rho 3

BAPS2
Brief description

Version 105

BOSCH
Automationstechnik

rho 3

BAPS2
Brief description

1070 073 035-105 (94.12) GB

Reg. Nr. 16149-03

Discretionary charge 10.– DM

1. BAPS2

2. BAPSPIC

GENERAL

This manual provides a summary and brief description of the language instructions available in **BAPS2** and **BAPSPIC**.

Please refer to the **BAPS2** programming instructions for an in−depth description of the individual instructions of **BAPS2**.

PROGRAM STRUCTURE

```
;;CONTROL = RHO3

;;KINEMATICS:   (  1 = ma_1 ,
                   2 = ma_2 )

;;ma_1.JC_NAMES   =  A_1,A_2,A_3,A_4
;;ma_1.WC_NAMES  =  X_K,Y_K,Z_K,A_K

;;ma_2.JC_NAMES   =  B_1,B_2,B_3
;;ma_2.WC_NAMES  =  X_B,Y_B,Z_B

PROGRAM example

;;KINEMATICS = ma_1



EXTERNAL: maon, maoff

INPUT:      1 = partthere,
            2 = slide_forwar

OUTPUT:     1 = belt_on

SPC_FCT:    1 = path_io ( VALUE INTEGER:
                          .
                          .
                          .  )




ma_1.POINT       : pal_corner1,pal_corner2
ma_2.JC_POINT    : @stroke_cent

BEGIN
   .
   .
   MOVE ma_2 TO @stroke_cent
   sub_example
   .
   .
   .
PROGRAM_END

SUBROUTINE sub_example
      INPUT: 4 = le_shi
      REAL: counter

      BEGIN
      .
      belt_on = 1
      .
SUB_END
```

Control type declaration

Number of kinematics with kinematic names (as defined in MPP).

Number of axes and axis names in JC for Kin 1. Number of axes and axis names inWC for Kin 1
Number of axes and axis names in JC for Kin 2. Number of axes and axis names inWC for Kin 2
Program name max. 8−character

All movement instructions and point entries without kinematic entry relate to the kinematic ma_1 (default kinematic).

Declarations for external subroutines or processes.
Declarations for inputs/outputs

Declaration of the special functions

Order of other variable declarations is arbitrary, but they must be in front of **BEGIN**.

Declaring points for different kinematics.

Main program begin

Subroutine call

Main program end

Subroutine identification (max. 12−character)
Declaration part for the subroutine

Subroutine begin

Subroutine end

COMPILER STATEMENTS

Except the instructions **KINEMATICS**, **INCLUDE** and **INT** all compiler instructions must be declared before the program name. Lines containing compiler instructions should have no other characters following the statement.

;;CONTROL = rho3	applies as of Version-TO01E	Control type declaration currently available: rho3, IQ120, IQ140
;;DEBUGINFO+ **;;DEBUGINFO –**	applies as of Version TO01I	Compiler statement **DEBUGINFO +** permits programs to be debugged with the debug system. In the case of **DEBUGINFO –** , debugging is not possible but the IRD file is smaller and is executed more quickly(default: DEBUGINF0+).
;;WARNING+ **;;WARNING –**	applies as of Version TO01I	Compiler statement **WARNING –** suppressess output of warnings to the error file (default: WARNING +).
;;ERROR = 10	applies as of Version TO01I	Consequential errors frequently occur during the compiling process, thus very greatly increasing the size of the error file. By defining the maximum number of errors (1...100), this statement can be used to abort the compiling process prematurely when the specified number of errors have been detected. All errors are output without this statement.
;;KINEMATICS: (1=ma_1, 2=ma_2)	applies as of Version TO01E	Defines the number of kinematics with the relevant kinematic numbers and kinematic names.
;;ma_1.JC_NAMES= A_1,A_2	applies as of Version TO01E	Defines the number of axes and axis names in JC for Kin1. **(!!! This statement must be in one line)**
;;ma_1.WC_NAMES= X_C,Y_C	applies as of Version TO01E	Defines the number of axes and axis names in WC for Kin1. **(!!! this statement must be in one line)**
;;PROCESS_KIND=PERMANENT	applies as of Version TO01E	Defining that this program can run as a permanent process. A permanent process runs in operating modes Auto and Manual and is not aborted with Reset etc. **(!!! this statement must be in one line)**

;;KINEMATICS = ma_2	applies as of Version TO01E	Changes the default kinematic(the default kinematic is always no. 1 unless otherwise defined in this declaration. **(!!! this statement must be in one line)**
;;DRIVE_TYPE (sr_1.A_1 = SM, sr_2.A_1 = SM_CAN)	applies as of Version "ECO"	The drive type is defined. The statement is skipped by the BAPS 2 compiler. It only affects the BAPS−ECO compiler for the BOSCH IQ120 control.
;;INCLUDE exdat	applies as of Version TO01E	The compiler includes file EXDAT.QLL at this point and generates the IRD code for it. EXDAT.QLL may have no other INCLUDE statements.
;;INT = PTP ;;INT = LINEAR ;;INT = CIRCULAR	applies as of Version TO01E	Interpolation mode presetting The interpolation mode can be defined by the compiler statement in a program. The default is PTP.
;;ma_2.INT = PTP ;;ma_2.INT = LINEAR ;;ma_1.INT = CIRCULAR	applies as of Version TO01E	Ditto with kinematic assignment
;;SER_IO_STOP−	applies as of Version TO02F	The compiler statement can be used to prevent a user program being aborted if an interface error occurs.
;;SER_IO_STOP+	applies as of Version TO02F	The compiler statement indicates that a user program is to be aborted if an interface error occurs.The default is SER_IO_STOP+.

PROGRAM DECLARATIONS

```
PROGRAM dana
.
.
.
```

applies as of Version TO01E

MAIN PROGRAM DECLARATION
The program name may have a maximum length of 8 characters. The first character must be a letter. The only special character permitted is the underscore character "_".

```
BEGIN
.

PROGRAM_END
```

applies as of Version TO01E

MAIN PROGRAM BEGIN−END
The declaration part is followed by the start of the main program with instruction BEGIN. It ends with the keyword "PROGRAM_END".

```
BEGIN
.
HALT
PROGRAM_END
```

applies as of Version TO01E

MAIN PROGRAM−END BY HALT
HALT is optional and may occur at several points in the main program.
Program execution is stopped (e.g. in the case of decisions IF...THEN...).

```
EXTERNAL: subrout_name
.

subrout_name
```

applies as of Version TO01E

EXTERNAL SUBROUTINE CALLS
The names of the external subroutines must be declared before the keyword "BEGIN".

```
EXTERNAL: subrout_name
 (VALUE INTEGER : sp,ze,
  VALUE REAL : de_z)

subrout_name (3,2,−50)
```

applies as of Version TO01E

CALLING EXTERNAL SUBROUTINES
with parameter transfer.

```
PROGRAM subrout_name
        (VALUE INTEGER: sp,ze
        VALUE REAL:     de_z)
```

applies as of Version TO01E

PROGRAM DECLARATION
with parameter transfer.
Can be called as an external subroutine.

SUBROUTINE DECLARATIONS

```
SUBROUTINE subrout_name
BEGIN
.
SUB_END
```

applies as of Version TO01E

SUBROUTINE DECLARATION
The subroutine name may have a maximum length of 12 characters. The first character must be a letter. The only special character permitted is the underscore "_".

```
SUBROUTINE subrout_name
    (VALUE REAL:aw)
BEGIN .....
SUB_END
```

applies as of Version TO01E

SUBROUTINE DECLARATION with parameter-list. If variable 'aw' is modified in subroutine SUBROUTINE_NAME, variable 'oh' is unchanged after the return (see below).

```
SUBROUTINE subrout_name
    (REAL:aw)
BEGIN .....
SUB_END
```

applies as of Version TO01E

SUBROUTINE DECLARATION with parameter-list. If variable 'aw' is changed in subroutine SUBROUTINE_NAME, variable 'oh' has the same value after the return (see below).

```
.
subroutine name
.
.
```

applies as of Version TO01E

SUBROUTINE CALL
A subroutine call can be performed from the main program. A subroutine, in turn, can call another subroutine.

```
.
subroutine name (oh)
.
.
```

applies as of Version TO01E

SUBROUTINE CALL
with parameter transfer.

```
RETURN
```

applies as of Version TO01E

SUBROUTINE–END
This can be used optionally if you wish to quit the subroutine at various points.

