax Element Description:

Syntax Element	Description
*TRANSLATE (*operand*,LOWER)	**Lower Case Translation** When the keyword LOWER is used as a second argument, the character string in *operand* is translated to lower case.
*TRANSLATE (*operand*,UPPER)	**Upper Case Translation** When the keyword UPPER is used as a second argument, the character string in *operand* is translated to upper case.

Example

```
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 !§$%&/()=?
```

5 *TRIM - Remove Leading and/or Trailing Blanks

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

Format/length: same as *operand* (A, U or B)/DYNAMIC.

Function

The Natural system function *TRIM removes all leading and/or trailing blanks from an alphanumeric or a binary string. The content of the operand is not modified. When using a dynamic variable as 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, U or B is allowed.

Restrictions

When using the system function *TRIM, the following restrictions apply:

- *TRIM must not be used where a target variable is expected.
- You may not nest *TRIM in a system function.
- If the operand is a static variable, it is not possible to remove trailing blanks using *TRIM, because for static variables the remaining trailing positions of the variable memory are filled with space characters.

Syntax Description

Operand Definition Table:

Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
operand	C	S	A			A	U	B									yes	no

Syntax Element Description:

Syntax Element	Description
*TRIM(*operand*, LEADING)	**Remove Leading Blanks** When the keyword LEADING is used as a second argument, all leading blanks are removed from the string contained in *operand*.
*TRIM(*operand*, TRAILING)	**Remove Trailing Blanks** When the keyword TRAILING is used as a second argument, all trailing blanks are removed from the string contained in *operand*.
*TRIM(*operand*)	**Remove Both Leading and Trailing Blanks** When no keyword is used as a second argument, both the leading and the trailing blanks are removed from the string contained in *operand*.

Examples

Example 1 - Using an Alphanumeric Argument

```
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!'
END
```

Output of Example 1:

```
#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 2 - Using a Binary 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 of Example 2:

```
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
```

6 POS - Field Identification Function

Format/length:	I4

The system function POS(*field-name*) returns an identification of the field whose name is specified with the system function. The value returned is an internal representation of the field address.

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.

Note: POS cannot distinguish between two different variables that start at the same storage position (REDEFINE variables) since the internal field address returned by POS is the same for both.

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
```

Notes:

1. The values of *CURS-FIELD and POS(*field-name*) serve only as internal identifications of the fields and cannot be used for arithmetic operations.
2. 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.
3. 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.
4. In Natural for Ajax applications, *CURS-FIELD identifies the operand that represents the value of the control that has the input focus. You may use *CURS-FIELD in conjunction with the POS function to check for the control that has the input focus and perform processing depending on that condition.
5. *CURS-FIELD and POS(*field-name*) cannot distinguish between two different variables that start at the same storage position (REDEFINE variables) since the internal field addresses returned by *CURS-FIELD and POS(*field-name*) are the same for both variables.

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

7 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
...
```

8 SORTKEY - Sort-Key Function

SORTKEY (*character-string*)

This system function is used 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.

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*). The values computed by SORTKEY are 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.

The general calling conventions for external programs are explained in the description of the CALL statement.

For details on the calling conventions for user exits, see *User Exits*.

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" and "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 "C*x*" - where *x* would correspond to the last character in the character set used, i.e. hexadecimal 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
```

III Functions Supplied as Natural Objects

9 Functions Supplied as Natural Objects

This document describes functions that are implemented by using Natural objects of the type function.

These function objects (and their prototype definitions) whose names start with `SAG` are supplied in the Natural system library `SYSTEM` on the system file `FNAT`. Example function calls are provided in the system library `SYSEXPG`.

For detailed information on function calls, see the relevant section in the *Programming Guide*.

URL Encoding

Interfacing Natural applications with HTTP requests often requires that the URI (Uniform Resource Identifiers) is URL-encoded. The `REQUEST DOCUMENT` statement needs such a URL to access a document.

URL-Encoding (or Percent-Encoding) is a mechanism to replace some special characters in parts of a URL. Only characters of the US-ASCII character set can be used to form a URL. Some characters of the US-ASCII character set have a special meaning when used in a URL - they are classified as "reserved" control characters, which structure the URL string into different semantic subcomponents. The quasi standard concerning the generic syntax of an URL is laid down in RFC3986, a document composed by the Internet community. It describes under which conditions the URL-Encoding is needed. This includes the representation of characters which are not inside the US-ASCII character set (for example, Euro sign), and it describes the use of reserved characters.

