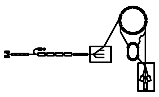

The NODAL Interpreter at GSI

L. Hechler, H. Hübner, U. Krause

GSI, Gesellschaft für Schwerionenforschung mbH
Planckstraße 1, D-64291 Darmstadt
Tel.: 06159/71-0

Issued: 25. Juli 2002
Key: U-NOD-02
Printed: 17. September 2014

This paper is designed to summarize all the commands and functions of the NODAL interpreter at GSI in one document. The commands and functions are explained in a short and formalized manner to achieve an overview for the user. For a more precise information about NODAL we recommend the three documentations [1], [2] and [3]. Control system functions are explained in detail in [5].

Document Revision History		
Date	Name	Comment
23. Feb. 1989	L. Hechler	Creation of document
1. Mar. 1989	L. Hechler	Description of keyboard commands added
10. Mar. 1989	L. Hechler	Resident function UPGRADE added
24. Apr. 1989	L. Hechler	Description of file-i/o functions added
29. May 1989	L. Hechler	Operators and their precedence rules added and new organisation of the document
17. Jul. 1989	U. Krause	Complete new database functions
19. Jul. 1989	L. Hechler	Last corrected revision, ready to publish
25. Jul. 2002	L. Hechler	Adapted for LaTeX2E and WWW

Inhaltsverzeichnis

1 Invoking NODAL

To invoke the NODAL interpreter use the following command without parameters:

```
NODAL [qualifier[qualifier]]
```

Qualifiers are

/CAPITALIZE
: NODAL allows lower case letters. To stay compatible with older versions of NODAL, the qualifier causes NODAL to capitalize all letters.

  The abbreviated version is /C.

/LONGCOMMAND
: After execution of a NODAL line, NODAL cuts all commands to their shortest possible abbreviation. The qualifier causes NODAL to write out the commands in their full length.

  CAUTION: After execution, NODAL lines may grow longer than the original line. The line is cut, if it exceeds the maximum length of 80 characters.

  The abbreviated version is /L.

2 Syntax

2.1 Representation

The description of the syntax of the NODAL language follows approximatly the Extended Backus Naur Formalism (EBNF).
Metasymbols to describe the syntax are:

`[ ]` Optional terms are enclosed in square brackets. They may appear in a NODAL command or not.

`{ }` Optional terms that may appear zero, one or more times are enclosed in braces.

`( )` Parantheses may be used to group terms and factors.

`!` If an expression may consist of one term or, alternativly, of another term, the terms are seperated by the exclamation-mark (alternation).

`' '` To distinguish between symbols and metasymbols, symbols are enclosed in single quotation marks.

2.2 Symbols

`Identifier` Identifiers are names denoting variables and functions. They may contain up to 8 characters. The first character must be a letter, it may be followed by any combination of letters, digits, underscores, dots and colons ('_', '.', ':').

`Expression` An expression is composed of operands and operators. It is evaluated by applying the operands on the operators in the correct order. Parts of expressions which should be an expression in its own right may be enclosed in parantheses. Additional rules apply:

- Every variable in an expresion should exist.
- Two operators must never be written side by side.
- Two operands must never be written side by side.
- Precedence rules determine the order of evaluation.
- Parantheses may be used to force precedence.

`Concatenation` A concatenation is a list of strings and expressions which may be preceded by a format definition. Concatenations are evaluated by joining all elements to one resultstring. Before combining the elements all expressions are evaluated and their result transfered to strings, using the format directives given. Format specifications may be used the way they are described in the NODAL command TYPE.

The total length of the result of a concatenation should not exceed 80 characters.

2.3 NODAL syntax in brief

```
nodalline    ::= [linenumber] nodalcommand ! nodalline ';' nodalcommand
nodalcommand ::= command [command specific syntax] [';']
linenumber   ::= number in the range 0.01 to 99.99
                 (linenumbers nn.00 are not allowed)
```

If you type a NODAL line on the Keyboard, you select implicit the execution mode on the interpreter by specifying the linenumber or not:

- Without linenumber
  NODAL works in the immediate or command mode. The NODAL command is executed immediately (e.g.: TYPE 3*3.568).
- With linenumber
  NODAL works in the line or program mode. The NODAL line is stored internal for later execution started by a RUN command (e.g.: 10.05 TYPE 3*3.568).

For a more detailed information, see [3].
Lowercase letters are accepted by NODAL. They are not distinguished from their uppercase equivalent. Identifiers are stored as they are written at the first appearance during execution.

3 Operators

3.1 Arithmetic Operators

NODAL includes the following operators to form an arithmetic expression.

```
+    addition
-    subtraction
*    multiplication
/    division
^    exponentiation
```

3.2 Relational Operators

NODAL includes the following operators to form a logical expression.

```
<    less than
<=   less than or equal
=    equal
<!   not equal
>=   greater than or equal
>    greater than
```

3.3 Operators Precedens Rules

When evaluating expressions with more than one operator, NODAL follows precedens rules to determine the order of the evaluation. The operation with the highest priority is evaluated first. Operators on the same level of priority are evaluated from left to right.

Operator	Priority
()	highest
^	.
*	.
/	.
-	.
+	.
< <= = <> > >=	lowest

4 Commands

At the present time the following commands are implemented in NODAL at GSI (for further information, see [1, 2]).

4.1 ASK

Description:
Input numeric values from the terminal.

Type:
NODAL Command

Syntax:
`A[SK] ['"'string'"'] ' ' num_variable {',' num_variable}`

Parameter:

num_variable
The asked numeric values are assigned to these variables.

Example:

```
>ASK "Give Current in Ampere" CURR
>A "give first 3 elements" e(1),e(2),e(3)
```

4.2 $ASK

Description:
Input string values from the terminal.

Type:
NODAL Command

Syntax:
`$A[SK] ['"'string'"'] ' ' string_variable {',' string_variable}`

Parameter:

num_variable
The asked string values are assigned to these variables.

Example:

```
>$A "Name of Device" NOMEN
>$ASK PROPERTY PCLASS
```

4.3 CALL

Description:
Call a procedure, a function or perform a Data Module Call.

Type:
NODAL Command

Syntax:
`C[ALL] name['('parameter {',' parameter}')']`

Parameter:

name
: Name of the procedure, function or Data Module.

parameter
: Depends on the called subroutine, but maximum 8 parameters.

Note:
See DEFINE for more information on definition of parameter exchange.

Example:

```
>% perform a Data Module Call
>DI-I CON(4)
>CALL TK9QD21("R", CON, CONSTANT)
```

4.4 DEFINE

Description:
Definition of NODAL procedures or functions.

Type:
NODAL Command

Syntax:

```
DE[FINE]-F[UNCTION] numeric_fct['('parameter {',' parameter}')']
DE[FINE]-S[TRING] stringfunction['('parameter {',' parameter}')']
DE[FINE]-C[ALL] procedure['('parameter {',' parameter}')']
```

Parameter:

numeric function
: Defined function returns a numeric value.

stringfunction
: Defined function returns a stringvalue.

procedure
: Defined procedure returns no value. The exchange of data is only through parameters.

parameter

'V'-variable
: Variable is a value parameter.

'R'-variable
: Variable is a reference parameter.

'S'-variable
: Variable is a string parameter.

Note:
For a detailed information about defining and working with defined functions, see chapter 9 of [1].

4.5 DIMENS

Description:
Create an array of numbers or strings.

Type:
NODAL Command

Syntax:
```
DI[MENS]-I[NT] name'('number [',' number]')'
DI[MENS] name'('number [',' number]')'
DI[MENS]-S[TR] name
```

Parameter:
The command appendix `-I[NT]` denotes an integer array, whereas `-S[TR]` denotes a string array. If the command appendix is omitted, the command dimensions a floating point array.

name
The desired name of the array variable

number
The number of elements in the array. The elements 1... <number> can be accessed.

Note:
Two dimensional arrays are available for float and integer arrays only.
Only single dimensional string arrays are available. For string arrays, the number of elements is not declared in the DIMENS command.

Example:

```
>SE SIZE=10
>SE ELEMENT=7
>DIM-I ARRAY(SIZE)
>DIM-I ARRAY_2(3,9)
>SE ARRAY(6) = 1
>SE ARRAY(ELEMENT) = 2
>SE ARRAY_2(1,1) = ARRAY(7)
```

4.6 DO

Description:
Execute a NODAL group or line.

Type:
NODAL Command

Syntax:
```
DO expression {' !' expression} [' !']
```

Parameter:

expression
NODAL line or group number.

! (alternation)
If an error occurs in the line or group specified before the alternation the line or group specified behind the alternation is executed instead. If no line or group is specified behind the alternation the NODAL program will continue behind the DO command as if no error has occured.

Example:

```
>% simple example
>10.10 SE A = 0
>10.20 SE B = 4711
>10.30 RET
>DO 10.10
>DO 10
>% example with error handling
>10.1 T 1/0
>20.1 T "The result is undefined"
>DO 10 !20
The result is undefined
```

4.7 $DO

Description:
Execute a concatenation.

Type:
NODAL Command

Syntax:
`$D[O] concatenation`

Parameter:

concatenation
The concatenation (string) to execute.

Example:

```
>$SE str = 'TYPE str; SE a = 4711'
>$DO str
TYPE str; SE a = 4711
>TY a
    4711
```

4.8 EDIT

Description:
Invoke the NODAL Screen Editor for VT220-like terminals.

Type:
NODAL Command

Syntax:
`ED[IT]`

Parameter:

Note:
All edited lines longer than 79 characters (inclusive line number) are truncated.

4.9 END

Description:
End execution of a NODAL program.

Type:
NODAL Command

Syntax:
`EN[D]`

Parameter:

4.10 ERASE

Description:
Erase NODAL program or variables.

Type:
NODAL Command

Syntax:
`ER[ASE] [specification]`

Parameter:

specification
A list of specifiers separated by blanks. Specifiers may be the following, if none is specified, ALLV is default:

`identifier`	A variable, pattern variable, NODAL defined function or an array.
`line`	A line number of the current NODAL program.
`group`	A group of the current NODAL program.
`'ALL'`	Is equivalent to ALLP plus ALLV.
`'ALLD'`	Erases all NODAL defined functions.
`'ALLP'`	Erases all lines of the current NODAL program.
`'ALLV'`	Erases all variables.

Example:

```
>10.10 SE A = 0
>10.20 SE B = 4711
>10.30 RET
>ERASE A
>ERASE 10.20
```

4.11 FOR

Description:
Execute a FOR loop.

Type:
NODAL Command

Syntax:
`F[OR] num-variable '=' expr1',' [expr2','] expr3 [';' statements]`

Parameter:

num-variable
: The index which runs from expr1 to expr3.

expr1
: The start index.

expr2
: The index incrementing value. If it is omitted, the default value is 1.

expr3
: The end index.

statements
: The body of the loop.

Example:

```
>DI-I A(20)
>SE FROM=0
>SE STEP=2
>SE TO=20
>F N=FROM,STEP,TO; SE A(N)=N;      % N = (0, 2, 4, ... 18, 20)
```

4.12 GOTO

Description:
: Go to NODAL line.

Type:
: NODAL Command

Syntax:
: `G[OTO] expression`

Parameter:

expression
: The NODAL line to go to. If the fractional part is zero, execution continues at the first line in the group.

Example:

```
>10.10 SE A = 0
>10.20 SE B = 4711
>10.30 RET
>G 10.2
>G 10
```

4.13 IF

Description:
: Conditional arithmetic or logic branch.

Type:
: NODAL Command

Syntax:

```
IF '('arith_expr')' line [',' line [',' line]]; [statements]
IF log_expr ['OR' log_expr]; [statements]
```

Parameter:

arithmetic expression
: If the arithmetic expression is negativ, control is passed to the first line. If it is zero, control is passed to the second line. If it is positive, to the third line. If there is no line corresponding to the sign of the arithmetic expression, control is passed to the statements behind the IF command.

logic expression
: If the condition of the logical expression is true, control is passed to the statements behind the IF command. Logical expression is of the form

  `<arith. expr> <logical operator> <arith. expr.>`.