STANDARD CONSTANTS

```
WRITE PHG,CLS
```

applies as of Version TO02F

CLEAR SCREEN CLS
You can clear the PHG display by output of ”CLS”.

```
IF VERSION  < 2.31
THEN WRITE
  'Old compiler version'
```

applies as of Version TO02F

VERSION
You can scan the version number of the compiler in the BAPS program.
The constant is of the type REAL.

DATA TYPES

POINT: p_name **JC_POINT: @p_name**	applies as of Version TO01E	Points in world coordinate representation. Points in joint coordinate representation.
ma_1.POINT: kp_name **ma_2.JC_POINT: @kp_name**	applies as of Version TO01E	Ditto with kinematic assignment.
INTEGER: i_var **REAL: r_var**	applies as of Version TO01E	Integer values (whole numbers) Decimal (real) floating point
BINARY: b_var	applies as of Version TO01E	Binary variables or I/O signals. There are two states (logical 0 or logical 1).
CHAR: char **TEXT: message**	applies as of Version TO01E	ASCII characters in accordance with DIN 66003 Character string, e.g. 'SCHOOL'
FILE: danadat	applies as of Version TO01E	This is where the names of the files with extension DAT are declared. Values can be read from or written to DAT files. (See FILES I/O.)
SEMAPHORE: sema_var	applies as of Version TO01E	Variables of type SEMAPHORE are required for the EXCLUSIVE statement.

POINT DECLARATIONS

DEF POINT: d_pnt	applies as of Version TO01E	DEFine points are points stored in a point file. They can be influenced in operating mode 'DEFINE', 'TEACH' and by the BAPS program.
DEF JC_POINT: @d_jcp	applies as of Version TO01E	DEFine joint coordinate points. (see above)
DEF ma_1.POINT: d_pnt **DEF ma_1.JC_POINT: @d_pnt**	applies as of Version TO01E	Ditto with kinematic assignment.
DEF ARRAY [1..6] POINT:ap	applies as of Version TO01E	DEFine world coordinate points via an array with lower limit and upper limit.
DEF ARRAY [1..6] **JC_POINT: @ap**	applies as of Version TO01E	DEFine joint coordinate points via an array with lower limit and upper limit.
DEF ARRAY [1..6] **ma_1.POINT: ap** **DEF ARRAY [1..6]** **ma_2.JC_POINT: @ap**	applies as of Version TO01E	Ditto with kinematic assignment.

GLOBAL VARIABLES

Global variables permit simple data exchange between several independent BAPS2 user programs (processes). The underlying idea is that of combining programs operating with the same global variables to form one program group. One program can export data within this group while the other programs of this group may only import these data. Any number of such program groups can be created on the control (restricted only by the available memory space).

Example program

```
;Exporting program
PROGRAM exp_var

;global data
PUBLIC DEF POINT: start_pos
PUBLIC INTEGER: index
PUBLIC
SEMAPHORE: writeprotect

REAL: mvalue

BEGIN
;BAPS statements
EXCLUSIVE writeprotect
start_pos =POS
index=5
EXCLUSIVE_END
;...
;... further statements
;...
PROGRAM_END



;Importing program
PROGRAM imp_var

;global data
EXTERNAL  exp_var: start_pos
EXTERNAL exp_var: index
EXTERNAL  exp_var:  writepro-
tect

;local data
POINT: end_pos

BEGIN
;BAPS statements
EXCLUSIVE writeprotect
RPT index TIMES

MOVE TO start_pos
MOVE TO end_pos
RPT_END
EXCLUSIVE_END
;
PROGRAM_END
```

applies as of Version TO02F

Declaration part in the **exporting program**:

Global variables are declared by adopting the reserved word PUBLIC in the type declaration of these variables. The data range for these variables is reserved in the IRD file. Teach points (e.g. identified by the keyword DEF) are stored in the PNT file.

Declaration part in the **importing program**:

The variables can be accessed by other BAPS programs if the variables are declared with EXTERNAL and the name of the exporting program.

Restrictions

The following restrictions must be taken into consideration when using global data:

1. The exporting program of a program group must have been compiled before the importing programs of this group (because of the type checking of the compiler), i.e. the user must ensure that the importing programs are 'more recent' than the exporting program. If this is not the case, an error message or a warning is issued at the start of the program.

2. In a program, it is not possible to import and export data simultaneously.

3. Data may be imported only from one program.

4. Global data may be exported or imported only in the declaration part of the main program, and not in the subroutines.

5. Access to the variables must be prevented by use of the EXCLUSIVE statement before interruption in order to guarantee data consistency. The consistency of simple standard types such as BINARY, INTEGER, REAL, CHAR is provided by the operating system.

ARRAY DECLARATIONS

Arrays are data structures consisting of a fixed number of elements of the same type. The lower and upper limits are specified in square brackets. The data type is specified after the square brackets. The array limits are monitored for the following range when declaring arrays of any element type: [− 8388608... 8388607].

```
ARRAY [1..6] POINT:  ap

ARRAY [1..6] JC_POINT: @ap
```

applies as of Version TO01E

Array of type POINT/JC_POINT

```
ARRAY [1..6]
          ma_2.POINT: ap
ARRAY [1..6]
       ma_1.JC_POINT: @ap
```

applies as of Version TO01E

Ditto with kinematic assignment

```
ARRAY [1..6] CHAR: az

ARRAY [21..36] TEXT: at
```

applies as of Version TO01E

Array of type CHAR and TEXT

```
ARRAY [1..6] INTEGER: ai

ARRAY [21..36] REAL: ar
```

applies as of Version TO01E

Array of type INTEGER and REAL

```
ARRAY [1..6] BINARY: ab
```

applies as of Version TO01E

Array of type BINARY

```
ARRAY [1..4]
 ARRAY [1..2] INTEGER: a2dimi

a2value = a2dimi [1] [2]
```

applies as of Version TO01E

Two−dimensional array of type INTEGER

Access to the field components

I/O ARRAY DECLARATIONS

ARRAY [1..2] **OUTPUT BINARY:** **(1,32) = o_sen_F1**	applies as of Version TO01E	Array of type OUTPUT BINARY
ARRAY [11..14] **INPUT BINARY:** **(5,6,7,22) = i_sen_F2**	applies as of Version TO01E	Array of type INPUT BINARY
ARRAY [1..3] **OUTPUT INTEGER:** **(401,402,403) = oi_sen_F3**	applies as of Version TO01E	Array of type OUTPUT INTEGER
ARRAY [11..13] **INPUT INTEGER:** **(401,404,408) = ii_sen_F4**	applies as of Version TO01E	Array of type INPUT INTEGER
ARRAY [1..2] **OUTPUT REAL:** **(201,202) = or_sen_F5**	applies as of Version TO01E	Array of type OUTPUT REAL
ARRAY [11..12] **INPUT REAL:** **(201,203) = ir_sen_F6**	applies as of Version TO01E	Array of type INPUT REAL

I/O arrays may be declared only in one dimension.

CHANNEL DECLARATIONS

INPUT BINARY: 2 = gr_force

OUTPUT BINARY: 9 = magnet

applies as of Version TO01E

The names of the USER INPUTS/OUTPUTS are declared. The names may not be BAPS keywords. Entry **"BINARY"**may be omitted since BINARY is the default type.
(Permitted channel number: 1..120, 6xy with high speed inputs)

Structure of the high speed input number:

6 x y

- y: Number of the high speed input
  - 0 = Probe input
  - 1..9 = high speed inputs
- x: Number of the Servoboard (1..3)
- 6: characteristic for probe and high speed inputs

INPUT INTEGER:
401 = decade
OUTPUT INTEGER:
403 = lcd_display

applies as of Version TO01E

The names of the INTEGER USER INPUTS/ OUTPUTS are declared. These are output on the interface with a width of 8 BITS (significance 0−255).
(Permitted channel number: 401..408)

INPUT REAL: 201 = poti_1

OUTPUT REAL:203 = v_pressure

applies as of Version TO01E

The names of the REAL USER INPUTS/OUTPUTS are declared.
(Permitted channel number: 201..299)

BELT: 501 = belt1_ma1

ma_2.BELT: 502 = belt2_ma2

applies as of Version TO01E

The names of the belts are declared.
(Permitted channel number: 501..508)

Asynchronous inputs:

The declaration of asynchronous inputs is made by adding an offset of **1000** to the channel number.