Reserved characters are:

?	=	&	#	!	$	%	'	(	)	*	+	,	/	:	;	@	[	]

Non reserved characters are:

A-Z	a-z	0-9	-	_	.	~

A URL may only consist of reserved and non-reserved characters, other characters are not permitted. If other byte values are needed (which do not correspond to any of the reserved and non-reserved characters) or if reserved characters are used as data (which should not have a special semantic meaning in the URL context), they need to be translated into the "%-encoding" form - a percent sign, immediately followed by the two-digit hexadecimal representation of the code point, due to the Windows-1252 encoding scheme. This causes a plus sign (+) to appear as `%2B`, a percent sign (%) to appear as `%25` and an at sign (@) to appear as `%40` in the string.

The following encoding functions are operating the complete input string. You should take care not to encode a complete URL or parts of it if they contain control characters (reserved characters) which must not be translated into the percent-form. These functions should only be applied to

characters not permitted for use in a URL, and to characters with a special meaning inside the URL context, which are supplied as a data item.

Simple Encoding

The single input parameter contains the character string to be encoded or decoded. All data inside is regarded as represented in code page Windows-1252, regardless which session code page is really active at this time. The execution of the SAGENC/SAGDEC functions does not require Unicode support. The following characters are replaced with the corresponding US-ASCII hexadecimal equivalents.

Character	<	(	+	|	&	!	$	*	)	;	/	,	%	>	?	`	:	#	@	'	=	"	^	[	]	{	}	\
is encoded to %*nn*	3C	28	2B	7C	26	21	24	2A	29	3B	2F	2C	25	3E	3F	60	3A	23	40	27	3D	22	5E	5B	5D	7B	7D	5C

In addition, a space is replaced with a plus sign (+). All other characters are not translated and remain as they are. The simple encoding function should be sufficient in most cases of URL encoding. The result field returned is of format A dynamic.

The following functions are available:

- SAGENC - Simple encoding (format A to format A)
- SAGDEC - Simple decoding (format A to format A)

SAGENC - Simple Encoding (Format A to Format A)

The function SAGENC encodes a character string into its percent-encoded form. According to standard RFC3986, reserved characters and characters below US-ASCII x'7F' (*which are not allowed in a URL*) will be percent-encoded, a space character is replaced with a plus sign (+). Unreserved characters according to RCF3986 and characters above US-ASCII x'7F', such as German umlauts, are not encoded. If you want to encode such characters, use the extended encoding function SAGENCE.

Object	Description
SAGENC	This is the simple encoding function call.
SAGENCP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGENCP is optional.
URLX01	**Example program** contained in library SYSEXPG. `#URL-ENC := SAGENC(<#URL-DEC>)`

SAGDEC - Simple Decoding (Format A to Format A)

The function SAGDEC decodes the percent-encodings as provided by the function SAGENC. Besides the decoding string, no other input parameters are necessary.

Object	Description
SAGDEC	This is the simple decoding function call.
SAGDECP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGDECP is optional.
URLX01	**Example program** contained in library SYSEXPG. `#URL-DEC := SAGDEC(<#URL-ENC>) ↩`

Extended Encoding

The extended function considers all issues which are specified or recommended in RFC 3986. The following parameters may be considered (default settings shown in bold):

1. *<dynamic U-string>* to be encoded/decoded
2. Return code: <> 0 (Natural error) if error in MOVE ENCODED statement.
3. Error character if return code <> 0
4. Space character: %20/+/don't encode (default: **+**)
5. Unreserved characters: encode/**don't encode**
6. Reserved characters: **encode**/don't encode
7. Other special characters (neither unreserved nor reserved): **encode**/don't encode

8. Character Percent-Encoding: ISO-8859-1/UTF-8/any other code page/if = ' ' then `*CODEPAGE` (default Natural code page, not default encoding code page!)
9. User-selected character in an X-array of format U, which shall not be percent-encoded according to the above parameters, for example, for the Euro sign character, which is not in the ISO-8859-1 code page, or to prevent a character from percent-encoding.
10. User defined percent-encoding in an X-array of format A, for a user-selected character in the same occurrence of the X-array.

The input parameter for a character string will be in Natural format U. This means the input string may contain all Unicode characters. The output string of the extended function is of format A in the Natural default code page (`*CODEPAGE`). The code page of the percent-encoding can be selected. The UTF-8, ISO-8859-1, percent-encoding of the Euro sign will be done by the `MOVE ENCODED` statement. If an input character does not exist in the target code page used for percent-encoding, the character will not be encoded. This means the character will be returned unchanged in the default Natural code page. If the character does not exist in the default Natural code page either, it will be replaced by that substitution character which is returned by the `MOVE ENCODED` statement. The substitution character will be percent-encoded. This may happen only if the percent-encoding code page is not UTF-8. The last `MOVE ENCODED` error will be returned.

