

Natural for Mainframes

System Functions

Version 4.2.6 for Mainframes (Update)

February 2010

This document applies to Natural Version 4.2.6 for Mainframes (Update).

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

Copyright © 1979-2010 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

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

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Table of Contents

1 System Functions	1
2 Natural System Functions for Use in Processing Loops	3
Using System Functions in Processing Loops	4
AVER(r)(field)	6
COUNT(r)(field)	6
MAX(r)(field)	6
MIN(r)(field)	7
NAVER(r)(field)	7
NCOUNT(r)(field)	7
NMIN(r)(field)	8
OLD(r)(field)	8
SUM(r)(field)	8
TOTAL(r)(field)	9
Examples	9
3 Mathematical Functions	15
4 Miscellaneous Functions	19
5 POS - Field Identification Function	21
6 RET - Return Code Function	23
7 SORTKEY - Sort-Key Function	25
Index	29




1 System Functions

This documentation describes various Natural “built-in” functions for use in certain statements.



Note: As of Natural Version 6.2 for Windows and UNIX, Version 6.3 for OpenVMS and Version 4.2 for Mainframes, all new system functions are preceded by an asterisk (*) to avoid naming conflicts with, for example, user-defined variables in existing applications.

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 Functions	Describes the system functions which are supported in arithmetic processing statements and in logical condition criteria.
	Miscellaneous Functions	Describes various system functions for field identification, receiving return code from a non-Natural program called via a <code>CALL</code> statement, converting “incorrectly sorted” characters.

See also:

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

2 Natural System Functions for Use in Processing Loops

■ Using System Functions in Processing Loops	4
■ AVER(r)(field)	6
■ COUNT(r)(field)	6
■ MAX(r)(field)	6
■ MIN(r)(field)	7
■ NAVER(r)(field)	7
■ NCOUNT(r)(field)	7
■ NMIN(r)(field)	8
■ OLD(r)(field)	8
■ SUM(r)(field)	8
■ TOTAL(r)(field)	9
■ Examples	9

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

Using System Functions in Processing Loops

- Specification/Evaluation
- Use in SORT GIVE 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
- 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, <code>AVER(field)</code> 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(<i>field</i>) 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 <code>N</code> , <code>SUM(field)</code> will be of format <code>P</code> (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

[illegible]

```

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 20200	42200
ROLLING	ALBUQUERQUE	34000 31200	65200
FREEMAN	ALBUQUERQUE	34000 31200	65200
LINCOLN	ALBUQUERQUE	41000 37700	78700
AVERAGE:		32750	
AVERAGE CUMULATIVE:			62825

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

```

** Example 'AEDEXIS': 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 ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY 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 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

3 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
$ABS(field)$	same as <i>field</i>	Absolute value of <i>field</i> .
$ATN(field)$	F8	Arc tangent of <i>field</i> .
$COS(field)$	F8	Cosine of <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , $COS(field)$ will be "1".
$EXP(field)$	F8	Exponentiation of exponent <i>field</i> to base e, that is, e^{field} , where e is Euler's number.
$FRAC(field)$	same as <i>field</i>	Fractional part of <i>field</i> .
$INT(field)$	same as <i>field</i>	Integer part of <i>field</i> .
$LOG(field)$	F8	Natural logarithm of <i>field</i> .
$SGN(field)$	same as <i>field</i>	Sign of <i>field</i> (-1, 0, +1).
$SIN(field)$	F8	Sine of <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , $SIN(field)$ will be "0".
$SQRT(field)$	(*)	Square root of <i>field</i> . 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 <i>field</i> . If the value of the <i>field</i> is equal to or greater than 10^{17} , $TAN(field)$ will be "0".

Function	Format/Length	Explanation
VAL(<i>field</i>)	same as target field	<p>Extract numeric value from an alphanumeric <i>field</i>. The content of the <i>field</i> must be the alphanumeric (code page or Unicode) 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.</p> <p>If the target field is not long enough, decimal digits will be truncated (see also <i>Field Truncation and Field Rounding</i> in the section <i>Rules for Arithmetic Assignment</i> of the <i>Programming Guide</i>).</p>

* These functions are evaluated as follows:

- 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 N_n.7 or P_n.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 (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

```


4

Miscellaneous Functions

The following topics are covered:

- **POS - Field Identification Function**
- **RET - Return Code Function**
- **SORTKEY - Sort-Key Function**

5

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
```



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 expandable) 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.

See also

- *Dialog Design, Field Sensitive Processing and Simplifying Programming in the Programming Guide.*
- *POS22 - Version 2.2 Algorithm for POS System Function in the Parameter Reference.*

6 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
...
```


7

SORTKEY - Sort-Key Function

SORTKEY (<i>character-string</i>)

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 `NATUSKnn` 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.

See *User Exit for Computation of Sort Keys* for more details on the calling conventions for `SORTKEY` 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 "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
```


Index

A

ABS
 system function, 15
ATN
 system function, 15

C

COS
 system function, 15

E

EXP
 system function, 15

F

FRAC
 system function, 15

I

INT
 system function, 15

L

LOG
 system function, 15

S

SGN
 system function, 15
SIN
 system function, 15
SQRT
 system function, 15
system functions, 1

T

TAN
 system function, 15

V

VAL
 system function, 16