INPUT BINARY:1002 = gr_force

applies as of Version TO01E

The names of the asynchronous INPUTS are declared. The names may not be BAPS keywords. Entry "BINARY" may be omitted since BINARY is the default type.
(Permitted channel number: 1001..1020)

INPUT INTEGER:1401= decade

applies as of Version TO01E

The names of the integer−asynchronous INPUTS are declared.
(Permitted channel number: 1401..1408)

INPUT REAL:1201= poti_1

applies as of Version TO01E

The names of the real−asynchronous INPUTS are declared.
(Permitted channel number: 1201..1299)

PROGRAM FLOW CONTROL

```
WAIT 5.5
```

applies as of Version TO01E

Program execution is stopped for the specified period. The constant (5.5) indicates the time in seconds.

```
REAL:    dwell_time
       dwell_time = 5.5

WAIT dwell_time
```

applies as of Version TO01E

A variable may also be used instead of a constant. This variable must have been declared beforehand as a REAL or INTEGER variable. This variable must be assigned a value before it is used.

```
INPUT: 4 = part_there
WAIT UNTIL part_there = 1
```

applies as of Version TO01E

WAIT for a condition to occur. (Only scanning inputs is permitted in the condition.)

```
INPUT:   5 = sig
   WAIT UNTIL sig = 0
   MAX_TIME = 5.6
   ERROR statement1
```

applies as of Version TO01E

WAIT for a condition to occur with a time limit. If the time has elapsed, the instruction ERROR statement1 is executed. If the condition is fulfilled in the max. time, the error is skipped and the next BAPS instruction is executed.

```
PAUSE
```

applies as of Version TO01E

Program execution is stopped. The program is continued with the interfacesignal “RC–START”

```
REPEAT 5 TIMES
.
.
REPEAT_END
```

applies as of Version TO01E

A fixed number of program runs is executed between the two instructions.

```
INTEGER: number
        number = 5
REPEAT number TIMES ...
REPEAT_END
```

applies as of Version TO01E

A fixed number of program runs is executed between the two instructions.

```
IF condition
   THEN statement1
   ELSE statement2
```

applies as of Version TO01E

CONDITIONAL STATEMENT
If the condition is fulfilled, the control executes statement 1. If the condition is not fulfilled, statement 2 is executed (after ELSE).

```
IF in1 = 1 THEN
    BEGIN
     out36 = 0
     out01 = 1
    .
    END
ELSE BEGIN
          .
          END
```

applies as of Version TO01E

BEGIN ... END compound
Several statements can be combined by means of compound statements at points at which only one single statement may occur.

```
JUMP label_1
    .
    .
label_1:
```

applies as of Version TO01E

It is possible to jump to labels within main programs or subroutines. Forward and backward jumps are possible.

```
INTEGER: magnitude,offset
BEGIN
magnitude=0
offset=0

 READ magnitude
  CASE  magnitude
   EQUAL  9,10 : offset=0
   EQUAL 11,12 : offset=1
   EQUAL 13 : offset=2
   DEFAULT  offset=5
  CASE_END
.
.

PROGRAM_END
```

applies as of Version TO03B

Using the branch statement ”CASE”, it is possible to implement a selection from several alternatives. Nested ”IF – THEN ” scans can thus be avoided and the program execution time can be shortened.

LOGIC – ARITHMETIC OPERATIONS

+ – * /	applies as of Version TO01E	Arithmetic operations (addition, subtraction, multiplication, division)
SIN **COS**	applies as of Version TO01E	Standard functions (Sine, cosine)
ATAN	applies as of Version TO01E	Standard functions (arc tangent)
SQRT	applies as of Version TO01E	Standard functions ($\sqrt{x}$)
MOD	applies as of Version TO01E	Arithmetic operations (Modulo calculation corresponds to: dividing a number and using only the remainder)
AND **OR** **NOT**	applies as of Version TO01E	Logic operations
< , > <= , >= <> =	applies as of Version TO01E	Comparison operations (less than, greater than, equal to or less than, equal to or greater than not equal to, equal to)

VALUE ASSIGNMENTS

```
variable = expression
```

applies as of Version TO01E

An assignment assigns a new value to a variable.
This value must have a type compatible with the variable type.

```
p1 = (0,0,−50.123,0)

@p1 = @(0,0,−19.4,0)
```

applies as of Version TO01E

POSITION ASSIGNMENT.
Complete value assignment of a world and joint coordinate point. The number of components is equal to the number of axes of the kinematic.

```
REAL: z_value

z_value = −50.0
    p1 = (0,0,z_value,0)
```

applies as of Version TO01E

Value assignment of a point via variables.

```
z_axis.Z_C = 31.2

p1 = (0,0,z_axis.Z_C,0)
```

applies as of Version TO01E

AXIS ASSIGNMENT
The axis name is used for the component value assignment.

```
INTEGER: i

   i = 0
   i = i+1
```

applies as of Version TO01E

The value assignment can be used for all data types (e.g. REAL, INTEGER, etc.) and for all standard variables (e.g. V_PTP, A, VFACTOR, AFACTOR etc.).

```
CONST:
  yellow = 0, white = 1,
  blue = 4, red = 3
INTEGER : color
```

applies as of Version TO03B

CONSTANTS.
Constants are defined before the variables of a program.

STANDARD FUNCTIONS

```
INTEGER:   i1
REAL:      r1

r1 = TRUNC (12.58)
i1 = TRUNC (1.32)
```

applies as of Version TO01E

Truncating the positions after the decimal point with assignment of an INTEGER number or varible or REAL number or variable.

```
INTEGER:   i1, it
REAL:      r1, rt

rt = ABS (r1)
rt = ABS (i1)
rt = ABS (i1)
```

applies as of Version TO01E

Absolute value generation of INTEGER or REAL variables or numbers.

```
REAL:      r1
INTEGER:   i1

r1 = ROUND (10.51)
i1 = ROUND (1.32)
```

applies as of Version TO01E

Assignment of a REAL number or variable to an INTEGER or REAL variable with rounding.
(> 0.5 is rounded up,
< = 0.5 is rounded down).

```
INTEGER:   varo1


varo1 = ORD ('a')
```

applies as of Version TO02A

Standard function ORD converts an ASCII character to an INTEGER value
(in this example:
varo1 = 97).

```
INTEGER: varo2, varo3
CHAR: c_var
ARRAY[1..8] CHAR: txt

txt = 'pRima___'
varo2 = ORD (txt [2])

c_var = 'a'
varo3 = ORD (c_var)
```

applies as of Version TO02A

ORD with arrays
(in this example:
txt[2] = 'R'
varo2 = 82).

ORD with character
(in this example:
varo3 = 97).

```
INTEGER: varc1

varc1 = CHR (97)
varc1 = CHR (80 + 17)
```

applies as of Version TO02A

Standard function CHR converts an INTEGER value to an ASCII character. The displayable character depends on the character set used
(in this example: varc1 = 'a').

```
CHAR: c_varc2
ARRAY[1..3] INTEGER: ain-
teger

ainteger [2] = 97

c_varc2 = CHR(ainteger[2])
```

applies as of Version TO02A

The value range of CHR (X) should be between 0 and 255. No range check is conducted. The function argument is converted as follows internally for values outside this range:
X = (X MOD 256)
(in this example: c_varc2 = 'a')

```
INTEGER: int_number
INTEGER: index_ca
INTEGER: length_char
INTEGER: error_rue
ARRAY[1..10]
       CHAR: ascii_array



int_number = −1234
index_ca = 1
length_char = 5



INT_ASC
    (int_number,
     ascii_array,
     index_ca,
     length_char,
     error_rue)



IF
    error_rue <> 0
THEN
    WRITE 'error in
    INTEGER−ASCII conver-
sion'
ELSE
    WRITE ascii_array
```

applies as of Version TO02A

The standard procedure **INT_ASC** converts an integer number to an array of characters.
Note:
The format is right justified with leading blanks.
If an error is detected during conversion process, the 'character array' remains unchanged and the corresponding error code is returned.