The parameters are optional parameters. If the user does not specify a parameter, the default value will be assumed. If the user specifies an own character translation table, the characters in the table will be percent-encoded according to this table and not according to the other parameters. If the percent-encoding of a character in the user-defined translation table is equal to the character or blank, this character will not be encoded. Thus, single characters from the reserved or unreserved character set can be excluded.

The following functions are available:

- `SAGENCE` - Extended encoding (format U to format A, optional parameters)
- `SAGDECE` - Extended decoding (format A to format U, optional parameters)

SAGENCE - Extended Encoding (Format U to Format A, Optional Parameters)

The function `SAGENCE` percent-encodes a string, using the hexadecimal value of the selected code page (default UTF-8). According to standard RFC3986, reserved characters and characters below US-ASCII `x'7F'`, which are not allowed in a URL, will be percent-encoded. Also, the space and the percent sign (%) will be encoded.

In addition, unreserved characters according to RCF3986 and characters above US-ASCII `x'7F'`, such as German umlauts, will be encoded by this function.

`SAGENCE` needs Natural Unicode support.

Object	Description
SAGENCE	This is the extended encoding function call.
	Parameters:

```
P-DEC-STR-E    (U)
P-RET          (I4)     OPTIONAL  /* 0:    ok
                                  /* else: Natural error returned
                                  /*       by the GIVING clause of
                                  /*       MOVE ENCODED.
                                  /*       This is the error which
                                  /*       comes up when a character
                                  /*       cannot be converted into
                                  /*       the target code page.
/* Error strategy:
/* Step 1: If a character shall be %-encoded and is not available
/* in the code page for %-encoding, the character will not be
/* %-encoded. It will be copied.
/* Step 2: If a character will not be %-encoded but copied from the
/* input format U-variable to a format A-variable (in *CODEPAGE)
/* and the character is not available in *CODEPAGE, a substitution
/* character will be used instead. The substitution character will
/* be %-encoded.
/* The last error will be returned in P-RET.
P-ERR-CHAR     (U1)     OPTIONAL  /* Character causing the error
P-SPACE        (A1)     OPTIONAL  /* '%'  => %20
                                  /* ' '  => ' '
                                  /* else => '+' (default)
P-UNRES        (A1)     OPTIONAL  /* 'E'  => encode
                                  /* else => don't encode (default)
P-RES          (A1)     OPTIONAL  /* 'E'  => encode (default)
                                  /* else => don't encode
P-OTHER        (A1)     OPTIONAL  /* 'E'  => encode (default)
                                  /* else => don't encode
P-CP           (A64)    OPTIONAL  /* IANA name e.g. UTF-8 (default)
                                  /* or ISO-8859-1
/* On mainframe only code page names defined with the macro NTCPAGE
/* in the source module NATCONFG can be used. Other code page names
/* are rejected with a corresponding runtime error.
/*
P-CP-TABLE-CHAR(U1/1:*)  OPTIONAL /* user selected char to be
                                  /* %-encoded, e.g. 'ö' or '/'
P-CP-TABLE-ENC (A12/1:*) OPTIONAL /* user %-encoding
                                  /* e.g. character 'ö'
                                  /*      '%F6'    -> ISO-8859-1
                                  /*      '%C3%B6' -> UTF-8
                                  /* e.g. character '/'
                                  /*      '/'    -> '/' not encoded
                                  /*       although P-RES = 'E'
/* Characters in this table will be encoded according to the
/* specifed %-encoding. If the U12 encoding part is blank (space
/* according to *CODEPAGE) or the P-CP-TABLE-ENC value is equal to
```

Object	Description
	`/* the character, then the character will not be encoded at all.` `/*`
SAGENCEP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGENCEP is optional.
URLX01	**Example program** contained in library SYSEXPG.

	Sample Calls
	
	Default values will be taken:
	
	`#URL-ENC := SAGENCE(<#URL-DEC-U>)`
	
	All possible parameters are specified:
	
	`#URL-ENC := SAGENCE(<#URL-DEC-U,L-RET,L-ERR-CHAR,L-SPACE,L-UNRES, ↩`
	`L-RES,L-OTHER,L-CP,L-CP-TAB-CHAR(*),L-CP-TAB-ENC(*) >)`

SAGDECE - Extended Decoding (Format A to Format U, Optional Parameters)

The function SAGDECE decodes the percent-encodings as provided by the function SAGENCE. If a space character and/or a code page is specified, the values must be the same as specified for encoding.

SAGDECE needs Natural Unicode support.

