SOFTWARE

KR C2

Seminar workbook
of ....

Advanced Robot Programming

Release 5.x
Issued: March 2005

Contents

1. Structured programming 5
   1.1. Programming methodology - Conceptualization 5
   1.2. Program flowchart 13
   1.3. Structogram 23
   1.4. Programming methodology - Realization 31
2. General inforamation about Expert level 45
   2.1. Navigator 45
   2.2. Fold – Concept 49
3. Variables and declarations 53
   3.1. Simple data types 53
   3.2. Monitoring of variables 59
   3.3. Arrays 63
   3.4. Structures 69
   3.5. Enumeration data types 73
4. Subprograms and functions 77
   4.1. Definition of subprograms 77
   4.2. Transfer of parameter 81
   4.3. Function 85
5. Data lists 89
6. Data manipulation 93

7. Motion programming 99
 7.1. Coordinate systems 99
 7.2. Axis motion – PTP 103
 7.3. Continuous path motion – LIN und CIRC 109
 7.4. Orientation control 113
 7.5. Advance run 119
 7.6. Approximate positioning 123
 7.7. Status and turn 129
8. System variables 133
9. Program execution control 137
 9.1. Jump instruction and branches 137
 9.2. Loops 141
 9.3. Wait and Halt function 147
10. Automatik External function 151
 10.1. Definition and sequence 151
 10.2. Configuration inputs 159
 10.3. Configuration outputs 169
11. Submit – Interpreter 175
12. Exercises 179

1. Structured programming

1.1. Programming methodology - Conceptualization

Programming methodology – Objectives

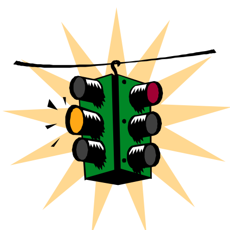
Objectives for consistent programming methodology:

- Structured programming makes it easier to solve complex problems by imposing a rigidly structured, hierarchical program layout.
- Structured programming is a method of creating programs in which the underlying process is comprehensible and transparent.
- The aim of structured programming is not to restrict the programmer, but to increase efficiency during program maintenance (modification and expansion).

Demands on programs

Programs must be:

- Efficient
- Free from errors
- Easy to understand
- Maintenance-friendly
- Clearly structured
- Economical



- Each of these features describes a specific aspect of the utility value of the program.

Design and planning

Necessity:

- ➢ Before planning how to proceed, it is necessary first to define **what** is to be achieved.

- ➢ This does not increase the overall amount of programming work, since the extra work at the planning stage is more than offset by the amount of work saved in the test phase.

Conceptualization and planning - Requirements specification

User expectations as basis for program planning:

➢ Tender specification or requirements specification

- What does the program do?
- What inputs does the program require?
- What outputs does the program supply?
- How must the program be operated?
- How does the program react to a fault/malfunction?

Conceptualization and planning - Requirements specification

Definition:

Requirements specification

A requirements specification is the organizational and technical specification for program creation.

Conceptualization and planning - Requirements specification

Advantages of a requirements specification:

- Description of the application
- Definitions of inputs/outputs/parameters and program sequences
- Transparent breakdown
- Project familiarization significantly easier

Design and planning

An exact **functional description** must exist first before programming can begin. It must be clear and simple, but nonetheless exhaustive. The functional description must correspond exactly to the task requirements. It forms the basis for the planning.

This is because:

- Uncertainties increase the probability of user and/or operator errors.
- Wherever possible, the program must do what the user expects it to do – what seems natural and understandable.

Program planning methodology

1. Break down task into modules:

- The overall task is broken down into subtasks.
- Individual components can safely be developed separately.

- Components that do the same job can be exchanged simply without affecting other functions.

Program planning methodology

2. Component independence:

- Starting from the user expectations, the problem is gradually refined until the resulting components are manageable enough to be converted into KRL.

- The drafts made at successive stages of development become steadily more detailed.