−−−−−−−− PRESET −−−−−−−−
int_number = number to be converted
(only type INTEGER permitted
| int_number | < =2147483647)

−−−− RESULT ACKNOWLEGMENT −−−
ascii_array = Result array of
Type ARRAY [ ] char
(The destination area is initialized with blanks before conversion)

−−−−−−−− PRESET −−−−−−−−
index_ca = Start index in the character array

−−−−−−−− PRESET −−−−−−−−
length_char = Maximum number of charac ters reserved for the number to be converted

−−−− RESULT ACKNOWLEDGMENT −−−
error_rue = Error number output

0	**No errors**
−1	**Start index outside array limits**
−2	**End index (start index + length) outside array limits**
−3	**Reserved length too small**
−4	**Range transgression (value too big)**
−5	**Array length < 0**
−6	**Array length = 0**

```
INTEGER: int_number
INTEGER: index_ca
INTEGER: length_char
INTEGER: error_rue
ARRAY [1..10]
       CHAR: ascii_array



ascii_array = 'AaBc'
index_ca = 1
length_char = 5



ASC_INT
    (ascii_array,
     int_number,
     index_ca,
     length_char,
     error_rue)



IF
    error_rue <> 0
THEN
    WRITE 'error in
    ASCII-INTEGER conver
    sion'
ELSE
    WRITE int_number
```

applies as of Version TO02A

The standard procedure **ASC_INT** converts an array of characters to an integer.
The procedure reads in characters as of the start position until:
- a character which is not a digit is detected
- the maximum number of characters has been read
- the end of the character array is reached

-------- PRESET --------
ascii_array = array of type ARRAY [ ] char to be converted

---- RESULT ACKNOWLEDGMENT---
integer_number = converted number (only type INTEGER permitted)

-------- PRESET --------
index_ca = Position in the array as of which the number is to be read

-------- PRESET --------
length_char = Maximum number of characters to be read

---- RESULT ACKNOWLEDGMENT----
error_rue = Error number output

0	**No errors**
-1	**Start index outside the array limits**
-2	**End index (start index + length) lies outside the array limits**
-3	**-------**
-4	**Range transgression (value too high)**
-5	**Array length < 0**
-6	**Array length = 0**
-7	**Character string does not start with a number or sign**

```
@pa = JC (ph)

@p1 = JC (22) + @p3
```

applies as of Version TO01E

Conversion from world coordinates to joint coordinates.

```
pos_a = WC (@pal)

pos_1 = p3 + WC (@p4)
```

applies as of Version TO01E

Conversion from joint coordinates to world coordinates.

```
R =  50

MOVE LINEAR TO p1
```

applies as of Version TO02F

Global passing radius.
”R” acts in the case of LINEAR interpolation (acts kinematic−specifically)

```
MOVE LINEAR WITH R= 20
TO p1
```

applies as of Version TO02F

Statement−specific passing radius
”R” acts in the case of LINEAR interpolation (acts kinematic−specifically)

```
R_PTP= 1.4
;(or R_PTP=140%)

MOVE PTP VIA p2
```

applies as of Version TO02F

Global passing factor
”R_PTP” acts in the case of PTP interpolation (acts kinematic−specifically)

R_PTP is multiplied with the value of P213

```
R_PTP= 1.4
;(or R_PTP=140%)

MOVE PTP WITH
R_PTP =0.5 VIA p2
```

applies as of Version TO02F

Statement−specific passing factor.
”R_PTP” acts in the case of PTP interpolation (acts kinematic−specifically)

If R = 0 or R_PTP = 0 is programmed, Spatial passing movement is deactivated.

```
PROGRAM io_test

EXTERNAL: procname
INTEGER:  index,number,
          status
FILE: fina

CONST:
        textconst = 'I85.7'
        txbyconst= 'I13'

BEGIN

number=0
READ V24_1, index
IF CONDITION(V24_1)
       < 0 THEN
WRITE 'Read error'

status=
   CONDITION(fina)




status=
   CONDITION(procname)









status=
CONDITION('I0.0')

status=
CONDITION('I0')

status=
CONDITION(textconst)

status=
CONDITION(txbyconst)


PROGRAM_END
```

applies as of Version TO02F

Standard function CONDITION permits:
1a. The status of an interface to be determined.
V24_1 ... V24_4,PHG,TTY are supported

Function value:	
0	no errors
−1	Interface not found
−2	Interface blocked
−3	Timeout
−4	Parity error
−5	Overrun error
−6	Framing error
−7	Interface error
−8	Wrong string length
−9	Protocol error
−10	Illegal REAL value
−11	Point not defined

1b. The status of a file to be determined.
CONDITION (*filename*) checks, if the specified file exists.

Function value:	
0	specified file does not exist
1	specified file exists

applies as of Version TO04J

2. To get the status of processes.
The name of the process must be declared with EXTERNAL 'processname'.

Function value:	
−1	process does not exist
0	process is holding
1	process is running
2	process is waiting
3	process is ready
11	1 sub process of main process procname is active
12	2 subprocesses are active
13	3 sub processes are active
10+n	n sub proc. active (n=1..90)
>100	process error error code=func.−value −100

applies as of Version TO04J

3. To get the status of system signals (inputs and outputs).
There are two ways possible:
1. The address of the signal to be test can be used directly as parameter. Example: 'Ox.y', 'Ix.y '
2. The address of the signal is a CONST (of type text). Same for an input byte.

Function value:	
0/1	status of a signal "0" or "1"
0..255	status of a byte
−1	invalid signal group
−2	invalid bit address
−3	invalid character
−4	invalid signal address

applies as of Version TO04J

The BAPS command ASSIGN makes it possible to change the names of files or to change the assignment of standard channels (V24_1..V24_4, SER_1..SER_4, PHG) at the runtime of a BAPS program.

```
PROGRAM progname

FILE:filevar
TEXT:filename,one_line,
     channel_var


BEGIN
WRITE PHG,CLS
WRITE PHG,'file name:'
READ PHG, filename
ASSIGNfilevar,filename


READ_BEGIN filevar
READ filevar,one_line
WRITE one_line
CLOSE filevar




channel_var = 'V24_2.'
ASSIGN PHG, channel_var
WRITE PHG, one_line
ASSIGN PHG,'PHG.'




PROGRAM_END
```

Clears display of PHG

Read *filename* from the PHG
Assign the contents of *filename to filevar*. Following file I/O uses the file specified by *filevar*.

Open file for read
Read a line from the file and writes it on the PHG, then close the file.

note:
when ASSIGN is used with files, it is necessary to close the files before using **ASSIGN** with these files

assigns channel PHG to the content of channel_var (V24_2), writes *one_line* to channel V24_2 and assigns PHG back to channel PHG

note:
it is important to write the character '.' as limitation of channel names. If you omit the point '.', assignment is done to a filename (extension DAT), e.g. PHG.DAT

MOVEMENT INSTRUCTIONS

REF_PNT (3) **REF_PNT (1,2,3,...)**	applies as of Version TO01E	REFERENCE POINT TRAVEL Those axes that are to travel simultaneously to the reference point are specified in brackets. This should be the first command used in the INIT program. The default kinematic is moved.
REF_PNT ma_1 (3) **REF_PNT ma_2 (1,2,3,...)**	applies as of Version TO01E	REFERENCE POINT TRAVEL with kinematic entry
MOVE fetchpos **MOVE @fetchpos**	applies as of Version TO01E	MOVE ABSOLUTE This movement instruction traverses the robot in world or joint coordinates (default kinematic).
MOVE ma_2 fetchpos **MOVE ma_1 @fetchpos**	applies as of Version TO01E	MOVE ABSOLUTE with kinematic entry.
MOVE fetch + up **MOVE @fetch + @up**	applies as of Version TO01E	As for Move absolute, but the position is computed beforehand and, only then, the system moves.
MOVE ma_2 **fetch + up** **MOVE ma_1** **@fetch + @up**	applies as of Version TO01E	Ditto with kinematic entry.
MOVE_REL down **MOVE_REL @down**	applies as of Version TO01E	MOVE INCREMENTAL The robot can be advanced by a distance (in mm) or an angular distance (in degrees) from the current position (default kinematic).
MOVE_REL ma_2 down **MOVE_REL ma_2 @down**	applies as of Version TO01E	Ditto with kinematic entries
MOVE_REL **down + distance_z** **MOVE_REL** **@down + @distance_z**	applies as of Version TO01E	As for Move incremental, but the position is computed beforehand, and only then, the system moves. (default kinematic)

```
MOVE_REL ma_2
   down + distance_z
MOVE_REL ma_2
   @down + @distance_z
```