Object	Description
SAGDECE	This is the extended decoding function call.
	Parameters:

	`1 P-ENC-STR-E    (A)`
	`1 P-RET          (I4)     OPTIONAL  /* 0:    ok`
	`                                     /* else: Natural error returned`
	`                                     /*       by the GIVING clause of`
	`                                     /*       MOVE ENCODED.`
	`                                     /*       This error comes up`
	`                                     /*       when a %-encoded`
	`                                     /*       character cannot be`
	`                                     /*       converted into the`
	`                                     /*       target code page.`
	`  /* The last error will be returned in P-RET.`
	`1 P-ERR-CHAR     (A12)    OPTIONAL  /* Error character %-encoded`
	`1 P-SPACE        (A1)     OPTIONAL  /* ' '  => ' '`
	`                                     /* else => '+' (default)`

Object	Description

	`1 P-CP (A64) OPTIONAL /* IANA name e.g. UTF-8 (default)`
	`/* or ISO-8859-1`
	`/* On mainframe only code page names defined with the macro NTCPAGE`
	`/* in the source module NATCONFG can be used. Other code page names`
	`/* are rejected with a corresponding runtime error.`
	`/*`

SAGDECEP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGDECEP is optional.
URLX01	**Example program** contained in library SYSEXPG.

	Sample Calls
	
	Default values will be taken:
	
	`#URL-DEC-U := SAGDECE(<#URL-ENC>)`
	
	All possible parameters are specified:
	
	`#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,L-ERR-CHAR-DEC,L-SPACE,L-CP>)`