  Logical operators are:

```
>   greater than
>=  greater than or equal
=   equal
<>  not equal
<=  less than or equal
<   less than
```

OR
: Constitutes the logical OR of several logic expressions.

line
: Valid NODAL line number.

statements
: One or more NODAL statement(s).

Note:

- The expression of the logic IF must not begin with an bracket to avoid confusion with the arithmetic IF.
- The arithmetic expression is treated to be zero if its value is inside the range $\pm 5 * 10^{-16}$.
- An effective logical AND is obtained by cascading several IF commands (see the example).

Example:

```
>% arithmetic IF
>1.1 T "negative"; END
>1.2 T "zero"; END
>1.3 T "positiv"; END
>SE A=0
>IF (A) 1.1, 1.2, 1.3
zero
>% logic IF
>SE B=1
>IF A=0; IF B=1; T "true";    % IF a=0 AND b=1 THEN true
true
```

4.14 $IF

Description:
Conditional arithmetic or logic branch with string arguments. It can be used just as the IF command described above.

Type:
NODAL Command

Syntax:
```
$I[F] '('conc '-' conc')' line [','line [','line]]; [statements]
$I[F] log_stringexpr ['OR' log_stringexpr]; [statements]
```

Parameter:

conc
Concatenation of strings.

logic stringexpression
Lexical character by character comparison of strings on the basis of ASCII characters where
`"." < "0" < "9" < ":" < "A" < "Z" < "_" < "a" < "z"`.

Example:

```
>$IF A = "YES"; DO 20
>$IF (A-"YES") 1.1, 1.2, 1.3
```

4.15 LDEF

Description:
Load Defined Functions from file.

Type:
NODAL Command

Syntax:
```
LDEF filename {' 'filename}
```

Parameter:

filename
The name of the file where the Defined Functions are stored.

4.16 LIST

Description:
List NODAL program, function and data information.

Type:
NODAL Command

Syntax:
```
LI[ST] [specification]
```

Parameter:

specification
A list of specifiers separated by blanks. Specifiers may be the following, if none is specified, ALLP is default:

`DataElementName`	List informationen about the data element.
`line`	A line number of the current NODAL program.
`group`	A group of the current NODAL program.
`'ALLD'`	List all 'Defined Functions'.
`'ALLP'`	List whole actual NODAL program.
`'ALLR'`	List all resident functions with their type and formal parameters.
`'ALLV'`	List all user defined variables and arrays.
`'ALL'`	Is the combination of ALLP plus ALLV

Note:
See also LISD, LISR, LISV.

4.17 LOAD

Description:
Load the contents of a file into the working area. The file may contain NODAL program or data. Existing lines or variables in the working area are only replaced if the file containes the same lines or variables.

Type:
NODAL Command

Syntax:
`LO[AD] filename`

Parameter:

filename
The name of the file to load.

Note:
See also OVERLA.

4.18 $MATCH

Description:
Match a concatenation with a pattern. Replace the pattern with a string if the match succeeds. Go to a specified NODAL line if the match fails.

Type:
NODAL Command

Syntax:
`$M[ATCH] concat_1 pattern [' ='concat_2] [' :'expression]`

Parameter:

concatenation 1
The string in witch to search.

pattern
The pattern to search for in the concatenation 1.

concatenation 2
: If the pattern matches replace it with the concatenation 2.

expression
: Go to this NODAL line or group if the match fails.

Note:
: For a more detailed information, see [3, Section 10.5ff].

Example:

```
>1.10 $SE a = 'abcd'
>1.20 $MATCH a 'cd' :1.90; T "pattern matched"
>1.30 $MATCH a 'xy' :1.90; T "pattern matched"
>1.90 T ! "match failed"
>RUN
pattern matched
match failed
>
```

4.19 ?OFF

Description:
: Switch off trace option.

Type:
: NODAL Command

Syntax:
: `?OF[F]`

Parameter:

Note:
: See ?ON for details.

4.20 OLD

Description:
: Clear all program and data in the working area and load the contents of a file into the working area. The file may contain NODAL program or data.

Type:
: NODAL Command

Syntax:
: `OL[D] filename`

Parameter:

filename
: The name of the file to load.

4.21 ?ON

Description:
Switch on the trace option. This causes each line of the NODAL program to be typed before execution.

Type:
NODAL Command

Syntax:
`?ON`

Parameter:

4.22 OPEN

Description:
Clear the working area, get a NODAL defined function or procedure into the working area and delete the defined function.

Type:
NODAL Command

Syntax:
`OP[EN] defined-function`

Parameter:

defined-function
The name of the defined function or procedure to open.

4.23 OVERLA

Description:
Load and execute a NODAL subprogram without disturbing the main NODAL program. The subprogram may use the same line numbers and/or variables as the main program without interference. The subprogram disappears after execution.

Type:
NODAL Command

Syntax:
`OV[ERLA] filename`

Parameter:

filename
The desired file with the NODAL subprogram.

Example:

```
>% main program
>1.10 SE A=1;             % set variable A in main
>1.20 OVERLA SUBPROG;     % execute subprogram
>1.30 TY A;               % result will be 1
>% subprogram SUBPROG
>1.10 SE A=9;             % set variable A in subprogram
>1.20 TY A;               % result will be 9
>1.30 END;                % subprogram
```

4.24 $PATTE

Description:
Create a pattern.

Type:
NODAL Command

Syntax:
`$P[ATTE] pat [(' .'id) ! (' $'id)] ['!'] {pat [(' .'id) ! (' $'id)]}`

Parameter:

pattern
A string which may include concatenations and/or alternations.

' .' (dot)
The dot denotes an immediate assignment field. If the whole pattern matches, the matched part is copied into the string variable following the dot.

' $' (dollar)
The dollar also denotes an immediate assignment field. Here, only the part of the pattern belonging to this specific assignment field has to match to copy the matched part into the string variable following the dollar.

identifier
The string variable.

Note:
For a more detailed information, see [3, Section 10.5ff].

Example:

```
>$PAT P1 = "AB" ! "CD";                    %matches 'AB' or 'CD'
>$PAT P2 = "A" ("B" ! "C") "D";            %matches 'ABD' or 'ACD'
>
>% matches if a string contains 'XABD' or 'XACD'. Only then
>% V1 becomes 'X' and V2 becomes 'ABD' or 'ACD'.
>$PAT P3 = "X" .V1 P2 .V2
>
>% matches if a string contains 'XABD' or 'XACD'.
>% V1 becomes 'X' if the string contains 'X'
>% V2 becomes 'ABD' or 'ACD' if the string contains 'ABD' or 'ACD'
>$PAT P4 = "X" $V1 P2 $V2
```

4.25 QUIT

Description:
Leave the NODAL interpreter and return to DCL.

Type:
NODAL Command

Syntax:
`Q[UIT]`

Parameter:

4.26 RETURN

Description:
Return from a NODAL program group to the calling DO.

Type:
NODAL Command

Syntax:
`RET[URN]`

Parameter:

Note:
May be used to return before the last line in a group is reached.

4.27 ROF

Description:
Break out from the body of a FOR loop and continue on next line.

Type:
NODAL Command

Syntax:
`RO[F]`

Parameter:

4.28 RUN

Description:
Start execution of a NODAL program.

Type:
NODAL Command

Syntax:
`RU[N] ['['number']'] [filename]`

Parameter:

number
The line or group number of the NODAL file.

filename
The desired filename. Before loading the program, any previous contents of the working area is deleted.

Example:

```
>RUN MAGTEST;       % runs the program named MAGTEST
>RU [10];           % runs from group 10 of a previously loaded program
>RU [1.5] TST;      % runs the program TST from line 1.5
```

4.29 SAVE

Description:
Save program and/or data for later reference.

Type:
NODAL Command

Syntax:
`SA[VE] filename [specification]`

Parameter:

filename
The desired filename.

specification
A list of specifiers separated by blanks. Specifiers may be the following, if none is specified, ALLP is default:

`identifier`	A variable, pattern variable, NODAL defined function or an array.
`line`	A line number of the current NODAL program.
`group`	A group of the current NODAL program.
`'ALL'`	Is equivalent to ALLP plus ALLV.
`'ALLD'`	Stores all NODAL defined functions.
`'ALLP'`	Stores all lines of the current NODAL program.
`'ALLV'`	Stores all variables.

Note:
Specification `'ALL'` does not save defined functions!

Example:

```
>% save group 25 in file TEST
>SAVE TEST 25
>% save lines 3.1 and 4.1 and all defined functions in file TST
>SA TST 3.1 4.1 ALLD
```

4.30 SDEF

Description:
Save Defined Functions on file.

Type:
NODAL Command

Syntax:
`SDEF filename`

Parameter:

filename
The name of the file where the Defined Functions are to be stored.

Note:
SDEF is the abbreviated command of SAVE ALLD.

4.31 SET

Description:
Assign a value to a RO- or RW-function or a numeric variable.

Type:
NODAL Command

Syntax:
`SE[T] num-var ! function '=' [format-control] expression`

Parameter:

format-control

`[` Denotes an octal number.

`[[` Denotes an hexadecimal number.

4.32 $SET

Description:
Assign a concatenation to a RO- or RW-Stringfunction or a string variable.

Type:
NODAL Command

Syntax:
`$S[ET] stringvar ! stringfct '=' [format-control] concatenation`

Parameter:

format-control

`'\'n` n is the value of the ASCII character.

`'&'n` Assign n spaces.

Where n could be an octal (`n := '['figure(s)`), a decimal (`n := figure(s)`) or a hexadecimal (`n := '[['figure(s)`) number.

concatenation
A concatenation of strings.

Example:

```
>$SE EX = "Example:"
>$SE A = "Fred's friend said "
>$SE B = '"Where are you"'
>T EX ! A B
Example:
Fred's friend said "Where are you"
>$S STR = \[[41
>T STR
A
>
```

4.33 TYPE

Description:
Perform an output operation on the screen.

Type:
NODAL Command

Syntax:
```
TYPE type-element {',' ! ' ' type-element}
type-element := [form-control] (expression ! concatenation)
```

Parameter:
Form-controls may be the following, if nothing is specified, `%11.04` is default for expressions:

`'!'`	Start a new line
`'%,'`	Print all digits (exponential for decimal output)
`'%'n`	Print n digits without fractional part
`'%'n'.'mm`	Print floating format in a field with n digits using mm digits as fractional part; for octal/hexadecimal/binary output is equivalent to '%'n
`'%0.'mm`	Print number in exponential notation using mm digits as mantissa, mm=0: all digits; for octal/hexadecimal/binary output: print all digits
`'%-1'`	Print number in a field where the value exactly fits in
`']'`	Print in octal notation
`']]'`	Print in hexadecimal notation
`'?'`	Print in binary notation
`'&'n`	Print n blanks
`'\'n`	Print ASCII character of value n

Note:
All formattings may be also used within any concatenation.

Formatting by `'%...'` works also on hexadecimal, octal and binary representation.

Each comma within the type list resets output to standard format.

If the field-width is to small, it is extended until the integral part of the number can be displayed. If the format for numbers in the range from -1 to 1 does not allow to display at least the two most significant digits, output is changed to exponential form (according to `'%0.04'`, using a total width of 11).

In octal, hexadecimal, binary format only the rounded integral part of the number is displayed.

Example:

```
>SE Val=4711.0815
>T %, Val;%           all digits, exponential
 4.711081500000000E3
>T %-1 Val;%          only significant digits
4711.0815
>T %9.03 Val;%        floating format
 4711.082
```

```
>T %0.5 Val;%          exponential format
 4.7111E3
>T % ]] Val;%          hexadecimal
00001267
>T %6 ] Val;%          octal, six digits
011147
>
```

4.34 VALUE

Description:
Assign the return value of a defined function.

Type:
NODAL Command

Syntax:
`V[ALUE] expression`

Parameter:

expression
The value to return to the caller.