applies as of Version TO01E

Ditto with kinematic entries

```
MOVE TO pos1

MOVE_REL EXACT stre
```

applies as of Version TO01E

Move to points EXACT (with brief axis stop).

```
MOVE p1,p2,p3
```

applies as of Version TO01E

Move EXACT to several points (with brief axis stop).

```
MOVE VIA @palpos

MOVE_REL APPROX home
```

applies as of Version TO01E

Move to points WITHOUT EXACT HALT.

```
MOVE UNTIL force = 1

TO hole_5
```

applies as of Version TO01E

ABORT MOVEMENT
Movement occurs in direction hole_5 until hole_5 is reached or until condition (FORCE=1) is fulfilled.

```
MOVE UNTIL force = 1
   ERROR pause
   TO hole_5
```

applies as of Version TO01E

ABORT MOVEMENT
Movement occurs in direction hole_5 until the condition (FORCE=1) is fulfilled. If the destination position is reached and the condition is not fulfilled, the instruction after ERROR is executed

```
MOVE PTP @pos_door

MOVE_REL PTP palpos
```

applies as of Version TO01E

MOVE POINT−TO−POINT
in joint or world coordinates. The control moves synchronously PTP, i.e. all axes start and finish at the programmed point at the same instant.

```
MOVE LINEAR TO a_z

MOVE_REL LINEAR
              APPROX up
```

applies as of Version TO01E

LINEAR INTERPOLATION
The points are connected by an exact straight line (path), if the working area permits.

```
MOVE CIRCULAR
   (hi_put,end_put)
MOVE_REL CIRCULAR
   (@pos1,@pos2)
```

applies as of Version TO01E

CIRCULAR INTERPOLATION
Circular interpolation permits movement in circular arcs. One pair of points is always required so that the control can compute the circular arc size.

ACTUAL POSITION

```
posi = POS
   IF posi.X_C > 300
   THEN posi.X_C = 300
   MOVE TO posi
```

applies as of Version TO01E

ACTUAL POSITION
The current position of the robot can be read with the aid of POS, and it is thus available to the program for further evaluation (default kinematic)

```
posi = ma_2.POS
   IF posi.X_C > 300
   THEN posi.X_C = 300
   MOVE TO posi
```

applies as of Version TO01E

ACTUAL POSITION
Ditto with kinematic entries

```
@current = @POS
```

applies as of Version TO01E

The actual position can also be read in JC (joint coordinates).
(Default kinematic)

```
@current = ma_1.@POS
```

applies as of Version TO01E

The actual position can also be read in JC with kinematic entries.

MEASUREMENT POSITION

```
@p1 = @MPOS

@p1 = ma_1.@MPOS
  p1 = WC(ma_1.@MPOS)
```

applies as of Version TO02A

The component @MPOS contains the measurement position which can be determined with the high−speed inputs on the servo card with high accuracy. @MPOS acts only in the joint coordinate system but may be converted with the standard function WC.

SPEEDS

V_PTP = 80% **V_PTP = 0.8**	applies as of Version TO01E	POINT−TO−POINT SPEED The global PTP speed is written in one block on its own. It acts until it is overwritten by another speed (acts on the default kinematic).
ma_1.V_PTP = 80% **ma_2.V_PTP = 0.8**	applies as of Version TO01E	Ditto with kinematic entries. Acts only on the specified kinematic
V = 800	applies as of Version TO01E	PATH SPEED The global path speed acts on linear and circular movements of the default kinematic.
ma_1.V = 800	applies as of Version TO01E	Ditto with kinematic entries
A = 765	applies as of Version TO01E	PATH ACCELERATION The path acceleration acts only on linear and circular movements and not on PTP movements.
ma_1.A = 765	applies as of Version TO01E	Ditto with kinematic entries
MOVE PTP WITH **V_PTP = 30%** **TO home**	applies as of Version TO01E	LOCAL PTP SPEED The local PTP speed acts only in a MOVE (or Move_REL.) block. After this the global speed becomes active again.
MOVE ma_2 PTP WITH **V_PTP = 30%** **TO home**	applies as of Version TO01E	Ditto with kinematic entries
MOVE LINEAR WITH **V = 2000** **TO edge**	applies as of Version TO01E	LOCAL PATH SPEED The local path speed acts only in this block. It must be entered in mm/s.

MOVE ma_2 LINEAR WITH V = 2000 TO edge	applies as of Version TO01E	Ditto with kinematic entries
MOVE PTP WITH VFIX_PTP = 0.8 TO @pos_up	applies as of Version TO01E	Local point−to−point speed (block−by−block) without VFACTOR active (i.e. VFACTOR = 100%).
MOVE ma_1 PTP WITH VFIX_PTP = 0.8 TO @pos_up	applies as of Version TO01E	Ditto with kinematic entries.
MOVE LINEAR WITH VFIX = 300 VIA pal_pos_down	applies as of Version TO01E	Local path speed (block−by−block) without effect of the VFACTOR (i.e. VFACTOR = 100%).
MOVE ma_2 LINEAR WITH VFIX = 300 VIA pal_pos_down	applies as of Version TO01E	Ditto with kinematic entries.
AFIX = 300	applies as of Version TO01E	PATH ACCELERATION without AFACTOR active (= 100%). This statement is also possible only locally in the MOVE instruction for LINEAR and CIRCULAR (as VFIX above).
MOVE WITH T = 2.35 TO destpos	applies as of Version TO01E	TIME INPUT (in seconds) The next position is to be approached within a specific period. (Only local programming possible.)
MOVE ma_2 WITH T = 2.35 TO destpos	applies as of Version TO01E	Ditto with kinematic entries.

V / A / D FACTORS

VFACTOR = 100% **VFACTOR = 1**	applies as of Version TO01E	SPEED−FACTOR The PTP speed and the tool path feedrate can be influenced with the VFACTOR. The VFACTOR can also be modified via the PHG (acts on the default kinematic).
ma_1.VFACTOR = 100% **ma_2.VFACTOR = 1**	applies as of Version TO01E	Ditto with kinematic entries. Acts only on the specified kinematic.
AFACTOR = 100% **AFACTOR = 1** **DFACTOR = 100%** **DFACTOR = 1**	applies as of Version TO01E	ACCELERATION and DECELARATION These factors influence the tool path accelerations and the stored PTP axis accelerations. (Only above the slope point in the case of ramp slope.)
ma_1.AFACTOR = 100% **ma_2.DFACTOR = 1**	applies as of Version TO01E	Ditto with kinematic entries. Acts only on the specified kinematic.

SLOPE MODES

MOVE WITH **TFIX = 2.35** **TO destpos**	applies as of Version TO01E	Time input without VFACTOR active (=100%).
MOVE ma_2 WITH **TFIX = 2.35** **TO destpos**	applies as of Version TO01E	Ditto with kinematic entries
PROGR_SLOPE	applies as of Version TO01E	Global switchover to the program slope. No downslope (setpoint = 0 V) is executed between several 'MOVE VIA' blocks. A 'MOVE TO' statement executes a downslope (acts on the default kinematic).
;; KINEMATICS = ma_2 **PROGR_SLOPE**	applies as of Version TO01E	Ditto. If a kinematic other than the default kinematic is to be switched over, this must be set beforehand as the default kinematic.
BLOCK_SLOPE	applies as of Version TO01E	Global switchover to the block slope. A downslope (setpoint = 0 V) is also executed between 'MOVE VIA' blocks.
;; KINEMATICS = ma_2 **BLOCK_SLOPE**	applies as of Version TO01E	Ditto with kinematic entry (see above). See rho3 machine parameters

BELT SYNCHRONIZATION

The synchronization statements ensure that the controlled machine assumes the correct position and orientation with respect to the belt. The belt may move forwards and backwards, change its speed or stop as required.
The belt must be a “straight line”. This line may be arbitrarily positioned in space.

```
ma_1.BELT:
      501 = belt_kin1
```

applies as of Version TO01E

DECLARATION OF THE BELT VARIABLES. Several belts can be declared for a kinematic. The belt names must be different.
The same belt can be used for several kinematics.

```
SYNC
      belt_kin1 >= 10.0
SYNC
      BELT, light = 1
```

applies as of Version TO01E

Belt counter is reset. Reseting may be dependent upon the current value of the belt variables themselves or upon a condition (see SPC_FCT 28).

```
SYNCHRON
      ma_1 belt_kin1
```

applies as of Version TO01E

Instruction SYNCHRON activates belt synchronization. From this point on, the programmed movements are synchronized with the belt as regards position and orientation. The kinematic entry is optional.

```
SYNCHRON_END
      ma_1 belt_kin1
```

applies as of Version TO01E

Instruction SYNCHRON_END deactivates belt synchronization. The kinematic entry is optional; if it is omitted, the default kinematic is used.