Example Program

Example program contained in library SYSEXPG:

```
** Example 'URLX01': ENCODED-STR := SAGENC(<DECODED-STR>)
************************************************************************
DEFINE DATA
LOCAL
1 SAMPLE-STRING (A72)
/*
1 #URL-DEC      (A) DYNAMIC
1 #URL-ENC      (A) DYNAMIC
/*
1 #URL-DEC-U    (U) DYNAMIC
/*
1 L-RET         (I4)   /* Return code
1 L-ERR-CHAR    (U1)   /* Error character
1 L-ERR-CHAR-DEC(A12)  /* Decoded error character
1 L-SPACE       (A1)   /* '%'  => %20, ' '  => ' ',
                       /* else => '+' (default)
1 L-UNRES       (A1)   /* 'E' => encode, else => don't encode (default)
1 L-RES         (A1)   /* 'E' => encode (default), else => don't encode
1 L-OTHER       (A1)   /* 'E' => encode (default), else => don't encode
1 L-CP          (A64)  /* default *CODEPAGE
```

```
1 L-CP-TAB-CHAR (U1/1:1)
1 L-CP-TAB-ENC  (A12/1:1)
1 L-MSG         (U72)
END-DEFINE
/*
/*
/*
WRITE 'Sample string to be processed:'
/* The string below shall be encoded and decoded again.
/* After decoding it should be unchanged.
SAMPLE-STRING := '"Decoded data!"'
WRITE SAMPLE-STRING (AL=72) /
/*
/* Assign the sample string to the input variable #URL-DEC of the
/* simple encoding function.
#URL-DEC      := SAMPLE-STRING
/*
/* Copycode SAGENCP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGENCP is optional.
INCLUDE SAGENCP
/*
/* SAGENC(<#URL-DEC>) is the simple encoding function call.
/*
/* Function SAGENC %-encodes a string to code page ISO-8859-1.
/* According to standard RFC3986 reserved characters and characters
/* below US-ASCII x'7F' which are not allowed in a URL will be
/* %-encoded.
/* Also the space and the percent sign will be encoded.
/* Unreserved characters according to RCF3986 and characters above
/* US-ASCII x'7F' will not be encoded. If you want to encode such
/* characters, use the extended encoding function.
/*
/* ---- Space                        ' ' -> '+'
/* ---- Percent sign                 '%' -> '%25'
/*
/* Unreserved characters according to RFC3986 (will not be encoded!):
/* ---- Period (fullstop)            '.' -- '%2E'
/* ---- Tilde                        '~' -- '%7E'
/* ---- Hyphen                       '-' -- '%2D'
/* ---- Underscore character         '_' -- '%5F'
/* ---- digits, lower and upper case characters
/* ---- 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
/*
/* Reserved characters according to RFC3986:
/* ---- Exclamation mark             '!' -> '%21'
/* ---- Number sign                  '#' -> '%23'
/* ---- Dollar sign                  '$' -> '%24'
/* ---- Ampersand                    '&' -> '%26'
/* ---- Apostrophe                   ''' -> '%27'
/* ---- Left parenthesis             '(' -> '%28'
```

```
/* ---- Right parenthesis           ')' -> '%29'
/* ---- Asterisk                    '*' -> '%2A'
/* ---- Plus sign                   '+' -> '%2B'
/* ---- Comma                       ',' -> '%2C'
/* ---- Reverse solidus (backslash) '/' -> '%2F'
/* ---- Colon                       ':' -> '%3A'
/* ---- Semi-colon                  ';' -> '%3B'
/* ---- Equals sign                 '=' -> '%3D'
/* ---- Question mark               '?' -> '%3F'
/* ---- Commercial at               '@' -> '%40'
/* ---- Square bracket open         '[' -> '%5B'
/* ---- Square bracket close        ']' -> '%5D'
/*
/* Other characters below x'7F' (US-ASCII) but not allowed in URL
/* ---- Quotation mark              '"' -> '%22'
/* ---- Less than                   '<' -> '%3C'
/* ---- Greater than                '>' -> '%3E'
/* ---- Reverse solidus (backslash) '\' -> '%5C'
/* ---- Accent, Circumflex          '^' -> '%5E'
/* ---- Accent, Grave               '`' -> '%60'
/* ---- Opening brace               '{' -> '%7B'
/* ---- Vertical bar                '|' -> '%7C'
/* ---- Closing brace               '}' -> '%7D'
/*
#URL-ENC := SAGENC(<#URL-DEC>)
/*
/*
WRITE 'Simple function, encoded:'
WRITE #URL-ENC (AL=72)
/*
/* Copycode SAGDECP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGDECP is optional.
INCLUDE SAGDECP
/*
/* SAGDEC(<#URL-ENC>) is the simple decoding function call.
/* It decodes the above described %-encodings.
/*
#URL-DEC := SAGDEC(<#URL-ENC>)
/*
/*
/* The result after encoding and decoding must be equal to the original
/* SAMPLE-STRING.
WRITE 'Simple function, decoded:'
WRITE #URL-DEC (AL=72)
/*
/*
/*
WRITE /
/*
/*
```

```
/*
/* Assign the sample string to the input variable #URL-DEC-U of the
/* enhanced encoding function.
#URL-DEC-U := SAMPLE-STRING
/*
/* Copycode SAGENCEP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGENCEP is optional.
INCLUDE SAGENCEP
/*
/* This is the enhanced encoding function call.
/* The way, characters will be %-encoded dependes on the input
/* parameter of the function.
/* The parameters of the encoding and decoding function are preset
/* with the default values.
/* L-CP-TAB-CHAR(*) and L-CP-TAB-ENC(*) don't have default values.
/* L-CP-TAB-CHAR(1) = 'ä' and L-CP-TAB-ENC(1) = '%C3%A4' will not be
/* used for the sample string '"Decoded data!"'. The string does not
/* contain an 'ä'.
L-SPACE     := '+'           /* encoding and decoding
L-UNRES     := 'D'           /* encoding only
L-RES       := 'E'           /* encoding only
L-OTHER     := 'E'           /* encoding only
L-CP        := 'UTF-8'       /* encoding and decoding
                             /* e.g. ISO-8859-1, UTF-16BE, UTF-32BE
L-CP-TAB-CHAR(1) := 'ä'      /* encoding only
L-CP-TAB-ENC (1) := '%C3%A4' /* encoding only
/*
/* Note that all possible parameters are specified for this sample
/* call.
/* If the default values shall be used and no return code is wanted,
/* all parameters can be omitted, besides the string #URL-DEC-U.
/*
#URL-ENC := SAGENCE(<#URL-DEC-U,L-RET,L-ERR-CHAR,L-SPACE,L-UNRES,
  L-RES,L-OTHER,L-CP,L-CP-TAB-CHAR(*),L-CP-TAB-ENC(*) >)
WRITE 'Extended function, encoded:'
WRITE #URL-ENC (AL=72)
IF L-RET NE 0 THEN
  /* If L-RET = 0, the function worked ok. Else L-RET contains the
  /* Natural error returned by the GIVING clause of MOVE ENCODED.
  /* The error comes up when a character cannot be converted into
  /* the target codepage, e.g. because a character does not exist
  /* in the target codepage.
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
END-IF
/*
/* Copycode SAGDECEP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGDECEP is optional.
```

```
INCLUDE SAGDECEP
/*
/* This is the 1st enhanced decoding function call with 5 parameters.
/* Note that all possible parameters are specified for this sample
/* call.
/* Since the parameters have the default values, the subsequent
/* function calls return the same result although parameters
/* have been omitted.
#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,L-ERR-CHAR-DEC,L-SPACE,L-CP>)
WRITE 'Extended function, decoded:'
WRITE #URL-DEC-U (AL=72)
IF L-RET NE 0 THEN
  /* If L-RET = 0, the function worked ok. Else L-RET contains the
  /* Natural error returned by the GIVING clause of MOVE ENCODED.
  /* The error comes up when a %-encoded character cannot be converted
  /* into the target codepage, e.g. because a character does not exist
  /* in the target codepage.
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
  RESET L-RET
END-IF
/*
/* This is the 2nd enhanced decoding function call with one parameter.
#URL-DEC-U := SAGDECE(<#URL-ENC>)
WRITE #URL-DEC-U (AL=72)
/* L-RET will not be returned
/*
/* This is the 3rd enhanced decoding function call with 3 parameters.
#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,2X,L-CP>)
WRITE #URL-DEC-U (AL=72)
IF L-RET NE 0 THEN
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
  RESET L-RET
END-IF
/*
END
```