4.35 $VALUE

Description:
Assign the return value of a defined string function.

Type:
NODAL Command

Syntax:
`$V[ALUE] concatenation`

Parameter:

concatenation
The string to return to the caller.

4.36 WAIT-T

Description:
Wait a defined amount of time.

Type:
NODAL Command

Syntax:
`WA[IT]-T time`

Parameter:

time
(RealValue) Time to wait in seconds.

Example:

```
>% return with NODAL prompt after 7 seconds
>WA-T 7
>
```

4.37 WHILE

Description:
Execute a WHILE loop.

Type:
NODAL Command

Syntax:
`WH[ILE] log-expression [';' statements]`

Parameter:

log-expression
While the logical expression is true, the body of the loop is executed.

statements
The body of the loop.

4.38 ZDEF

Description:
Erase (zero) all Defined Functions.

Type:
NODAL Command

Syntax:
`ZDEF`

Parameter:

Note:
ZDEF is the abbreviated command of ERASE ALLD.

5 NODAL Functions

Functions implemented for NODAL. They are resident functions and grouped as follows:

- mathematical functions,
- input/output functions,
- string functions,
- pattern functions,
- others.

5.1 ABS

Description:
Returns absolute value of argument.

Type:
NODAL mathematics RO-Function

Syntax:
`ABS'('expression')'`

Parameter:

expression
The value to make absolute.

5.2 AT2

Description:
Calculates the arcustangens of expression1/expression2 in the range of 0 to 2*PIE.

Type:
NODAL mathematics RO-Function

Syntax:
`AT2'('expression1',' expression2')'`

Parameter:

expression1, expression2
The real-value arguments.

5.3 COS

Description:
Returns cosine of argument (must be radian).

Type:
NODAL mathematics RO-Function

Syntax:
`COS'('expression')'`

Parameter:

expression
The angle in radian.

Example:

```
>SE Y=COS(X)
```

5.4 EXP

Description:
Return e^x.

Type:
NODAL mathematics RO-Function

Syntax:
`EXP'('expression')'`

Parameter:

expression
The exponent of e.

5.5 FPT

Description:
Returns fractional part of argument with same sign as argument.

Type:
NODAL mathematics RO-Function

Syntax:
`FPT'('expression')'`

Parameter:

expression
The real-value.

Note:
INT(X) + FPT(X) delivers X

5.6 INT

Description:
Returns integer part of argument.

Type:
NODAL mathematics RO-Function

Syntax:
`INT'('expression')'`

Parameter:

expression
The real-value.

Example:

```
>SE X = INT(1.6);        % X is 1
```

5.7 LOG

Description:
Returns natural logarithm ($log_e x$) of argument.

Type:
NODAL mathematics RO-Function

Syntax:
`LOG'('expression')'`

Parameter:

expression
The real-value.

5.8 MOD

Description:
Returns expression1 modulo expression2.

Type:
NODAL mathematics RO-Function

Syntax:
`MOD'('expression1',' expression2')'`

Parameter:

expression1, expression2
The real-values.

5.9 PIE

Description:
Return the constant $\pi = 3.141592653589793$

Type:
NODAL mathematics RO-Variable

Syntax:
`PIE`

Parameter:

Example:

```
>TY %, PIE
    3.141592653589793E0
```

5.10 SGN

Description:
Returns sign of expression: is 1 for expression >= 0; is -1 for expression < 0.

Type:
NODAL mathematics RO-Function

Syntax:
`SGN'('expression')'`

Parameter:

expression
The real-value.

5.11 SIN

Description:
Returns sinus of argument (must be radian).

Type:
NODAL mathematics RO-Function

Syntax:
`SIN'('expression')'`

Parameter:

expression
The angle in radian.

Example:

```
>SET X = SIN(Z)
```

5.12 SQR

Description:
Returns square root of argument.

Type:
NODAL mathematics RO-Function

Syntax:
`SQR'('expression')'`

Parameter:

expression
The non-negative real-value.

Example:

```
>SE Y=SQR(X)
```

5.13 AND

Description:
Returns the bitwise 'AND' of the two arguments.

Type:
NODAL mathematics RO-Function

Syntax:
```
AND'('expression1',' expression2')'
```

Parameter:

expression1, expression2
The integer values. Real-values are rounded to integer values.

Example:

```
>TYPE AND([[F1,[[F)
    1
```

5.14 BIT

Description:
Set or read a specific bit in a supposed 32-bit word. The bit manipulation procedure may be used on left side of equal sign in SET commands and in arithmetic expressions.

Type:
NODAL mathematics R/W-Function

Syntax:
```
BIT'('expression1',' expression2')'
```

Parameter:

expression1
Bit position, may be 0 - 31.

expression2
Destination, may be any settable item.

Note:
To set a specific bit in a bitset, all values > 0 are assumed as true (1), all values $<= 0$ are assumed as false (0).

Example:

```
>SE a = 0
>SE BIT(2,a) = 1
>T a
    4
>SE a = [[55
>TY BIT(6,a)
    1
```

5.15 IOR

Description:
Returns the bitwise 'OR' of the two arguments.

Type:
NODAL mathematics RO-Function

Syntax:
`IOR'('expression1',' expression2')'`

Parameter:

expression1, expression2
The integer values. Real-values are rounded to integer values.

Parameter:

Example:

```
>T IOR([[F0,[[F)
    FF
```

5.16 NEG

Description:
Bitwise negation of the argument.

Type:
NODAL mathematics RO-Function

Syntax:
`NEG'('expression')'`

Parameter:

expression
The integer value. Real-values are rounded to integer values.

5.17 SHIFT

Description:
Return the bitwise shifted bit pattern.

Type:
NODAL mathematics RO-Function

Syntax:
`SHIFT'('expression1',' expression2')'`

Parameter:
Real-values are rounded to integer values (patterns).

expression1
The pattern to shift.

expression2
Specifies the shift-count:
expression2 > 0 right shift expression2 bits,
expression2 < 0 left shift expression2 bits,
expression2 $= 0$ no shift.

5.18 ASCII

Description:
Form the ASCII value of a concatenation by addition of all characters.

Type:
NODAL Stringhandling RO-Function

Syntax:
```
ASCII'('concatenation')'
```

Parameter:

concatenation
The string to handle.

Note:
```
\ASCII(character) = character
ASCII(\number) = number
```

Example:

```
>TYPE ASCII('ab')
    198
```

5.19 CAP

Description:
Convert a concatenation to uppercase letters.

Type:
NODAL Stringhandling RO-Function

Syntax:
```
CAP'('concatenation')'
```

Parameter:

concatenation
The string to convert.

Example:

```
>TYPE CAP('ab')
AB
```

5.20 EVAL

Description:
Treat a concatenation as an arithmetic expression and return its value.

Type:
NODAL Stringhandling RO-Function

Syntax:
```
EVAL'('concatenation')'
```

Parameter:

concatenation
: The string to evaluate.

Example:

```
>se a = 13
>$se conc = "(3 - 1) * a"
>ty eval(conc)
    26
```

5.21 FIND

Description:
: Return the array index of a string in an array, or -1 if the string is not in the array.

Type:
: NODAL Stringhandling RO-Function

Syntax:
: `FIND'('name',' string')'`

Parameter:

name
: The name of the string array variable.

string
: The string to search for in the array.

Example:

```
>% find string 'empty' in array A and return array index
>DI-S A
>$SE A(3)='ABC'
>$SE A(4)='DUMMY'
>$SE A(7)='EMPTY'
>$SE A(9)='123'
>T FIND(A,'EMPTY')
    7
```

5.22 FINDS

Description:
: Return the array index of a substring in an array or -1 if the substring is not in the array or -2 if the substring is more than one time in the array.

Type:
: NODAL Stringhandling RO-Function

Syntax:
: `FINDS'('name',' substring')'`

Parameter:

name
: The name of the string array variable.

substring
: The substring to search for in the array.

5.23 SIZE

Description:
: Return number of characters in a concatenation.

Type:
: NODAL Stringhandling RO-Function

Syntax:
: `SIZE'('concatenation')'`

Parameter:

concatenation
: The string to measure.

Example:

```
>TY SIZE('abcd')
    4
```

5.24 SORT

Description:
: Sort string array in ascending or descending order.

Type:
: NODAL Stringhandling Call-Function

Syntax:
: `SORT'('array-name',' 'a' ! 'd' ')'`

Parameter:

array-name
: The name of the string array to sort.

'a'
: Sort in ascending order.

'd'
: Sort in descending order.

Example:

```
>DI-S str
>$S str(1)="abc"
>$S str(2)="aaa"
>$S str(3)="aab"
>SORT(str,a)
>F i=1,3; T str(i) " "
aaa aab abc
```

5.25 STRARG

Description:
This is a NODAL task global string variable. It is valid as long as a NODAL task is active. It can be used to communicate between different NODAL programs.

Type:
NODAL Stringhandling R/W-Stringfunction

Syntax:
`STRARG`

Parameter:

Note:
Contrary to ARG only one STRARG is provided. It can hold up to 40 characters. STRARG is not affected by ERASE ALL, OLD, SAVE, LOAD, RUN etc.

Example:

```
>% program 1
>1.10 $SET STRARG = "ABCD";   % set global variable
>1.20 RUN PROGRAM2;           % clear all and run program 2
>% program 2
>1.10 T STRARG;               % the result will be "ABCD"
```

5.26 SUBS

Description:
Return or insert specified substring of given concatenation.

Type:
NODAL Stringhandling R/W-Stringfunction

Syntax:
`SUBS'('startindex',' endindex',' concatenation')'`

Parameter:

startindex
: The first substring character to read or substitude. The first character of the string has the index 1.

endindex
: The last substring character to read or substitude. The first character of the string has the index 1.

concatenation
: The string to read or substitude in.

Example:

```
>% return substring
>$SE str = 'abcdefgh'
>TY SUBS(2,4,str)
bcd
>% insert substring in string
```

```
>$SE SUBS(2,4,str) = 'xyz'
>TY str
axyzefgh
>
```

5.27 ABORT

Description:
A pattern that, if tried, causes the complete failure of the $MATCH command. No more other alternatives are tried.

Type:
NODAL Patternfunction

Syntax:
`ABORT`

Parameter:

Example:

```
>$MAT 'abcdef' 'x' ! ABORT ! 'a'
>% the match fails, alternative 'a' is never tried
```

5.28 ANY

Description:
Match any single character contained in the concatenation.

Type:
NODAL Patternfunction

Syntax:
`ANY'('concatenation')'`

Parameter:

concatenation
Contains the characters to search for.

Example:

```
>$MATCH 'abcdef' ANY('xby')
>% matches because of 'b'
```

5.29 ARB

Description:
Match any number of any character.

Type:
NODAL Patternfunction

Syntax:
`ARB`

Parameter:

Example:

```
>$MATCH 'abcdef' 'b' ARB 'e';          %matches 'bcde'
>$MATCH 'ab123abcxyzef' 'b' ARB 'e';   %matches 'b123abcxyze'
```

5.30 BREAK

Description:
Match up to but not including any break character contained in the concatenation, start from the current cursor position. BREAK is the opposite of SPAN.

Type:
NODAL Patternfunction

Syntax:
`BREAK'('concatenation')'`

Parameter:

concatenation
Contains the break characters.

Note:
For cursor positioning see functions POS and RPOS.

Example:

```
>$MATCH 'abcdef321' POS(2) BREAK('123') .var1
>TY var1
cdef
```

5.31 FAIL

Description:
A pattern that, if tried, causes a pattern mis-match of the $MATCH command and so causes other alternatives to be tried.

Type:
NODAL Patternfunction

Syntax:
`FAIL`

Parameter:

Example:

```
>$PAT p1 = 'abc'
>$PAT p2 = 'def'
>$MATCH 'abcdef' (p1 ! p2) .var
>TY var
abc
>$PAT p1 = FAIL
>$PAT p2 = 'def'
>$MATCH 'abcdef' (p1 ! p2) .var
>TY var
def
```

5.32 LEN

Description:
Match any string of given length.