WORKING AREA LIMITS

```
LIMIT_MIN =
    (x_value,...,u_value)

LIMIT_MAX =
    (x_value,....,u_value)

LIMIT_OFF
```

applies as of Version TO01E

Minimum and maximum values for working area limits. These values are always world coordinate positions.

Cancellation of the working area limits.

```
ma_1.LIMIT_MIN =
    (x_value,...,u_value)

ma_1.LIMIT_MAX =
    (x_value,....,u_value)

LIMIT_OFF  ma_1
```

applies as of Version TO01E

Working area limits with kinematic entries.

WRITE / READ INTERFACES

The WRITE/READ operations in BAPS2 are performed on so−called logical devices to which specific transmission protocols are assigned by default and by machine parameters. Assignment to a physical interface is also performed via machine parameters. This permits flexible use of the interfaces on the control.

PHG	applies as of Version TO01E	SERIAL INTERFACE. Permanently assigned to the hand−held programming unit PHG3.
V24_1 **V24_2** **V24_3** **V24_4**	applies as of Version TO01E	SERIAL INTERFACES. See Default via PHG/machine parameters for the logical and physical assignment.

Various communication protocols are available for communication. These can be selected via machine parameter (default) or via mode 9.1 using the PHG.

P.− No.	Protocol structure	Read Echo
1 a 1 b	<DATA> followed by <CR><LF> <DATA> followed by<CR> oder <LF>	yes
2	<DATA>	yes
3	<SOH><STX><DATA><ETX> followed by <SOH><STX><CR><LF><ETX>	no
4	<SOH><STX><DATA><ETX>	no
5	<DATA>	no
6	PHG − Protocol	yes
7	rho1 / rho2 compatible with P.No. 3	no
8	Data protection layer ot the Siemens−Protocol 3964/R	no

1a = Data−Input
1b = Data−Output

```
WRITE V24_1,POS

WRITE PHG, 'Number', i
```

applies as of Version TO01E

It is possible to write variables and texts from the BAPS program to the interface V24_1, V24_2, V24_3, V24_4 and PHG.

```
ARRAY[1..70] CHAR:
            F1_CHAR

WRITE F1_CHAR

ARRAY[1..4]
ARRAY[1..50] CHAR:
            F2_CHAR

WRITE F2_CHAR [1]
```

applies as of Version TO02C

It is possible to write variables, texts and one−dimensional arrays from the BAPS program to the interface V24_1, V24_2, V24_3, V24_4 and PHG.
Note:
If a BAPS program has to communicate via an interface which is used by ROPS3, the ROPS3−rho3 communication has to be deactivated before.

```
WRITE PHG, '<−[J', ...
```

applies as of Version TO01E

Clear PHG display
<− corresponds to the control character of ESC
(See chapter STANDARD CONSTANTS "CLS")

```
WRITE PHG, '<−[K', ...
```

applies as of Version TO01E

Clear to end of line on the PHG display
(<− corresponds to the control character of ESC)

```
WRITE PHG,
   '<−[line;columnH', ...
```

applies as of Version TO01E

Position the cursor to line and column on the PHG display. (The H is required as the end character for the column)
(<− corresponds to control character of ESC)

```
READ M

READ V24_2, M
```

applies as of Version TO01E

It is possible to read variables and texts from the interfaces V24_1, V24_2, V24_3, V24_4 and PHG, starting from the BAPS program.

FILES – I/O (READ)

```
FILE: fina
```

applies as of Version TO01E

Type declaration for ASCII files which are to permit read and/or write access within a BAPS program.
(E.g. in this case, access to FINA.DAT)

```
FILE: fina

READ_BEGIN
   fina, 5
```

applies as of Version TO01E

Positioning the READ pointer to the start of a defined line of the given file (in this case, the fifth line in file FINA.DAT). The pointer is positioned to line 1 if the line entry is omitted.

```
FILE: fina
INTEGER:   otto

READ fina, otto
```

applies as of Version TO01E

The first value is read from the file (in this case: FINA.DAT) as of the positioned line, and is stored into the variable (e.g. OTTO). The invisible READ pointer is advanced by one position.

```
FILE: fina

IF
END_OF_FILE (fina)
   THEN JUMP finished

finished: ...
```

applies as of Version TO01E

This function permits interrogation of whether the file end has been reached
(e.g. FINA.DAT).

```
BNR_FILE:  fina
```

applies as of Version TO05G

Type declaration of BINARY–files which are to permit read and/or write access within a BAPS program.
(E.g. in this case, access to FINA.BIN)

FILES – I/O (WRITE)

```
FILE: danadat
INTEGER:   number

number = 1

WRITE_BEGIN
       danadat, 1

WRITE_BEGIN
       danadat, number
```

applies as of Version TO01E

The write pointer is positioned to the start of a specific line in the specified file (e.g. danadat.DAT).

note:
old data, existing behind the declared line number are deleted

```
FILE: danadat
REAL:    rslt
TEXT:    display

display =  'the result:'

WRITE danadat,
       display, rslt
```

applies as of Version TO01E

The contents of the variables are written to the specified file. The write position is the position to which the write pointer is pointing.

note:
data behind the write pointer are deleted

```
FILE: danadat


WRITE_END
       danadat
```

applies as of Version TO01E

The write pointer is set to the end of the file. The file is extended (appended) at the end with the next WRITE instruction.

```
FILE: danadat

CLOSE danadat
```

applies as of Version TO01E

A DAT file must always be closed after writing before it can be read. (The DAT file is closed automatically at the end of the program).

PARALLEL PROCESSES

```
PARALLEL PROCESSES (EXTERNAL)
..
EXTERNAL: gripper

.
START gripper
.
START gripper PRIO=100
.
.
STOP gripper
.
.
```

applies as of Version TO01E

(no synchronized end with parallel processes)

External program with name must be defined in the declaration part.

The main program (HP) and external program (in this case: GRIPPER) are run in parallel.

A priority scale from 100 (highest) to 150 (lowest) can be assigned.

Parallel program Gripper is stopped and only the main program continues to run.

```
PARALLEL PROCESSES (INTERNAL)
.
..
PARALLEL
    MOVE pal_pos_1
    I = 5
    .
    .   .
ALSO
    MOVE ma_2 TO pal_corner_I
    .
    .
    ALSO
        .
        .       .
PARALLEL_END
.
.
MOVE TO home
```

(synchronized end in the case of parallel processes)

1st process

2nd process

3rd process
run simultaneously.

When all processes are finished, the system continues in the program after PARALLEL_END.

```
SEMAPHORE:sema_var

EXCLUSIVE
    sema_var
    .
EXCLUSIVE_END
```

applies as of Version TO01E

If it is intended to access exclusively to common resources (e.g. printers), this can be achieved with the Exclusive statement. The semaphore variable must have been declared beforehand in the declaration part and acts globally in the operating system, i.e. an interlock function can be performed by all programs.
If another parallel−running process is using the semaphore, the second process remains in ”wait” state until the semaphore is released again.
EXCLUSIVE − uses the semaphore
EXCLUSIVE_END − releases the semaphore
If the parallel running process is not using the semaphore, the interlock has no effect.

SPECIAL FUNCTIONS

SPC_FCT:	applies as of Version TO01E	Functions to which no BAPS keywords are assigned are declared as SPECIAL FUNCTIONS. All special functions are listed in order of number below.
1 = eal **(VALUE INTEGER: eal_no** **VALUE INTEGER: kin_no** **VALUE INTEGER: coordno** **VALUE REAL: outp_pos** **VALUE REAL: para_outp** **VALUE INTEGER: rate time)**	applies as of Version TO01E	Exact−position switching of digital signals on the path with rate time. Up to 8 channels can be used for the function.
2 = ppa **(VALUE INTEGER: ppa_no** **VALUE INTEGER: kin_no** **VALUE INTEGER: coord_no** **VALUE REAL: outp_pos** **VALUE REAL: param** **VALUE INTEGER: rate)**	applies as of Version TO01E	Exact−position output of process parameters output with rate time. SPC2 includes 8 functions for output of 8 bit−wide outputs (value range 0..255) on the interface to the control or any decimal value via a free analog value output on the servo card.
3 = mach_pos **(VALUE INTEGER : kin_no** **kin_name.JC_POINT :** **@p_name)** **3 = mach_pos** **(VALUE INTEGER : kin_no** **JC_POINT : @p_name)**	applies as of Version TO03D	Special function 3 sets the internal machine position to the values specified in the point variables @p_name. If no kinematic is entered when declaring the special function, the point relates to the default kinematic. The special function must be declared separately for each kinematic to which it is intended to assign machine positions.