Base64 Encoding

This section describes Natural functions which can be used to convert binary data into printable, network-compatible data or vice versa, using Base64 conversion.

Base64 conversion means conversion from format B to format A and back to format B, where 6 (binary) bits will be converted into 8 (alphanumerical) bits; for example, a B3 value will be converted into an A4 value.

Note: Every binary value will be converted into a non-ambiguous alphanumerical value. Re-converting this alphanumerical value again will result in the original binary value. However, this is not the case for most of the format A to format B and back to format A conversions.

The conversion may be used to transfer a `.bmp` file via TCP/IP, or to transfer Natural binary or integer values via the utility protocol.

On Open Systems only: There are 3 modes available: `RFC3548`, `RFC2045` and `NATRPC` (default). `NATRPC` means the conversion is done according the `NATRPC` logic. This is 100% mainframe compatible. `RFC2045` is the default of the CMBASE64 call. `RFC3548` is like `NATRPC`, but alphanumerical bytes which are not needed are filled with an equals sign character (=).

The following functions are available:

- `SAG64BA` - Binary to Alphanumerical Conversion
- `SAG64AB` - Alphanumerical to Binary Conversion

These two functions together provide the same functionality as the Natural application programming interface `USR4210N`, which is delivered in library `SYSEXT`.

SAG64BA - Binary to Alphanumerical Conversion

The function `SAG64BA` converts binary data into printable, network-compatible data, using Base64 encoding.

Object	Description
SAG64BA	This is the binary to alphanumerical format conversion function.
	Parameters:

	`1 PARM-B (B) DYNAMIC BY VALUE`
	`/* Binary source input/target output`
	`1 PARM-RC (I4) OPTIONAL`
	`/* 0: ok`
	`/* Mainframe`
	`/* 1 Source is not numeric`
	`/* 2 Source is not packed`
	`/* 3 Source is not floating point`
	`/* 4 Overflow, source doesn't fit into target`
	`/* 5 Integer overflow`
	`/* 6 Source is not a valid date or time`
	`/* 7 Length error (hex input not even)`
	`/* 8 Target precision is less than source precision`
	`/* 9 Float underflow (result->0)`
	`/* 10 Alpha source contains non-hex characters`
	`/* 20 Invalid function code`
	`/* Open Systems`
	`/* 1 Invalid value for RFC parameter`

Object	Description

	`/* 2 Invalid function code`
	`/* 3 CMBASE64: Overflow, source doesn't fit into`
	`/* target`
	`/* 4 CMBASE64: Non-base64 character found in encoded`
	`/* data`
	`/* 5 CMBASE64: Out of memory`
	`/* 6 CMBASE64: Invalid number of parameters`
	`/* 7 CMBASE64: Invalid parameter type`
	`/* 8 CMBASE64: Invalid parameter length`
	`/* 9 CMBASE64: Invalid function code`
	`/* 10 CMBASE64: Unkown return code`
	`1 PARM-ERRTXT (A72) OPTIONAL`
	`/* blank, if ok no error`
	`/* else error text`
	`1 PARM-RFC (B1) OPTIONAL`
	`/* OS only, not used for MF`
	`/* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC;`

SAG64BAP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAG64BAP is optional.

B64X01	**Example program** contained in library SYSEXPG.
	
	Default values will be taken:
	
	`PARM-A := SAG64BA(<PARM-B>)`
	
	All possible parameters are specified (PARM-RFC does not apply to mainframe):
	
	`PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>)`

SAG64AB - Alphanumerical to Binary Conversion

The function SAG64AB converts printable, network-compatible data into binary data, using Base64 encoding.