Type:
NODAL Patternfunction

Syntax:
`LEN'('length')'`

Parameter:

length
The length of the string to be matched.

Example:

```
>$MATCH 'abc' LEN(3);      % matches
>$MATCH 'abcd' LEN(3);     % does not match
```

5.33 NOTANY

Description:
Match any single character not contained in the concatenation.

Type:
NODAL Patternfunction

Syntax:
`NOTANY'('concatenation')'`

Parameter:

concatenation
Contains the characters not to match.

Example:

```
>$MATCH 'abcd' NOTANY('abd') .var
>TY var
c
```

5.34 POS

Description:
Match only when the cursor is in the position. This anchors the pattern only to occur at a fixed point in the string. Matching starts just to the right of the cursor position. POS(0) is just to the left of the first character.

Type:
NODAL Patternfunction

Syntax:
`POS'('position')'`

Parameter:

position
: The cursor position counted from the beginning of the string where the pattern should be anchored.

Example:

```
>$MATCH 'abcdef' POS(2) 'cd';       % matches
>$MATCH 'abcdef' POS(3) 'cd';       % does not match
```

5.35 RPOS

Description:
: Match only when the cursor is in the position. The position is counted from the end of the string. This anchors the pattern only to occur at a fixed point in the string. Matching starts just at the cursor position. RPOS(0) is just to the right of the last character.

Type:
: NODAL Patternfunction

Syntax:
: `RPOS'('position')'`

Parameter:

position
: The cursor position counted from the end of the string where the pattern should be anchored.

Example:

```
>$MATCH 'abcdef' RPOS(4) 'cd';       % matches
>$MATCH 'abcdef' RPOS(5) 'cd';       % does not match
```

5.36 RTAB

Description:
: Match up to but not including position, where position is counted from the end of the string, starting from the current cursor position.

Type:
: NODAL Patternfunction

Syntax:
: `RTAB'('position')'`

Parameter:

position
: The position counted from the end of the string.

Note:
: For cursor positioning see functions POS and RPOS.

Example:

```
>$MATCH 'abcdef' POS(1) RTAB(2) .var
>T var
bcd
```

5.37 SPAN

Description:
Match the run of characters contained in the concatenation, start from the current cursor position.

Type:
NODAL Patternfunction

Syntax:
```
SPAN'('concatenation')'
```

Parameter:

concatenation
Contains the characters to search for.

Note:
For cursor positioning see functions POS and RPOS.

Example:

```
>$MATCH 'ABCabc123' SPAN('CBA') .var;   % !!! case sensitive
>T var
ABC
```

5.38 TAB

Description:
Match up to and including position, start from the current cursor position.

Type:
NODAL Patternfunction

Syntax:
```
TAB'('position')'
```

Parameter:

position
The position counted from the beginning of the string.

Note:
For cursor positioning see functions POS and RPOS.

Example:

```
>$MATCH 'abcdef' POS(1) TAB(4) .var
>T var
bcd
```

5.39 ALPHA

Description:
Return the uppercase characters A... Z.

Type:
NODAL 'others' RO-Stringfunction

Syntax:
`ALPHA`

Parameter:

Example:

```
>T ALPHA
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

5.40 ARG

Description:
This is an array of 16 NODAL task global variables. It is valid as long as a NODAL task is active. It can be used to communicate between different NODAL programs.

Type:
NODAL 'others' R/W-Function

Syntax:
`ARG'('index')'`

Parameter:

index
The array index range is 1... 16.

Note:
ARG(1)... ARG(16) are not affected by ERASE ALL, OLD, SAVE, LOAD, RUN etc.

Example:

```
>% program 1
>1.10 SET ARG(13) = 0815;     % set global variable
>1.20 RUN PROGRAM2;           % clear all and run program 2
>% program 2
>1.10 T ARG(13);              % the result will be 0815
```

5.41 ARSIZE

Description:
Return the number of elements in the array, which name is specified by the argument.

Type:
NODAL 'others' RO-Function

Syntax:
`ARSIZE'('array')'`

Parameter:

array
: The name of the array.

Example:

```
> DI ARRAY(11)
> T ARSIZE(ARRAY)
     11
```

5.42 BRKPT

Description:
: NODAL debugger command to set a breakpoint.

Type:
: NODAL 'others' Call-Function

Syntax:
: `BRKPT'('environment',' line',' command')'`

Parameter:

environment
: (NODALnameValue) 'MAIN' for the main program or the name of a defined function.

line
: (RealValue) The line number.

command
: (IntegerValue) The number of the command in the specified line. The program breaks before this command.

Note:
: For more information about the NODAL debugger, see [4].

Example:

```
>% break execution in the main program at line 10.20
>% before the second command (SE B=2).
>10.20 SE A=1; SE B=2
>BREAK(MAIN, 10.20, 2)
```

5.43 COPY

Description:
: Copy the contents of an integer or real array into another integer or real array. Copying from integer to real or from real to integer is possible.

Type:
: NODAL 'others' Call-Function

Syntax:
: `COPY'('sourcename',' destname',' sourceindex',' destindex')'`

Parameter:

sourcename
: The name of the source array.

destname
: The name of the destination array.

sourceindex
: The index of the source array from where to start the copy.

destindex
: The index of the destination array from where to start the copy.

Note:
: Copy ends if the size of the source or destination array is reached.

Example:

```
>DI-I A(5)
>DI B(3)
>SE A(1)=1; SE A(2)=2; SE A(3)=3; SE A(4)=4; SE A(5)=5
>COPY(A, B, 3, 2)
>TY 'Array A = '; F I=1,5; T A(I)
Array A =   1   2   3   4   5
>TY 'Array B = '; F I=1,3; T B(I)
Array B =   0   3   4
```

5.44 ERMES

Description:
: Returns the error message corresponding to the input error number.

Type:
: NODAL 'others' RO-Stringfunction

Syntax:
: `ERMES'('expression')'`

Parameter:

expression
: The NODAL error number

Example:

```
>TY ERMES(13)
Nonexistend Line Addressed
```

5.45 ERROR

Description:
: Returns the last occurred NODAL error when used as term in an arithmetic expression, or triggers the NODAL error mechanism, when used on the left side in a SET command.

Type:
: NODAL 'others' R/W-Function

Syntax:
: `ERROR`

Parameter:

Note:
Error number 50 would force abort of NODAL, so it is not allowed to assign 50 to ERROR. If you do so, Error 33 (Unauthorized Action) is generated instead.

Example:

```
>SET ERROR=9
*** NODAL ERROR 9     Wrong Variable Type

>set abc
*** NODAL ERROR 41 SYNTAX ERROR

>TYPE ERROR
    41
```

5.46 LISD

Description:
List all Defined Functions.

Type:
NODAL 'others' Call-Function

Syntax:
`LISD`

Parameter:

Note:
LISD is the abbreviated function of LIST ALLD.

5.47 LISR

Description:
List all resident functions with type and formal parameters.

Type:
NODAL 'others' Call-Function

Syntax:
`LISR`

Parameter:

Note:
LISR is the abbreviated function of LIST ALLR.

5.48 LISV

Description:
List all user defined variables and arrays with type and dimension.

Type:
NODAL 'others' Call-Function

Syntax:
`LISV`

Parameter:

Note:
LISV is the abbreviated function of LIST ALLV.

5.49 LSTBRK

Description:
NODAL debugger command to list all currently set breakpoints.

Type:
NODAL 'others' Call-Function

Syntax:
`LSTBRK`

Parameter:

Note:
For more information about the NODAL debugger, see [4].

5.50 MAX

Description:
Find the maximum value of an integer or real array.

Type:
NODAL 'others' RO-Function

Syntax:
`MAX'('array')'`

Parameter:

array
The name of the array.

5.51 MIN

Description:
Find the minimum value of an integer or real array.

Type:
NODAL 'others' RO-Function

Syntax:
`MIN'('array')'`

Parameter:

array
The name of the array.

5.52 NODLIN

Description:
Get or create a NODAL line.

Type:
NODAL 'others' R/W-Stringfunction

Syntax:
`NODLIN'('line-number')'`

Parameter:

line number
A NODAL line number.

Example:

```
>% get the text of line 1.10 in var without the line number
>1.10 SET A=B
>$SE var = NODLIN(1.10)
>TY var
SET A=B
>% create a new line
>$SE NODLIN(1.20) = 'SE B=B/2'
>LIST
1.10 SET A=B
1.20 SE B=B/2
```

5.53 NUM

Description:
Returns the digits 0...9

Type:
NODAL 'others' RO-Stringfunction

Syntax:
`NUM`

Parameter:

Example:

```
>TY NUM
0123456789
```

5.54 UNBRK

Description:
NODAL debugger command to clear a specified breakpoint or to clear all breakpoints in an environment.

Type:
NODAL 'others' Call-Function

Syntax:

```
UNBRK'('environment',' line',' command')'
```

Parameter:

environment
: (NODALnameValue) 'MAIN' for the main program or the name of a defined function.

line
: (RealValue) The line number.

command
: (IntegerValue) The number of the command in the specified line.

Note:

For more information about the NODAL debugger, see [4].

Example:

```
>% clear the breakpoint in line 30.40 in function FUN
>UNBRK(FUN, 30.40, 1)
>% clear all breakpoints in main
>UNBRK(MAIN, 0.0, 0)
```

5.55 UPGRADE

Description:

UPGRADE helps to convert NODAL-programs developed under version 0.9 to the syntax required by version 1.00 (and further) which differ slightly. It modifies the program currently resident in memory by checking every program line and inserts blanks in the commands TYPE, ASK, $ASK and DO if nessecary. Furthermore it checks the usage of new defined reserved names MIN and MAX.

The syntax of the commands TYPE, ASK and $ASK requires the separation of strings and other data objects (which may be also strings) by spaces or commas. Versions before 1.00 did not require these separations and even removed eventually inserted spaces after first execution of the corresponding line. Where nessecary, UPGRADE includes one extra space to seperate the elements in the list of the commands TYPE, ASK and $ASK.

If alternations (identified by exclamation marks) in the command DO are specified they must be preceeded by at least one space. Versions before 1.00 did not require this and even removed eventually inserted spaces after first execution of the corresponding line. UPGRADE includes a space when missing before the alternation.

By including extra spaces the program lines become longer and may exceed the maximum length of the line (79 characters for statements plus additional 6 characters for the line number). If the modified line becomes too long, a warning message (showing the line number) is displayed and the line is truncated. It had explicitly to be shortened, i.e. by splitting into two lines.

Special care has to be taken when using the NODAL-editor EDIT: lines exceeding 79 characters (inclusive line number) may be displayed, but when modified they are truncated to 79 characters. In versions before 1.00 calling the LSEDIT will result in truncation of all lines to 79 characters. To indicate these lines, a warning message (including the line number) is displayed.

From version 1.00 on new functions MIN and MAX are introduced. Unfortunately the name MIN was often used as a variable name for storage of the number of minutes when

calculating times. UPGRADE indicates the lines where MIN and MAX are not used as functions, the names then had to be modified by the user.

After using UPGRADE the program had to be saved to make the changes permanent.

Type:
NODAL 'others' Call-Function

Syntax:
`UPGRADE`

Parameter:

6 File-I/O Functions

These functions form the interface for file input and output.

6.1 CLOSE

Description:
Close one file or all files after input or output is done.

Type:
File-I/O Call-Function

Syntax:
`CLOSE'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to close a specific file, or zero to close all previously opened files.

Example:

```
>SE fid = OPEN('R', 'READFILE.DAT')
>$SE str = INPUT(fid)
>CLOSE(fid)
```

6.2 INPBT

Description:
Return the first/next byte from the input stream. The last byte returned is the End Of Line character (EOL = \0). Trying to read more bytes causes a file error.

Type:
File-I/O RO-Function

Syntax:
`INPBT'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to read from the specific file.

Note:
To be shure that all bytes are read, read until a file error occurs and handle this error yourself.

Example:

```
>SE fid = OPEN('R', 'READFILE.DAT')
>SE var = INPBT(fid)
>TYPE var
    49
>CLOSE(fid)
```

6.3 INPC

Description:
Return the first/next character from the input stream. The last character returned is the End Of Line character (EOL = \0). Trying to read more characters causes a file error.

Type:
File-I/O RO-Stringfunction

Syntax:
`INPC'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to read from the specific file.

Note:
To be shure that all characters are read, read until a file error occurs and handle this error yourself.

Example:

```
>SE fid = OPEN('R', 'READFILE.DAT')
>$SE char = INPC(fid)
>TYPE char
A
>CLOSE(fid)
```

6.4 INPUT

Description:
Return the first/next line from the input stream. A line is terminated by 'Carriage Return' (CR = \13) followed by 'Line Feed' (LF = \10) or by the EOL character \0. The line terminators are not copied. Trying to read more than the last line causes a file error.

Type:
File-I/O RO-Stringfunction

Syntax:
`INPUT'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to read from the specific file.

Note:
To be shure that all lines are read, read until a file error occurs and handle this error yourself.

Example:

```
>SE fid = OPEN('R', 'READFILE.DAT')
>$SE string = INPUT(fid)
>TYPE string
ABCDEF
>CLOSE(fid)
```

6.5 OPEN

Description:
Open a file for reading or writing.

Type:
File-I/O RO-Function

Syntax:
`OPEN'('type',' file')'`

Parameter:

type

- `"R"` or `"r"` for reading from a file,
- `"W"` or `"w"` for writing to a file,
- `"A"` or `"a"` for appending to a file.

file
The desired filename.

Example:

```
>% open a file for writing
>$SE filename = 'WRITEFILE.DAT'
>$SE type = 'W'
>SE fid = OPEN(type, filename)
>% open a file for reading
>SE fid = OPEN("r", "READFILE.DAT")
```

6.6 OUTBT

Description:
Write the first/next byte to the output stream.

Type:
File-I/O WO-Function

Syntax:
`OUTBT'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to write to the specific file.

Example:

```
>SE byte = 49
>SE fid = OPEN('W', 'WRITEFILE.DAT')
>SE OUTBT(fid) = byte
>CLOSE(fid)
```

6.7 OUTC

Description:
Write the first/next character to the output stream.

Type:
File-I/O WO-Stringfunction

Syntax:
`OUTC'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to write to the specific file.

Example:

```
>$SE char = 'A'
>SE fid = OPEN('W', 'WRITEFILE.DAT')
>$SE OUTC(fid) = char
>CLOSE(fid)
```

6.8 OUTPUT

Description:
Write the first/next string to the file, append 'Carriage Return' (CR = \13) followed by 'Line Feed' (LF = \10) to the end of the string.

Type:
File-I/O WO-Stringfunction

Syntax:
`OUTPUT'('file-id')'`

Parameter:

file-id
The file identification given by the corresponding OPEN command to write to the specific file.

Example:

```
>$SE string = 'ABCDEF'
>SE fid = OPEN('W', 'WRITEFILE.DAT')
>$SE OUTPUT(fid) = string
>CLOSE(fid)
```

7 SIS/ESR Control System Functions

SIS/ESR Control System (SISCOS) functions, including Userface.
These functions support the general access to the control system for the GSI accelerators. There are specialized functions for collecting informations about the current state of equipment and the general interface to the control system called Userface (for further information, see [5]).
Some functions support the error tracking after failed Datamodule Calls.

7.1 EQMEVENT

Description:
Return the EQM event stamp of the last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`EQMEVENT`

Parameter:

Note:
The function USRESTAM is doing exactly the same.

Example:

```
SE Y=USRESTAM
```

7.2 EQMTIME

Description:
Returns the 4 byte formatted EQM timestamp information (hour:minute:second:second/100) of the last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`EQMTIME`

Parameter:

Note:
See also function USRTSTAM.

7.3 LISDM

Description:
Lists all nomenclatures found actually in the SIS Database. Separate lists are generated for devices and computers. The status is marked by reverse printout (reverse = online).

Type:
SISCOS Call-Function

Syntax:
`LISDM`

Parameter:

Note:
Call of `LISDM` is equivalent to `LSTDM("*")`

7.4 LSTDM

Description:
List selected datamodules (= nomenclatures). All nomenclatures found in the SIS Database according to the given nomenclature are listed in separate lists for devices and computers. The status is marked by reverse printout (reverse = online).

Type:
SISCOS Call-Function

Syntax:
`LSTDM'('nomenclature')'`

Parameter:

nomenclature
(StringValue) with wild cards for selection.

Note:
Call of `LSTDM("*")` is equivalent to `LISDM`

Example:

```
>LSTDM("TK*");   % lists all nomenclatures beginning with TK
>LSTDM("S*MS*"); % lists all steerer magnets in the SIS
```

7.5 LSTNOMEN

Description:
List infos from the mapping table currently found in the Database:

- nomenclature, device type, device subtype, equipment model, SIS address, device number
- for each property:
  property, property class, XSR number, property category, parameter count, parameter type, working data count, working data type, timeout (in seconds), exponent of unit, unit.

Type:
SISCOS Call-Function

Syntax:
`LSTNOMEN'('nomenclature')'`

Parameter:

nomenclature
(StringValue) specifies the nomen for searching in the mapping table.

Example:

```
>% This call lists the mapping infos from magnet 'S01MU2'.
>CALL LSTNOMEN("S01MU2")
```

7.6 UFCFACIL

Description:
Get facility code of the Userface error messages.

Type:
SISCOS RO-Variable

Syntax:
`UFCFACIL`

Parameter:

7.7 USRCFACI

Description:
Get facility code from completion state of last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`USRCFACI`

Parameter:

Example:

```
>% get facility and error code from last equipment access
>% and print out the message
>SE FACIL=USRCFACI
>SE ERROR=USRCSTAT
>T GETMSG1(FACIL,ERROR)
%MX-E-ChkPwr_Off_Fail, EQM 'CheckPower': Power could not ...
```

7.8 USRCSTAT

Description:
Return the USR completion state of the last Data Module Call with the facility code masked out.

Type:
SISCOS RO-Function

Syntax:
`USRCSTAT`

Parameter:

Example:

```
>SET X = USRCSTAT
```

7.9 USRCOMPL

Description:
Return information about USR completion of the last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`USRCOMPL'('select ['/'subselect]')'`

Parameter:

select
Specifies which USR information is to be returned. May be specified either as a numerical value or as a string. Strings may be abbreviated up to the first character.
Valid values:

num	string	
0	READCOUNT	number of data items returned by USR
1	PRIMARYSTATUS	primaray status of USR execution
2	SECONDARYSTATUS	secondaray status of USR execution
3	TIMESTAMP	time stamp (hour—min—sec—1/100)
4	[E]VENTSTAMP	event stamp
5	ENERGY	energy step (therapy execution only)
6	FOCUS	focus step (therapy execution only)
7	INTENSITY	intensity step (therapy execution only)
8	CYCLECOUNT	cycle count (therapy execution only)
9	DATAIDENT	data ident (therapy execution only)

subselect
If select parameter is specified by string single elements of structured USR parameters may be returned, classified by subselect. The subselect parameter may be abbreviated up to the first character. Valid values if select = TIMESTAMP:

HOUR	timestamp: hour field	
MINUTE	timestamp: minute field	Valid values if select
SECOND	timestamp: second field + 1/100 * fraction field	
	MACHINEID	machine ident field of data ident (16 Bit)
= DATAID:	DEVICEID	device ident field of data ident (8 Bit)
	SETID	set ident field of data ident (8 Bit)

Example:

```
>TYPE USRCOMPL(1);%            primary completion status
>TYPE USRCOMPL('P');%            ditto
>TYPE USRCOMPL('Prim');%         ditto
>TYPE USRCOMPL(4.0);%          event stamp
>TYPE USRCOMPL('V');%            ditto
>TYPE USRCOMPL('EV');%           ditto
>TYPE USRCOMPL('Event');%        ditto
>TYPE USRCOMPL('Timestamp');%  time stamp
>TYPE USRCOMPL('Time/Hour');%  hour part of time stamp
>TYPE USRCOMPL('Time/Sec');%   second parts (+fraction) of time stamp
>FOR i=0,9;TY USRCOMPL(i) !;%  all USR completion information
```

7.10 USRESTAM

Description:
Return the EQM event stamp of the last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`USRESTAM`

Parameter:

Note:
The function EQMEVENT is doing exactly the same.

Example:

```
>SET X=USRESTAM
```

7.11 USRSFACI

Description:
Get facility code from secondary state of last Data Module Call.

Type:
SISCOS RO-Function

Syntax:
`USRSFACI`

Parameter:

Example:

```
>SE FACIL=USRSFACI
```

7.12 USRSSTAT

Description:
Return the USR completion state of the last Data Module Call with the facility code masked out.

Type:
SISCOS RO-Function

Syntax:
`USRSSTAT`

Parameter:

Example:

```
>SET X = USRSSTAT
```

7.13 USRSTAT

Description:
Return the USR completion and secondary state of the last Data Module Call with the facility code masked out.

Type:
SISCOS Call-Function

Syntax:
`USRSTAT'('cstat',' sstat')'`

Parameter:

cstat
(RealReference) USR Completion State

sstat
(RealReference) USR Secondary State

Example:

```
>CALL TK9QD21(.....);            % calls equipment
>CALL USRSTAT(CSTAT, SSTAT);     % ask for completion state
>T ! "USR State: CSTAT=" CSTAT "SSTAT=" SSTAT;   % prints
```

7.14 USRTSTAM

Description:
Interprets the 4 byte timestamp information of the last Data Module Call. Return value is the time in seconds counted from the beginning of the day.

Type:
SISCOS RO-Function

Syntax:
`USRTSTAM`

Parameter:

Example:

```
>SE T=USRTSTAM
>SE HOUR=INT(T/3600)
>SE MINUTE=INT((T/60)-(HOUR*60))
>SE SECOND=T-(MINUTE*60)-(HOUR*3600)
>T "Timestamp is " %2 HOUR ":" %2 MINUTE ":" %5.02 SECOND
Timestamp is 16:12:22.77
```

7.15 VRTACC

Description:
Read or modify the actual virtual accelerator number. NODAL uses this number to specify the virtual accelerator in every Data Module Call.

Type:
SISCOS R/W-Function

Syntax:
`VRTACC`

Parameter:

Example:

```
>TYPE VRTACC
    5
>SET VRTACC=9
>TYPE VRTACC
    9
```

7.16 UFCBI

Description:
Type out statistical informations about the Userface Input Buffer on the Terminal.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCBI`

Parameter:

7.17 UFCCI

Description:
Type out informations about open channels in Userface on the Terminal.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCCI`

Parameter:

7.18 UFCCLOSE

Description:
Close channel and delete all buffered responses concerning this channel.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCCLOSE'('channel')'`

Parameter:

channel
(RealValue) channel number

7.19 UFCCONN

Description:
Connect a task to a time, to a time interval or to an event of the central timing unit. The connection type is specified by the corresponding UFCOPEN command.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCCONN'('channel')'`

Parameter:

channel
(RealValue) channel number

Example:

```
>% periodic connection of a task in channel 2
>UFCCONN(2)
```

7.20 UFCDCON

Description:
Disconnect a Connected Task in the specified channel.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCDCON'('channel')'`

Parameter:

channel
(RealValue) channel number

7.21 UFCEQACC

Description:
Initiate a Simple Task to be executed immediately and only once. A Simple Task can be handled synchronously, if the user's program wants to wait until the task is completly finished. Otherwise, the task is termed asynchronous.

In the synchronous case a task is said completly finished, if a response from the equipment was received and handled (data transfer or error handling) or no response was received within a specific time interval (timeout).

If an asynchronous task handling is requested, Userface separates the task initiation and response handling internally. After task initiation it returns directly to the user program. In this mode it is possible to suppress or demand the response, depending on the user's program's need.

Type:
SISCOS Userface Call-Function

Syntax:

```
UFCEQACC'('fct',' info',' par',' dat',' cst',' chn',' eflg')'
```

Parameter:

function
: (StringArray) with the elements:

  1. Nomenclature
  2. Property
  3. Property Class (in Userface notation: R, W, N, RA, WA)

info
: (Real/Integer-ARRAY) for numeric/logic information for the command:

  1. number of desired virtual accelerator
  2. sync-mode (0 = synchron, else = asynchron)
  3. quit-mode (0 = forced_quit, else = no_quit)
  4. data count (for write commands: count of data elements to send, for read commands: free space of elements in data array)
  5. data read count (for read commands: count of data returned)

parameter
: (Real/Integer-ARRAY) array of parameters (dim=0 -> no parameters)

data
: (Real/Integer-ARRAY) array of data

USR completion state
: (Real/Integer-ARRAY) array to return USR completion information. Minimum size is 1. The array elements have the meaning:

  1. primaray status of USR execution
  2. secondaray status of USR execution
  3. time stamp (hour—min—sec—1/100)
  4. event stamp
  5. energy step (therapy execution only)
  6. focus step (therapy execution only)
  7. intensity step (therapy execution only)
  8. cycle count (therapy execution only)
  9. data ident (therapy execution only)

channel
: (RealValue) channel for asynchronous tasks to associate the response of a task with its initiation. The channel is closed automatically after the response has been accessed.

event flag
: (RealValue) event flag for asynchronous tasks to be set when the initiated task responses.

Example:

```
>% write reference value to TK9QT13
>% same as SET TK9QT13(VOLTS)=1024,
>% but asynchronous without quit
>DIM-I INFO(5);%.............. dimension of arrays
>DIM PARA(0);%................. dto.
```

```
>DIM-S FUNC;%.................. dto.
>DIM DATA(1);%................. dto.
>DIM-I USRCST(4)%.............. dto.
>$SET FUNC(1) = "TK9QT13";%.... definition of the command
>$SET FUNC(2) = "VOLTS";%...... dto.
>$SET FUNC(3) = "W";%.......... dto.
>SET INFO(1) = 8;%............. set vrtacc
>SET INFO(2) = 1;%............. set asynchr. mode
>SET INFO(3) = 1;%............. set no quit
>SET INFO(4) = 1;%............. spec. of data
>SET INFO(5) = 0;%............. dummy
>SET DATA(1) = 1024;%.......... value to set
>UFCEQACC(FUNC,INFO,PARA,DATA,USRCST,0,0); %....call
```

7.22 UFCLSTEX

Description:
Return the last Userface Exception as message string.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCLSTEX`

Parameter:

Example:

```
>UFCLSTEX
%UFC-E-TIMEOUT, timeout during wait on USR response
```

7.23 UFCOPEN

Description:
Open a channel for Connected Tasks. Connected Tasks are not initiated immediately when the task is given to Userface but depending on events. The task may be connected to a time, to a time interval or to an event of the central timing unit. Connected Equipment Instructions are initiated automatically when the specified event occurs.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCOPEN'('ctyp',' cinf',' fct',' inf',' par',' dat',' chn',' eflg')'`

Parameter:

connection type
(StringValue) with:

- 'E' or 'e': event connection of a task
- 'P' or 'p': periodic connection of a task
- 'T' or 't': time connection of a task

connection info
: (RealValue) additional connection type information:
  - central timing unit event (if ctype = 'E' or 'e'),
  - time interval in seconds (if ctype = 'P' or 'p'),
  - absolute time of day (if ctype = 'T' or 't')

function
: (StringArray) array with the elements:
  1. Nomenclature
  2. Property
  3. Property Class (in Userface notation R, W, N, RA, WA)

info
: (Real/Integer-ARRAY) for numeric/logic information for the command:
  1. number of desired virtual accelerator
  2. sync-mode (0 = synchron, else = asynchron)
  3. quit-mode (0 = forced_quit, else = no_quit)
  4. data count (for write commands: count of data elements to send, for read commands: free space of elements in data array)
  5. not evaluated

parameter
: (Real/Integer-ARRAY) array of parameters (dim=0 -> no parameters)

data
: (Real/Integer-ARRAY) array of data

USR completion state
: (Integer-Array) array to return USR completion and secondary state

channel
: (RealValue) channel for asynchronous tasks to associate the response of a task with its initiation.

event flag
: (RealValue) event flag for asynchronous tasks to be set when the initiated task responses.

Example:

```
>% open channel to read voltage of 'TK9QT13'
>% every half second in channel 3
>DIM-S FUNC;%..................... dimension
>DIM-I INFO(5);%.................. dto.
>DIM PARA(0);%.................... dto.
>DIM DATA(1);%.................... dto.
>SET CTYPE = 0.5;%................ time interval in seconds
>$SET FUNC(1) = "TK9QT13";%....... spec. of command
>$SET FUNC(2) = "VOLTI";%......... dto.
>$SET FUNC(3) = "R";%............. dto.
>SET INFO[4]=ARSIZE(DATA);%....... spec. of extended data
>UFCOPEN("P",CTYPE,FUNC,INFO,PARA,DATA,3,0);%--- command
```

7.24 UFCPURGE

Description:

In the default case Userface samples all responses of a specific channel in order of reception in its associated input buffer. So, no response is lost.

With UFCPURGE the user specifies the maximum count of responses to save. If the count is reached and another response arrives, Userface deletes the oldest response in the input buffer and adds the new one.

With keepcount = 1, only the last response is available; keepcount = 0 keeps all responses.

Type:

SISCOS Userface Call-Function

Syntax:

```
UFCPURGE'('channel',' keepcount')'
```

Parameter:

channel
: (RealValue) specifies the desired channel.

keepcount
: (RealValue) specifies the maximum count of responses to keep

7.25 UFCREAD

Description:

Read the oldest response from an asynchronuos channel.

Type:

SISCOS Userface Call-Function

Syntax:

```
UFCREAD'('channel',' info',' timeout',' data',' usrcst')'
```

Parameter:

channel
: (RealValue) channel number

info
: (Real/Integer-ARRAY) for numeric/logic information for the command:

  1. not evaluated
  2. not evaluated
  3. not evaluated
  4. not evaluated
  5. return count of data

timeout
: (RealValue) timeout in seconds

data
: (Integer/Real-ARRAY) data array

USR completion state
: (Real/Integer-ARRAY) array to return USR completion information. Minimum size is 1. The array elements have the meaning:

1. primaray status of USR execution
2. secondaray status of USR execution
3. time stamp (hour—min—sec—1/100)
4. event stamp
5. energy step (therapy execution only)
6. focus step (therapy execution only)
7. intensity step (therapy execution only)
8. cycle count (therapy execution only)
9. data ident (therapy execution only)

Note:
Data array must have minimum the size spezified in the corresponding UFCOPEN or UFCEQACC command.

Example:

```
>% read the quit response of example UFCEQACC
>DIM-I info(5)
>DIM data(20)
>DIM usrcst(2)
>UFCREAD(1, info, 5, data, usrcst)
```

7.26 UFCSDAT

Description:
Access to SISDataType storage format. Intended to handle SISCOS data type 'Structure'. This type 'Structure' is used to exchange mixed format data with devices. UFCSDAT is used to get the bytewise representation of any of the allowed SISCOS data representation (types SISDataType). Vice versa UFCSDAT delivers the value of any SISCOS datatype given by its bytewise representation.

Type:
SISCOS Userface R/W-Function

Syntax:
`UFCSDAT'('dat','ind','type')'`

Parameter:

dat
(Real/Integer-ARRAY) byte representation of data
(1 byte per element)

ind
(RealValue) first element in dat to convert

type
(StringValue) type of data (SISCOS datatype SISDataType)
valid data are:
'BitSet8'—'B8'
'BitSet16'—'B16'
'BitSet32'—'B32'
'Integer8'—'I8'

'Integer16'—'I16'
'Integer32'—'I32'
'RealF'—'RF'
'RealD'—'RD'

Both upper and lower case letters may be used.

Example:

```
>DI Byte(10);% used for bytewise representation of data
>SET ival=37
>SET cval=42
>SET rval=17.04
>%   store rval as RealF (size 4 bytes) in bytes 1..4
>SET UFCSDAT(Byte,1,'RealF')=rval
>%   store ival as Integer16 (size 2 bytes) in bytes 5,6
>SET UFCSDAT(Byte,5,'Integer16')=ival
>%   store cval as BitSet32 (size 4 bytes) in bytes 7..10
>SET UFCSDAT(Byte,9,'BitSet32')=cval
>% check byte representaion by converting back
>%   interpret first elements as VAX Real
>TYPE UFCSDAT(Byte,1,'RealF')
>%   interpret next two elements as Integer16
>TYPE UFCSDAT(Byte,5,'Integer16')
>%   interpret last elements as BitSet32,
>%   value may be stored also in a variable
>SET TmpVar=UFCSDAT(Byte,9,'BitSet32');TYPE TmpVar
```

7.27 UFCRESET

Description:
Reset Userface to a defined initial state. Close all open channels and delete all responses.

Type:
SISCOS Userface Call-Function

Syntax:
`UFCRESET`

Parameter:

7.28 UFCWAIT

Description:
Wait for response. If a response arrives, return the associated channel number. If a response has already arrived, return the channel number immediatly. If timeout is reached, return -1.

Type:
SISCOS Userface RO-Function

Syntax:
`UFCWAIT'('timeout')'`

Parameter:

timeout
: (RealValue) maximum time to wait in seconds

Example:

```
>% wait maximum 2s for response, type out reception channel number
>TYPE UFCWAIT(2)
```

8 DBS Functions

Interface to database.
The informations about equipment needed to control the accelerators at GSI are stored in one main database. This database is evaluated by NODAL when executing equipment accesses. To allow explicit reading of entries from this database the DBS functions are implemented. For detailed description of the GSI database system (REDABAS) see [6].
To access the database, first one of the different tables has to be selected and the search attributes (their names and values) have to be defined. In a second step, successive entries are searched and the values of specified read attributes are extracted. Entries in the database may be strings, numerical and boolean values. To allow a general access for all kind of entries, values have to be specified (and are returned) as strings. Numerical data may be converted to strings by simply assigning them to a string variable or an element of a string array, using optional format specifications (see concatenation). For conversion of strings to numerical data the resident function EVAL may be used.

8.1 DBREAD

Description:

Read entries in the database. Entries specified by DBSET are searched and the values of attributes specified in DBREAD are returned. The first call of this function after calling of DBSET reads the first entry in the database, further calls read successive entries. If the access is successful the number of the entry is returned, else 0 (i.e. if no more entries are found).

Type:

DBS RO-Function

Syntax:

```
DBREAD'('attribute_nr',' attribute_names',' attribute_values')'
```

Parameter:

attribute_nr
: (RealValue) number of entries in attribute_names. If -1 is specified, all elements of attribute_names up to the first empty string are evaluated.

attribute_names
: (StringArray) specifies names of the attributes whose values are to be read from the database.

attribute_values
: (StringArray) returns the values of the attributes, defined in corresponding elements of attribute_names, read from the database.

Note:

The procedure DBSET must be called before first use of DBREAD.

Attributes of type boolean return 1 for TRUE or 0 for FALSE.

Example:

```
>% first define search-criteria
>DI-S Search;%                    define array for access
>$S Search(1)=' nomen   = *M* ';% define first search-criteria
>$S Search(2)=' address = 97  ';% define second search-criteria
```

```
>SE Stat=DBSET('devices',2,Search);% set search
>%
>% then define attributes to read for each entry found
>DI-S ReadName;%               array to specify attributes to read
>DI-S ReadVal;%                array to return value of attributes
>$S ReadName(1)=' nomen ';%    read the nomen,
>$S ReadName(2)=' dev_num ';%    the device-number and
>$S ReadName(3)=' dev_typ ';%    the device-type
>SE N=3;%                      three attributes to read
>%
>% read until no more entries found and display result
>WH DBREAD(N,ReadName,ReadVal)>0;F i=1,N;T ReadName(i) ':' ReadVal(i) !
```

8.2 DBSET

Description:

Define the search in the database. If the function finished successfully, 1 is returned, otherwise 0.

Type:

DBS RO-Function

Syntax:

```
DBSET'('table_name',' attribute_nr',' search_array')'
```

Parameter:

table_name
: (StringValue) name of the DBS-table to search within.

attribute_nr
: (RealValue) number of entries in search_array. If -1 is specified, all elements of search_array up to the first empty string are evaluated.

search_array
: (StringArray) specifies searching in DBS. Every entry has to be specified as 'attribute-name' '=' 'attribute-value', blanks may be used to separate the items, but must not appear within items. A '=' within an 'attribute-name' has to be specified as '=='.

Note:

DBSET has to be called before DBREAD can be used.

DBSET resets reading of the database, i.e. the first call of DBREAD following a call of DBSET always reads the first entry in the database.

Values of attributes of type boolean may be specified as strings whose first character is 'T', 't' (TRUE) or 'F', 'f' (FALSE) or all numbers which are rounded to 1 (TRUE) or 0 (FALSE).

Example:

```
>DI-S Search;%                       define array for access
>$S Search(1)=' nomen   = *M* ';%    define first search-criteria
>$S Search(2)=' address = 97  ';%    define second search-criteria
>T %2 DBSET('devices',2,Search);%    set search and show result
 1
```

```
>
>% second example, defining the same search as in first example
>%
>DI-S Search;%                        define array for access
>$S Name = ' nomen ';%                name of first attribute
>$S StrVal = ' *M* ';%                value of first attribute
>$S Search(1)=Name '=' StrVal;%       define first search-criteria
>SE NumVal = 97;%                     example of numerical value
>$S Search(2)=' address = ' NumVal;% define second search-criteria
>$S Search(3)='';%                    last entry (2 entries defined)
>T %2 DBSET('devices',-1,Search);%    set search and show result
 1
>
```

9 VMS Functions

Interface to VAX/VMS operating system.
These functions constitute the interface to the VAX/VMS operating system and utilities.

9.1 GCPU

Description:
Returns the current value of used CPU time of this NODAL session in seconds.

Type:
VMS RO-Function

Syntax:
`GCPU`

Parameter:

9.2 GETMSG

Description:
Returns the message text associated with the given VAX/VMS message identification.

Type:
VMS RO-Stringfunction

Syntax:
`GETMSG'('mess_id',' index')'`

Parameter:

mess_id
(Real/Integer-ARRAY) Array of message identifications

index
(RealValue) Index of the desired message identification

Note:
Often the construction `TY ! GETMSG(...)` leads to an 'result string filled' error, when the message is too long. Messages longer than 79 characters are truncated. So the construction `TY !; TY GETMSG(...)` will work well.

Example:

```
>DIM-I USTAT(2)
>UFCECACC(...., USTAT)
>T "Completion State: " GETMSG(USTAT,1)
>T "Secondary State: " GETMSG(USTAT,2)
```

9.3 GETMSG1

Description:
Returns the message text associated with the given VAX/VMS message identification. If the function is called with one parameter the parameter is interpreted as 32-Bit OpenVMS message code. If the function is called with two parameters the are interpreted as facilty and error code of the message.

Type:
VMS RO-Stringfunction

Syntax:
```
GETMSG1'('err_msg')'
GETMSG1'('facil',' err_code')'
```

Parameter:

err_msg
(RealValue) OpenVMS message identification (32-Bit code)

facil
(RealValue) Facility code of the message

err_code
(RealValue) Error code of the message (lowest 16 bit)

Note:
Often the construction `TY ! GETMSG1(...)` leads to an 'result string filled' error, when the message is too long. Messages longer than 79 characters are truncated. So the construction `TY !; TY GETMSG1(...)` will work well.

Example:

```
>% the following lines both evaluate the USR completion state
>% of the last Data Module Call
>TYPE "Completion State: " GETMSG1(USRCOMPL('Primarystatus'))
>TYPE "Completion State: " GETMSG1(USRCFACI,USRCSTAT)
```

9.4 HELP

Description:
Calls the HELP Librarian Utility for NODAL.

Type:
VMS Call-Function

Syntax:
```
HELP
```

Parameter:

9.5 LSEDIT

Description:
Invokes the VAX/VMS Language Sensitive Editor LSE to edit the NODAL program currently in memory or a defined function.

Type:
VMS Call-Function

Syntax:
```
LSEDIT ['('def-func-name')']
```

Parameter:

def-func-name
: Name of defined function which should be edited. If not specified, the main program will be edited.

Note:
: The defined function must be defined with command DEFINE first before editing.

Example:

```
>LSEDIT;         % edit main program
>LSEDIT(dfnam);  % edit defined function 'dfnam'
```

9.6 SPAWN

Description:
: Spawns a subprocess running the DCL command interpreter and suspends the running process.

Type:
: VMS Call-Function

Syntax:
: `SPAWN ['('concatenation')']`

Parameter:

concatenation (StringValue) Specifies a command line to be executed by the spawned subprocess before control is given to the terminal. If the parameter is specified, the subprocess ends and NODAL regains control upon the completion of the command. If the parameter is not specified, DCL answers with the `NODAL>` prompt for interactive mode until the user logged off the subprocess.

Example:

```
>% show contents of default directory
>% and return to NODAL
>SPAWN("directory")
>% type out file text.lis and return to NODAL
>$SE cmnd = "ty text.lis"
>SPAWN(cmnd)
>% go to DCL interactive mode
>SPAWN
```

9.7 SYSCANTI

Description:
: The Cancel Timer Request service cancels the Set Timer request previously issued. Cancellation is based on the identification specified in the Set Timer (SYSSETIM) service. If more than one timer request was given the same identification, all requests with that identification are canceled.

Type:
: VMS Call-Function

Syntax:
: `SYSCANTI'('timer_id')'`

Parameter:

timer_id
: (RealValue) Request identification

9.8 SYSCLREF

Description:
: The Clear Event Flag service clears (sets to 0) an event flag in an event flag cluster.

Type:
: VMS Call-Function

Syntax:
: `CLREF'('evtflg')'`

Parameter:

evtflg
: (RealValue) Event flag in the range 0...15. For values > 15, the event flag evtflg modulo 16 is cleared.

9.9 SYSREADE

Description:
: The Read Event Flags service returns the current status of the lowest 16 user settable event flags (0...15) plus the 8 system event flags (16...23) in an event flag cluster.

Type:
: VMS RO-Function

Syntax:
: `SYSREADE`

Parameter:

9.10 SYSSETIM

Description:
: The Set Timer service sets the timer to expire at a specified time. When the timer expires, an event flag is set.

Type:
: VMS Call-Function

Syntax:
: `SYSSETIM'('timer_id',' evtflg',' time')'`

Parameter:

timer_id
: (RealValue) Timer identification.

evtflg
: (RealValue) Event flag to be set when the timer expires. For values > 15, the event flag evtflg modulo 16 is set.

time
: (RealValue) Time in seconds after which the timer expires.

9.11 SYSWFLOR

Description:
The Wait for Logical OR of Event Flags service allows a process to specify a set of event flags for which it wishes to wait. The process is put in a wait state until any one of the specified event flags is set, at which time SYSWFLOR returns to the caller and execution resumes.

Type:
VMS Call-Function

Syntax:
`SYSWFLOR'('evtmask')'`

Parameter:

evtmask
(RealValue) Event flags for which the process is to wait. A bit, when set, selects the corresponding event flag for which to wait. For values $> FFFF_{hex}$, the event flag mask evtmask modulo 10000_{hex} is taken.

9.12 TIME

Description:
Returns the current time of day in seconds.

Type:
VMS RO-Function

Syntax:
`TIME`

Parameter:

Example:

```
>T TIME
    58342.78
>%----- Type Time of Day
>SE T=TIME
>SE HOUR=INT(T/3600)
>SE MINUTE=INT((T/60)-(HOUR*60))
>SE SECOND=T-(MINUTE*60)-(HOUR*3600)
>T "Time is " %2 HOUR ":" %2 MINUTE ":" %5.02 SECOND
Time is 16:12:22.77
```

10 Keyboard Commands

Only special keys that ease the interactive communication with the interpreter are described.

10.1 Uparrow

Description:
Recall a NODAL command line entered before. Maximum 20 lines are stored. The uparrow key steps in the direction to older entered lines.

Type:
Keyboard command

Key(s):
↑

Example:

10.2 Downarrow

Description:
Recall a NODAL command line entered before. Maximum 20 lines are stored. The downarrow key steps in the direction to newer entered lines.

Type:
Keyboard command

Key(s):
↓

Example:

10.3 Leftarrow

Description:
Move the cursor left in the actual NODAL line.

Type:
Keyboard command

Key(s):
←

Example:

10.4 Rightarrow

Description:
Move the cursor right in the actual NODAL line.

Type:
Keyboard command

Key(s):


Example:

10.5 Delete

Description:
Erase the character to the left of the cursor.

Type:
Keyboard command

Key(s):


Example:

10.6 Escape

Description:
Break a running NODAL program and return to immediate mode.

Type:
Keyboard command

Key(s):
ESC or F11

Note:
Ctrl C (^C) and Ctrl Y (^Y) can not be used to stop the execution of a NODAL program.

Example:

10.7 Control B

Description:
If an error occured in a NODAL line, NODAL prints an error message and enters the immediate mode. With Ctrl B (^B) NODAL prints out the erroneous line with the cursor positioned where the error occured.

Type:
Keyboard command

Key(s):
Ctrl and B

Example:

10.8 Control E

Description:
The NODAL default line editing mode is 'expand'. If you move the cursor in a NODAL line to the left and retype some characters, the older ones are moved to the right and the new characters are inserted.
With Ctrl E (^E) you switch into the 'overstrike' mode. Now, if you retype some characters, the old characters are overwritten.
If you are already in overstrike mode, Ctrl E switches back to expand mode.

Type:
Keyboard command

Key(s):
Ctrl and E

Example:

11 Screen Editor

This is the short description of the NODAL own screen editor. The editor is invoked by the command 'EDIT'. When modifiing lines containing more than 79 characters (inclusive line number) they are truncated to 79 characters.
It is also possible to use the more sophisticated VAX/VMS language sensitive editor LSE. Use the command 'LSEDIT'.

11.1 Keypad Commands

```
+-------------------------------------+
! Append ! PF2     ! Expand ! Delete  !
! mode   !         ! mode   ! char    !
+-------------------------------------+
! Move   ! Copy    ! Copy   ! Exit    !
! Lines  ! Lines   ! char   !         !
+-------------------------------------+
! Move   ! Move    ! Copy   ! Restore !
! Backw  ! Forward ! char   ! line    !
+-------------------------------------+
! Get    !         ! Copy   !         !
! again  !         ! char   !         !
+--------------------------!  Home   !
!   ^V - Mark      ! FUNC   !         !
+-------------------------------------+

* cursor steering with arrow keys
* PF2 + PF2          : delete line
* PF2 + 4            : delete from beginning
* PF2 + CR           : delete rest of line
* 'insert'           : insert above line
* ' Find'            : get string
* 'Remove'           : delete lines
* 'Prev/Next Screen': Page
* F8                 : move to Position
* F9                 : renumber lines
* F10                : Substitute string
* F11                : Escape
```

11.2 Home Commands

The HOME commands can be initiated from any screen position by:

```
[FUNC] <command>
```

Commands that are accepted in the HOME position are:

```
C        Copy lines <source lines> to <position>
D        Delete Lines <source lines>
E        Exit
F        Display the first window
G        Get string <sting>    ( search )
^G       Get the next occurrence of string
H / ?    Help
```