IMPORTANT :

The kinematic selected with the variable **kin_no** and kinematic defined in the declaration must be the same.
The operating system of the rho3 control does **not** check for correct assignment.

CAUTION ! (Special function 3) :

Note that if machine positions are activated due to incorrect programming, the controller will no longer 'know' where the machine actually is.

```
4= command
   (VALUE INTEGER: com
                TEXT: src,dest
             INTEGER: status)
```

This special function permits to use rho3 system functions in a BAPS2 program.

command = name of special function for use in BAPS

com = selects the system function
- 1= COMPILE (QLL−file)
- 2= COPY (any file)
- 3= DELETE (any file)
- 4= START (user process)
- 5= STOP (active user process)

src,dest = parameters for selected system function

status + result of the function as an integer value:
- 0 = no error
- −1 = invalid command
- −2 = error in source file name if COPY or error in file name if DELETE or error in file name if COMPILE was selected
- −3 = file extension is not QLL if COMPILE
- −4 = error in QLL file name if COMPILE
- −5 = file not closed if DELETE or destination file is not closed if COPY
- −6 = erro if COPY, e.g. out of memory
- −7 = compiler is already active if COMPILE
- >0= number of error and warnings if COMPILE or error number if START:
  - 1051: user process already exist
  - 1057: file to much open
  - 1073: process is an ext. sub program
  - 1079: sub process tries to stop its own main process
  - 1085: invalid PKT file
  - 1092: selected process not available
  - 1112: sequent. of IRD− and PKT− file is not possible because there's not enough memory available

```
17 = mirroring
   (VALUE BINARY:  X_INV
                   Y_INV
                   Z_INV)
mirror, (1,1,0)

MOVE @(50,120,−25.21)
```

applies as of Version TO01E

Axes 1 and 2 are mirrored in this example.

```
21 = belt_mode
   (VALUE INTEGER:belt_no
   VALUE INTEGER:mode_belt)
```

applies as of Version TO03D

This special function permits the required mode of belt synchronization to be selected. The following synchronization modes are currently implemented:

mode_belt = **1** Belt synchronization with belt−parallel traversing of the kinematic

mode_belt = **2** Belt synchronization without belt−parallel traversing of the kinematic.

mode_belt = **3** cam plate interpolation

23 = time_date
(INTEGER: hours,
minutes,
day,
month,
year)

applies as of Version TO01E

Access to the time and date of the system clock in the rho3.

24 = sys_time
(INTEGER:start_time)

applies as of Version TO01E

Access to an internal system counter (real−time counter).

27 = wc_mainarea
(INTEGER: kin_no)

applies as of Version TO01E

Coordinate transformation may lead to excess rotation of the world orientation angles in WC. Special function 27 serves to eliminate this excess rotation.

28 = set_belt
(VALUE INTEGER: belt_no
VALUE REAL: reset_value)

applies as of Version TO03D

Calling special function 28 assigns the value of variable reset_value to the internal belt reset value for the corresponding belt
The internal belt counter is set to the reset value with signal RESET BELT COUNTER.

29 = pnt_store_on
(VALUE INTEGER: kin_no
VALUE INTEGER: ms_no
VALUE REAL: distance)

Calling special function 29 activates storing of internal "commanded positions".
Stores positions as WC and JC points.

kin_no	=	number of selected kinematic
ms_no	=	number of measuring system (axis, belt or analogous input) which is related to the pa−rameter 'distance'
distance	=	stores actual "commanded position" when distance is reached

note:
Storing of position after every 'distance' will be done on default (option byte 36 = "0", but it's also possible to store positions at every transformation clock by setting the option byte 36 = "1".

30 = pnt_store_off
(VALUE INTEGER: kin_no
INTEGER: no_points)

Calling special function 30 terminates storing of "commanded positions".
The stored values are saved until the next call of special function 29 with the same kinematic.

kin_no	=	number of selected kinematic
no_points	=	returns the quantity of stored points since call of special function 29.

31 = pnt_st_read
(VALUE INTEGER: kin_no
VALUE INTEGER: index
JC−POINT: @position
POINT: position)

Calling special function 31 makes it possible to read the values stored by special function 29 in a ring buffer.

kin_no = number of selected kinematic
index = ring buffer index of selected value
@position = stored position in JC
position = stored position in WC

note:
To get valid positions it's necessary to call up SPC_FCT 29 and 30 before calling SPC_FCT 31.
The size of the ring buffer is fixed to 50 points independent of the number of axes.

41 = poti_modify
(VALUE INTEGER: kin_no
VALUE REAL: poti_value)

applies as of Version TO05G

Calling special function 41 "Asynchronous speed preselection" permits the modification of the Vfactor of a kinematics during movement.

kin_no = number of selected kinematics
poti_value = asynchronous Vfactor

42 = move_dist
(VALUE INTEGER: kin_no
VALUE REAL: m_distance)

applies as of Version TO05G

Opposite to the normal "MOVE UNTIL"−statement the movement by this special function is not directly aborted if the condition is true (probe input).
The distance in parameter *m_distance* is moved before the movement is ended.

kin_no = number of selected kinematics
m_distance = distance to move after active probe input

note:
This modified "MOVE UNTIL"−statement acts only with probe inputs.
The base of this inputs is "700".

43 = probe_on
(VALUE INTEGER: kin_no
VALUE INTEGER: flank
INPUT: channel_no)

applies as of Version TO05G

Special function 43 permits to store the actual position by a minimum reaction time of 0,01 ms. The measured value is stored in the standard varaiable "MPOS". The parameter *flank* is used to select the trigger condition (starting the measurement).
The *channel_no* indentifies, if a measurement has occured.
kin_no = number of selected kinematics
flank = positive flank = 0
negative flank = 1
channel_no = measurement occurs = 1
no measurement = 0
note:
The probe input acts only with incremental measuring systems.
The base of the probe inputs is "600".

44 = probe_off
(VALUE INTEGER: kin_no
INPUT: channel_no)

applies as of Version TO05G

Calling special function 44 disables an active special function 43.

kin_no = number of selected kinematics
channel_no = input number of probe input

45 = move_file
(VALUE INTEGER: kin_no
BNR_FILE: curve_x
VALUE INTEGER: base
JC_POINT:@mod_flag
VALUE ARRAY[1..6]
INTEGER: reserve)

applies as of Version TO05G

Calling SPC_FCT 45 "MOVE_FILE" a curve chart stored in a binary file is moved by reading the actual values in this file. The output takes place in the interpolation raster or positioning control raster of the servo board clock. The issued values are position values.
kin_no = number of selected kinematics
curve_x = name of the BNR_FILE
base = selects the output of the position values.
1 = interpolation raster
2 = positioning control raster
mod_flag = The modulo flag is used by endless axes. Other axes types must be initialized with "0.0" in the BAPS−program.
The following REAL−values are possible:
0.0 = no modulo calculation
1.0 = modulo calculation at the end of the block. The last axes value in the BNR_FILE is the modulo value.
2.0 = modulo calculation at the end of the block. The value stored in machine parameter P311 is used as modulo value.
reserve = reserved for future expansions of the operating system
restrictions:
This special function doesn't work with the rho 3.2 control.

46 = block_prep
(VALUE INTEGER: kin_no
VALUE INTEGER: number)

applies as of Version TO05G

Calling special function 46 ”reduced block preparation with MOVE−statements” permits to reduce the internal block preparation.
The use of this special function is recommended in connection with ”MOVE_FILE” and ”Asynchronous Inputs”.
Especially with applications of asynchronous inputs it could be necessary to reduce the block preparation to ensure faster recognition of the inputs.

kin_no = number of selected kinematics
number = This value is an upper limit of prepared blocks of a kinematics. The range of this parameter is between 1..11.
Values < 1 are set internal to 1, values > 11 are set to 11.