Object	Description
SAG64AB	This is the alphanumerical to binary format conversion function.
	Parameters:

Object	Description
	1 PARM-A (A) /* Alpha source input/target output 1 PARM-RC (I4) OPTIONAL /* 0: ok /* Mainframe /* 1 Source is not numeric /* 2 Source is not packed /* 3 Source is not floating point /* 4 Overflow, source doesn't fit into target /* 5 Integer overflow /* 6 Source is not a valid date or time /* 7 Length error (hex input not even) /* 8 Target precision is less than source precision /* 9 Float underflow (result->0) /* 10 Alpha source contains non-hex characters /* 20 Invalid function code /* Open Systems /* 1 Invalid value for RFC parameter /* 2 Invalid function code /* 3 CMBASE64: Overflow, source doesn't fit into /* target /* 4 CMBASE64: Non-base64 character found in encoded /* data /* 5 CMBASE64: Out of memory /* 6 CMBASE64: Invalid number of parameters /* 7 CMBASE64: Invalid parameter type /* 8 CMBASE64: Invalid parameter length /* 9 CMBASE64: Invalid function code /* 10 CMBASE64: Unkown return code 1 PARM-ERRTXT (A72) OPTIONAL /* blank, if ok no error /* else error text 1 PARM-RFC (B1) OPTIONAL /* OS only, not used for MF /* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC;
SAG64ABP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAG64ABP is optional.
B64X01	**Example program** contained in library SYSEXPG. Default values will be taken:

Object	Description

	`PARM-B := SAG64AB(<PARM-A>)`
	
	All possible parameters are specified (`PARM-RFC` does not apply to mainframe):
	