```
  L / $    Display the last window
  M        Move lines <source lines> to <position>
  N        Display next window
  P        Display previous window
  R        Renumber lines <group> with <interval>
  S        Substitute <old string> with <new string>
  +        Advance the displayed buffer by 5 lines
  -        Rewind the displayed buffer by 5 lines
space / 0  Re-display the current window
  .        Move to <position>
  ^E       Set/Reset expand mode (insert character mode)
  ^B       Set/Reset append mode (automatic insert lines)
  ^W       Repeat last command
```

11.3 Control Key Commands

```
  ^A       Delete this character or the previous if the cursor
           is positioned at the end of the line
  ^B       Set/Reset append mode
  ^C       Copy one character from previous line to current
  ^D<char> Delete characters up to and including <char>
           <char>= ^D : delete entire line
           <char>= (CR) : delete the rest of the line
           <char>= ^R   : delete from beginning of the line
  ^E       Set/Reset expand mode
  ^F<char> Move cursor forward to spec. character on line.
           <char>= ^F or (CR): Move beyond last
           significant character
  ^G       Find next occurrence of the string specified in
           the last GET STRING command
  ^L       Insert a new line above the current line
  ^N       Copy one character from the next line to current line
  ^O<x>    Accept control character, (displayed as ^x)
  ^P<char> Copy characters from previous to current line up to
           and including <char>
           <char>=^P : copy to end of line
           <char>=(CR):copy to end of line and go to next line
  [FUNC]^P<char> Same as the above, but characters are copied
   from the next line
  ^R<char> Move cursor backwards to char on the line
           <char> = ^R : move to the beginning of text in line
           <char> =(CR): Move to the very beginning of the line
  ^W       Restore last deleted line at cursor position.
   As HOME command: repeat previous command
  (DEL)    Delete the previous character
```

12 Summary of NODAL-Errors

This is a list of all possible NODAL-errors and their corresponding error numbers.

No.	Text	No.	Text
0	No error	1	Illegal line number
2	Illegal format specifier	3	Illegal arithmetic expression
4	Ambigious command	5	Illegal delimiter
6	Attempt to divide by zero	7	Working area full
8	Nonexistent name	9	Wrong variable type
10	Link resources exhausted	11	Command not properly terminated
12	Unallocated error	13	Nonexistent line addressed
14	Illegal shuffle attempted	15	Error in IF command
16	Escape typed	17	Illegal edit command
18	Illegal ASK command	19	Erase error
20	Argument list error	21	File error
22	Error in SAVE command	23	Array dimension error
24	Square root of negative number	25	Illegal arctangent argument
26	Sine argument too big	27	Cosine argument too big
28	Power error [negative argument?]	29	Power underflow
30	Exponential argument too big	31	Logarithm argument <= 0
32	Device not connected	33	Unauthorised action
34	Hardware error	35	Illegal equipment number
36	Illegal property	37	Value out of range
38	Not implemented	39	No such computer
40	Result string filled	41	Syntax error
42	No such file	43	File already exists
44	No file space	45	Link not open
46	Remitted data lost	47	End of file
48	Equipment error	49	SIOM error
50	Illegal error number	51	Checksum error
52	Defined function area full	53	Syntax error in DEFINE command
54	Illegal string in SET command	55	String function failure
56	Illegal concatenation	57	Error in $IF command
58	Error in $ASK command	59	String expected
60	Pattern too big	61	Bad pattern match
62	Bad pattern	63	Bad pattern assignment

... continued on next page

No.	Text	No.	Text
64	Indirection signal	65	not used
66	not used	67	not used
68	To many nested DO	69	not used
70	not used	71	Unknown terminal
72	Channel transfer error	73	Breakpoint found
74	Error during DMS call	75	Error during USERFACE call
76	Error in USR response	77	Error in DBS functions

13 General Info

13.1 History

NODAL was designed in the beginning 70's at CERN in Geneva as a flexible and easy to handle tool for controlling the SPS accelerator.
In NODAL features of FOCAL and SNOBOL-4 (with some influence of BASIC) are combined. The first implementation was written in assembler for the NORD-10 minicomputer, now an assembler version for the NORD-100 exists too. Many of theese computers are interconnected in a homogeneous network. Each of these nodes may be used to execute NODAL programs whereby exchange of NODAL statements and NODAL data between the different computers is possible.
In the control of the new accelerator LEP other processors than for the SPS are used for which the existing assembler implementations of NODAL cannot be adapted easily. Since NODAL was found to be very usefull at testing, during running-in and in all situations where flexible response was required, it was desirable to have NODAL also availible for LEP. To overcome the limitations of assembler programs when adapting them to different processors it was decided to create a new version, written in a portable language. The choice was MODULA-2, since for this language compilers are availible for all computers used.
When designing the control system for the new GSI accelerators, the Prozessrechnergruppe was looking for an interpreter for accelerator control. Because of the portability of the new NODAL version it was decided to coorperate with CERN and use NODAL also at GSI.

13.2 Characteristics of NODAL

Contrary to other interpreters NODAL is emphasized by the following characteristica:

- integrated elements for access to equipment, called data-modules.
- parts of programs can be exported to run on another computer and results of calculations can be recieved (not implemented in GSI-version).
- output of programs and data to files (and also to other computers) is formatted uniformly. Thus it is possible to store programs and data in one file.

Special features of the MODULA-2 version of NODAL are:

- The separation into kernel modules (processor independend) and target modules (processor dependend) eases the adaption to different types of processors. When transfering NODAL to another processor, only the target modules like file input/output, error handling, operating system access etc. have to be modified. Actually modules exist (or are under work) for MC68000, VAX/VMS, VAX/UNIX, NORD-100, LILITH, TMS9900, IBM-PC.
- Execution is faster than in the previous assembler versions. This is achieved by generating an intermediate code which is used after first execution of a program. Thus lexical and syntax analysis has to be done only once which speeds up repitative parts of NODAL programs. To calculate arithmetic expressions it is planned to install a 'throw-away' compiler.

14 Version Guide

This entry gives a little help for detecting new features in new NODAL Versions. Operational versions are released for GSI-wide usage, development versions are not released and can be changed without special information. Development versions are the predecessors of operational versions if they have the same version number.
The Version of the interpreter in use should be kept in mind when reading the BUGLIST, which you can find in the HELP service. The actual version of the running interpreter is displayed in the first line after startup.
Fixed bugs are not referenced in this Version Guide.
If not noted contrary, implemented features can be found in higher versions too.

14.1 Version GSI 0.7

First released version.
History:
Version developed from the actual CERN Development Version from 24-MAY-1985 with frame for indirections. Version is linked with a VT240-based NodalGraphic module. This one is not free of Errors and should not be used seriously. Some features for Defined Functions are included, but recursive calls causes a crash.
Implemented Commands:

```
SET     TYPE    IF      GOTO    FOR     ROF     WHILE
DO      RETURN  $MATCH  $PATTER ASK     CALL    $ASK
$IF     $DO     RUN     SAVE    OLD     LOAD    ERASE
DIM     LIST    ?ON     ?OFF    EDIT    END     COMMENT
QUIT.
```

Implemented new features:

- Screen Editor for VT200

Not implemented standard features:

- Defined functions
- Indirect referencable DataObjects (Variables, Arrays, Functions etc.)

14.2 Version GSI 0.8

History:
Version based on the CERN Development Version from 17-AUG-1985.
Implemented Commands:

```
ASK     $ASK    CALL    %(COM) DEFINE DIM      DO      $DO     EDIT
END     ERASE   FOR     GOTO    IF      $IF     LIST    LOAD    $MATCH
OLD     ?ON     ?OFF    OPEN    $PAT    QUIT    RETURN RUN      ROF
SAVE    SET     $SET    TYPE    VALUE   $VALUE  WHILE
```

Additional implemented features:

- Name Indirections
- Defined Functions incl recursions
- Nodal name now 8 Characters long (not NODAL standard !)

- New procedure to get USR-Status after DMS calls (USRSTAT)
- Interface to GKS

Not implemented features:

- Standard NODAL graphic functions: only a subset is implemented, the other functions can be easily substituted by GKS calls. The full NODAL Graphic functions are foreseen to be implemented with Version 0.9.

14.3 Version GSI 0.9

Version, based on version 0.8.
Additional implemented features:

- Database interface
- New procedures USRCSTAT, USRSSTAT for accessing error information of Userface calls
- Connections now possible with Userface interface
- Accessing VAX/VMS message utility by GETMSG/GETMSG1 ROStrFunction
- An unknown NODAL name is always tested on SIS device nomenclature

14.4 Version GSI 1.0

Actual operating version, based on version 0.9 and the new delivered CERN version. Released on 29. May 1989.
Implemented Commands:

```
ASK     $ASK    BKPT    CALL    DEFINE  DIMENS  DO    $DO
EDIT    END     ERASE   FOR     GOTO    IF      $IF   LDEF
LIST    LOAD    $MATCH  ?ON     ?OFF    OLD     OPEN  OVERLA
$PATTE  QUIT    RETURN  ROF     RUN     SAVE    SDEF  SET
$SET    TYPE    VALUE   $VALUE  WAIT-T  WHILE   ZDEF
```

Implemented Functions:

```
AT2     COS     EXP     FPT     INT     LOG     MOD     SGN
SIN     SQR     AND     BIT     IOR     NEG     SHIFT

ASCII   CAP     EVAL    FIND    FINDS   SIZE    SORT    STRARG
SUBS

ABORT   ANY     ARB     BREAK   FAIL    LEN     NOTANY  POS
RPOS    RTAB    SPAN    TAB

ALPHA       ARG         ARSIZE      BRKPT       COPY
ERMES       ERROR       LISD        LISR        LISV
LSTBRK      MAX         MIN         NODLIN      NUM
UNBRK       UPGRADE

CLOSE       INPBT       INPC        INPUT       OPEN
OUTBT       OUTC        OUTPUT
```

```
EQMEVENT    EQMTIME     LISDM       LSTDM       LSTNOMEN
USRCOMPL    USRCFACI    USRCSTAT    USRESTAM    USRSFACI
USRSSTAT    USRSTAT     USRTSTAM    VRTACC

UFCBI       UFCCI       UFCCLOSE    UFCCONN     UFCDCON
UFCEQACC    UFCFACIL    UFCLSTEX    UFCOPEN     UFCPURGE
UFCSDAT     UFCREAD     UFCRESET    UFCWAIT

DBSACC      DBSACC1     DBSCLOSE    DBSDEF      DBSDSET
DBSGET      DBSOPEN     DBSTABLE

GCPU        GETMSG      GETMSG1     HELP        LSEDIT
SPAWN       SYSCANTI    SYSCLREF    SYSREADE    SYSSETIM
SYSWFLOR    TIME
```

Additional implemented features:

- Lowercase letters.
- New 64 bit real format.
- Inputline buffer to recall up to 20 entered inputlines
- 'Expand' is the default line editing mode.
- New procedures USRCFACI and USRSFACI for accessing the facility codes of the completion and secondary state information of Userface calls.
- Additional VMS functions.
- New function UPGRADE to run Version-0.9-programs with Version 1.0.
- New file-i/o functions OPEN, CLOSE, INPBT, INPC, INPUT, OUTBT, OUTC and OUTPUT.
- NODAL Debugger
- New function CAP to change lowercase to uppercase letters.
- Entirely redisigned interface for database access.
- Additional features to format output of numerical data
- Rounding instead of truncation in output of numerical data

Not implemented features:

- GKS functions are no more available in NODAL.

Literatur

[1] K. H. Lundberg. *NODAL Primer*. SPS/ACC/Note/85-27.

[2] P. D. V. van der Stok. *NODAL Reference Manual*. SPS/ACC/Note/85-33.

[3] M. C. Crowley-Milling and G. C. Shering. *The NODAL System for the SPS*. CERN 78-07.

[4] A. Lacroix. *NODAL Debugger User's Guide*. LEP Controls Note 78. SPS/ACC/Note/86-23.

[5] H. Hübner. *Gerätezugriffe mit NODAL*. SIS/ESR Controls Documentation U-NOD-01.

[6] E. Schaffner. *REDABAS: Realtime Database System for SIS Controls*. SIS/ESR Controls Documentation B-DBS-01.

Index