TOOL – DAT

Coordinate transformation for determining the TOOL center point (TCP) is adjusted in two parts in order to permit different grippers/tools to be used during a processing run.

a) Transformation as far as the flange.
The robot–specific data are defined via the machine parameters.

b) Transformation as of the flange
The individual gripper parameters are stored in the file. Three translations (G_X, G_Y, G_Z) and three rotations (G_D1, G_D2, G_D3), each referring to the flange, are used for general definition.

TOOL.DAT

gripper_l	**= 10**	**2.5**	**5**	**1**	**2**	**3**
gripper_r	**= 0**	**50.0**	**1.3**	**5.4**	**0**	**0**
gripper_1	**=–20**	**0**	**120**	**5**	**0**	**6**
off	**= 0**	**0**	**0**	**0**	**0**	**0**

The order of the individual coordinates (first G_X, then G_Y, ... and last, G_D3) must always be observed. Zeroes must be set explicitly for values omitted. The gripper name must start in the first column and may have a maximum length of 12 characters.

```
TOOL
      ma_1 gripper_l
```

applies as of Version TO01E

A gripper is selected in the BAPS program via command TOOL.
e.g.: The gripper coordinate system ”gripper_1” from the file is offset.

```
TOOL
      ma_1 off
```

applies as of Version TO01E

There is no TOOL_END. However, this can obtained by a gripper coordinate system padded with zeroes.
(see example).

FIXED FILES

EXPROG.DAT

```
00 = init
01 = pal1
.
FF = basicpos
```

EXTERNAL PROGRAM SELECTION.
The interface input numbers are assigned to the program names (IRD) in file EXPROG.DAT.

MZA.DAT

```
01 =  'slide forward'
.
99 = 'end mza'
```

MACHINE STATUS DISPLAY.
The interface input numbers are assigned to the texts in file MZA.DAT. These selected texts can be displayed via the PHG in mode 7, 12, 1/2.

TEXTE.DAT

```
00 =  'text1'
.
FF = 'text256'
```

SELECTION OF TEXTS VIA THE INTERFACE.
The interface input numbers are assigned to the texts in file TEXTE.DAT.

TOOL.DAT

```
gripper_l    = 10    2.5     5     1     2  3
gripper_r    = 0     50.0    1.3   5.4   0  0
gripper_1    =−20    0       120   5     0  6
off          = 0     0       0     0     0  0
```

TOOL COORDINATE description
The order of the individual coordinates must always be observed (zeros must be set explicitly for values omitted). The gripper name must start in the first column and may have a maximum length of 12 characters. (See Tool.)

BAPSPIC – SYNTAX

1. Declaration part

All variables used in a BAPSPIC – program may have a maximum length of 12 characters. The first character must be a letter. As special character only the underline "_" is permitted. Upper – and lower letters are equivalent.
Compiler instructuions like **CONTROL** and **VERSION** must be declared before **PROGRAM**.
Lines containing compiler instructions should have no other characters following the statement.

```
;; CONTROL = PIC250
```

COMPILER INSTRUCTION "PIC250".
Indicates that the current program is a PIC – program (file extension "QLS").

```
;;VERSION = 200
```

COMPILER INSTRUCTION "VERSION".
Indicates which complier version is used compiling this PIC – program (from ROPS3 – Version W2C the new BAPSPIC – compiler version 2.00 is available).
The default setting is compiler version 1.00

```
PROGRAM pic_io
```

PROGRAM NAME.
The program name may have a maximum length of 8 characters. The name should agree with the file name, otherwise a warning is reported during compilation of this file.

```
;;INCLUDE pic_sym
```

COMPILER INSTRUCTION "INCLUDE".
Like in QLL – programs there is the possibility to include other files in the program.
This allows to create the symbol file in a separate file which can be included in the PIC – program by the *INCLUDE* statement.
From compiler version 2.00 the file extension is of your own choice. If there is no extension indicated, the compiler uses "QLS".

```
PROGRAM pic_io

.
INPUT:
0 = permiss_phg,
.
.
767 = user_24_di


OUTPUT:
16 = emerg_n_rci,
.
.
743 = user_24_do
```

Declaration of IN – and OUTPUTS.
All inputs and outputs used in the PIC – program must be declared.
The symbol names are separated by a comma.
After the last symbol name in a group, the comma is omitted (see example).

```
PROGRAM pic_io
.
.
BINARY:
0 = c0_decrem_m,
.
.
191 = c31_reset_m


TEMP BINARY:
marker_1, marker_2, marker_3
```

Declaration of REMANENT and NON – REMANENT MARKER.
All marker used in a PIC – program must be declared.
Remanent marker are declared as ”BINARY”, temporary marker as ”TEMP BINARY”.
The symbol names are separated by a comma.
After the last symbol name in a group, the comma is omitted (see example).

2. Program part

```
PROGRAM pic_io

BEGIN
      ;instructions
   .
PROGRAM_END
```

INSTRUCTION PART.
From compiler version 2.00 the syntactical construction of a BAPSPIC–program is adapted to the BAPS2–Syntax. This means that the instruction part is now separated from the declaration part by the keyword "BEGIN".
The end of the program is indicated by the keyword "PROGRAM_END".

```
feede_aa_rci = feeden_aa_di

cyclrun_do =  proc_act_rco
              OR
              permpr_a_rco

user_10_do =  feedhd_n_di
              AND
              contr_aa_di
```

ASSIGNMENTS.
With the "="–character the value of the variable or expression right of the sign of equality is assigned to the variable on the left.
The keyword "OR" permits a logic OR–combination of several variables (*var1* OR *var2* OR *var3*).
The keyword 'AND" permits a logic AND–combination of several variables (*var1* AND *var2* AND *var3*).

```
IFauto_mn_di
THEN
auto_mn_rci = 0

IF NOTuser_1_di
THEN
user_1_do = 1
```

IF–THEN INSTRUCTION.
The IF–THEN instruction permits the build of conditional instructions. This means the instruction is executed if the condition is true.

The keyword "NOT" permits the examination of a variable or expression of logic 0.

```
IF NOT auto_mn_dI
THEN
auto_mn_rci = 0
ELSE
user_1_do = 1
```

IF–THEN–ELSE INSTRUCTION.
This instruction permits the execution of several instructions depending on the input condition.

```
IF NOT auto_mn_di
THEN BEGIN
          auto_mn_rci = 0
          user_2_do = 1
       END
ELSE BEGIN
          user_2_do = 1
          user_3_do = 0
       END
```

COMPOUND INSTRUCTION.
The compound instruction is used to clamp several instructions.
Thereby it is possible to execute several instructions where normally only one instruction is permitted.

```
feede_aa_rci =   feeeden_aa_di
                 AND
                 (marker_1 OR marker_2)
                 OR
                 marker_3
```

BUILDING COMPONENT EXPRESSIONS.
Using the parentheses "(" and ")" permits the construction of one expression of several variables.

```
IF NOT auto_mn_di
THEN
JUMP automatic
.
.
automatic:
IF marker_3 THEN user_11_do = 1
```

JUMP INSTRUCTION.
The jump instruction permits jumps to several positions in the PIC – program.
The instruction consists of a jump instruction (JUMP) and a destination (*mark:*).

INDEX

Symbols

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Bosch-Automationstechnik

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Industriehydraulik
Postfach 30 02 40
D-70442 Stuttgart
Telefax (07 11) 8 11-18 57

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Fahrzeughydraulik
Postfach 30 02 40
D-70442 Stuttgart
Telefax (07 11) 8 11-17 98

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Pneumatik
Postfach 30 02 40
D-70442 Stuttgart
Telefax (07 11) 8 11-89 17

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Montagetechnik
Postfach 30 02 07
D-70442 Stuttgart
Telefax (07 11) 8 11-77 12

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Antriebs- und Steuerungstechnik
Postfach 11 62
D-64701 Erbach
Telefax (0 60 62) 78-4 28

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Schraub- und Einpreßsysteme
Postfach 11 61
D-71534 Murrhardt
Telefax (0 71 92) 22-1 81

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Entgrattechnik
Postfach 30 02 07
D-70442 Stuttgart
Telefax (07 11) 8 11-34 75

Technische Änderungen vorbehalten

Ihr Ansprechpartner

BOSCH

Robert Bosch GmbH
Geschäftsbereich
Automationstechnik
Antriebs- und Steuerungstechnik
Postfach 11 62
D-64701 Erbach
Telefax (0 60 62) 78-4 28

1070 073 035-105 (94.12) GB · HB RC · AT/VSP · Printed in Germany