	`PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)`

Example Program

Example program `B64X01` contained in library `SYSEXPG`:

```
** Example 'B64X01': BASE64-A-STR := SAG64BA(<BASE64-B-STR>)
************************************************************************
* Function ......... Convert binary data into printable,
*                    network-compatible data or vice versa using
*                    Base64 encoding.
*
*                    Base64 encoding means (B) -> (A) -> (B),
*                    where 6 (binary) bits will be encoded into 8
*                    (alpha) bits, e.g a (B3) value will be encoded
*                    into a (A4) value.
*
*                    Note: Every binary value will be encoded into
*                    a non-ambiguous alpha value. Re-encoding this
*                    alpha value again will result in the original
*                    binary value. However, this is not the case with
*                    most of the (A) -> (B) -> (A) encodings.
*
*                    The encoding may be used to transfer a .bmp
*                    file via TCP/IP, or to transfer Natural binary or
*                    integer values via the utility protocol.
*
*                    Open Systems only:
*                    On Open Systems, there are 3 modes:
*                    RFC3548, RFC2045 and NATRPC (default).
*                    NATRPC means the encoding follows
*                    the NATRPC logic. This is 100% MF compatible.
*                    RFC2045 is the default of the CMBASE64 call.
*                    RFC3548 is like NATRPC, but alpha bytes not
*                    needed are filled with '='.
*
DEFINE DATA
LOCAL
1 FUNCTION        (A2)
                  /* 'AB' Alpha to binary encoding
                  /* 'BA' Binary to alpha encoding
1 PARM-RC         (I4)
                  /* 0:    ok
                  /* Mainframe
```

```
                  /* 1  Source is not numeric
                  /* 2  Source is not packed
                  /* 3  Source is not floating point
                  /* 4  Overflow, source doesn't fit into target
                  /* 5  Integer overflow
                  /* 6  Source is not a valid date or time
                  /* 7  Length error (hex input not even)
                  /* 8  Target precision is less than source precision
                  /* 9  Float underflow (result->0)
                  /* 10 Alpha source contains non-hex characters
                  /* 20 Invalid function code
                  /* Open Systems
                  /* 1  Invalid value for RFC parameter
                  /* 2  Invalid function code
                  /* 3  CMBASE64: Overflow, source doesn't fit into
                  /*              target
                  /* 4  CMBASE64: Non-base64 character found in encoded
                  /*              data
                  /* 5  CMBASE64: Out of memory
                  /* 6  CMBASE64: Inalid number of parameters
                  /* 7  CMBASE64: Invalid parameter type
                  /* 8  CMBASE64: Invalid parameter length
                  /* 9  CMBASE64: Invalid function code
                  /* 10 CMBASE64: Unkown return code
1 PARM-ERRTXT     (A72)
                  /* blank, if ok no error
                  /* else error text
1 PARM-A          (A)   DYNAMIC
                  /* Alpha source input/target output
1 PARM-B          (B)   DYNAMIC
*                 /* Binary source input/target output
1 PARM-RFC        (B1)
                  /* OS only, not used for MF
                  /* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC;
/*
1 #BACKUP-A       (A) DYNAMIC
1 #BACKUP-B       (B) DYNAMIC
END-DEFINE
/*
/*
SET KEY ALL
/*
/* Copycode SAG64BAP and SAG64ABP containing the prototype definition
/* is used at compilation time only in order to determine the type of
/* the return variable for function call reference and to check the
/* parameters, if this is desired. SAG64BAP and SAG64ABP are optional.
INCLUDE SAG64BAP
INCLUDE SAG64ABP
/*
REPEAT
  RESET PARM-A PARM-B
  REDUCE DYNAMIC PARM-A TO 0
```

```
REDUCE DYNAMIC PARM-B TO 0
FUNCTION := 'BA'
PARM-B := H'0123456789ABCDEF'
INPUT (AD=MIL IP=OFF CD=NE) WITH TEXT PARM-ERRTXT
 // 10T 'Base64 Encoding:' (YEI)
  / 10T '-' (19) (YEI) /
  / 10T 'Function (BA,AB) ..' (TU) FUNCTION (AD=T)
  / 10T 'Alpha In/Output ...' (TU) PARM-A (AL=30)
  / 10T 'Binary In/Output ..' (TU) PARM-B (EM=HHHHHHHH)
  / 10T 'Response ..........' (TU) PARM-RC (AD=OD CD=TU)
  / PARM-ERRTXT (AD=OD CD=TU)
RESET PARM-ERRTXT
IF *PF-KEY NE 'ENTR'
  ESCAPE BOTTOM
END-IF
/*
RESET #BACKUP-A #BACKUP-B
REDUCE DYNAMIC #BACKUP-A TO 0
REDUCE DYNAMIC #BACKUP-B TO 0
#BACKUP-A := PARM-A
#BACKUP-B := PARM-B
/*
IF FUNCTION = 'BA'
  /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
  /* Parameter PARM-RFC does not apply to mainframe
  /* PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>)
  PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT>)
  /* PARM-A := SAG64BA(<PARM-B,PARM-RC>)
  /* PARM-A := SAG64BA(<PARM-B>)
ELSE
  /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
  /* Parameter PARM-RFC does not apply to mainframe
  /* PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)
  PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT>)
  /* PARM-B := SAG64AB(<PARM-A,PARM-RC>)
  /* PARM-B := SAG64AB(<PARM-A>)
END-IF
/*
IF PARM-RC NE 0 THEN
  WRITE 'Encoding' FUNCTION
  WRITE NOTITLE PARM-ERRTXT
ELSE
  IF FUNCTION = 'BA' THEN
    WRITE 'Binary -> Alpha'
    WRITE '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
      / '=' PARM-A (AL=50)
    RESET PARM-B
    REDUCE DYNAMIC PARM-B TO 0
    FUNCTION := 'AB'
  ELSE
    WRITE 'Alpha -> Binary'
    WRITE '=' PARM-A (AL=50) /
```

```
        '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
      RESET PARM-A
      REDUCE DYNAMIC PARM-A TO 0
      FUNCTION := 'BA'
    END-IF
    /*
    IF FUNCTION = 'BA'
      /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
      /* Parameter PARM-RFC does not apply to mainframe
      /* PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>)
      PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT>)
      /* PARM-A := SAG64BA(<PARM-B,PARM-RC>)
      /* PARM-A := SAG64BA(<PARM-B>)
    ELSE
      /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
      /* Parameter PARM-RFC does not apply to mainframe
      /* PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)
      PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT>)
      /* PARM-B := SAG64AB(<PARM-A,PARM-RC>)
      /* PARM-B := SAG64AB(<PARM-A>)
    END-IF
    IF PARM-RC NE 0 THEN
      WRITE 'Encoding' FUNCTION
      WRITE NOTITLE PARM-ERRTXT
    ELSE
      IF FUNCTION = 'BA' THEN
        WRITE 'Binary -> Alpha'
        WRITE '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
          / '=' PARM-A (AL=50)
        IF PARM-A = #BACKUP-A THEN
          WRITE '******** Encoding successful ********'
        ELSE
          WRITE '******** Value changed by encoding ********'
        END-IF
      ELSE
        WRITE 'Alpha -> Binary'
        WRITE '=' PARM-A (AL=50) /
          '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
        IF PARM-B = #BACKUP-B THEN
          WRITE '******** Encoding successful ********'
        ELSE
          WRITE '******** Value changed by encoding ********'
        END-IF
      END-IF
    END-IF
  END-IF
END-REPEAT
END
```