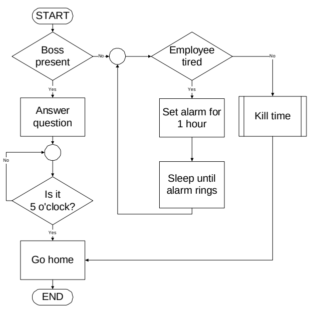
 I 20.11.2003 I College I SG I 9

Program planning representation

Graphic representation of the procedure:

- Using a program flowchart (DIN 66001)

- Using a structogram (DIN 66261)



 I 20.11.2003 I College I SG I 10

Program planning representation - advantages

This has the following advantages:

- The graphic representation of a program algorithm is significantly more legible than the same program algorithm in a sequential programming language as the structure is much easier to recognize.

- Structure errors (e.g. if, while, for) will be much easier to avoid when translating the program into program code as a direct translation into program code is possible in most cases.

- The program can be documented clearly.

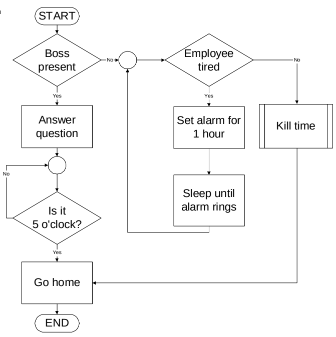
1.2. Program flowchart

Program flowchart

Graphic representation of the procedure:

➢ Program flowcharts (DIN 66001)

Program flowcharts are one way of formulating abstract descriptions of programs and processes. The sequence is represented using standardized symbols that are not dependent on the program code used.



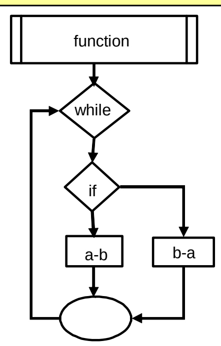
KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Program flowchart – comparison with programming language

Example of a program flowchart:

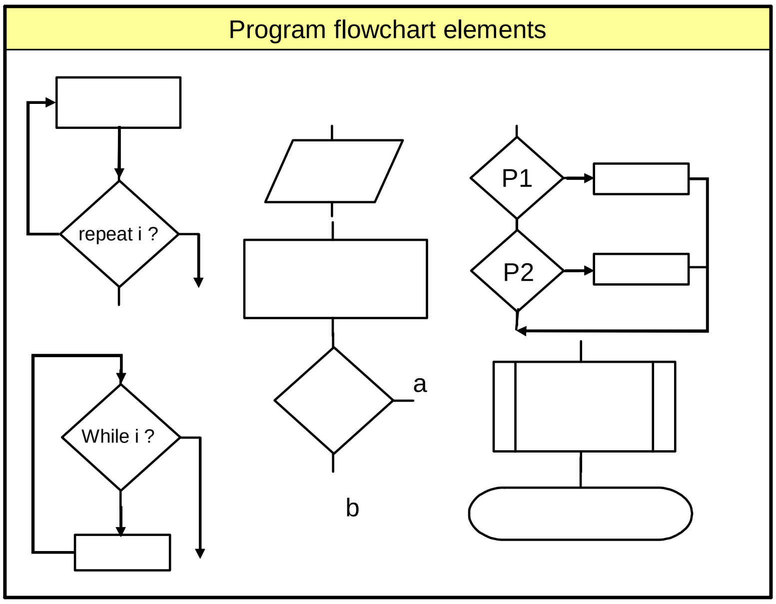
An algorithm is required for calculating the highest common factor of two numbers a and b. This is embedded in a function ‘hcf’ with two transfer parameters (a and b).

Program flowchart	KRL	Pascal

		function hcf (a,b:integer);
		
		while a<>b do
		
		If a>b
		
		then a:=a-b else b:=b-a;
		
		hcf:=a;

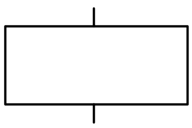
KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Program flowchart - overview of symbols



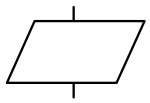
 I 20.11.2003 I College I SG I 3

Program flowchart - symbols

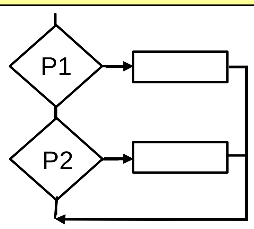
Symbols	**Statement**	**Description**
	General statements in the program code	The statements are executed in the sequence Statement 1, Statement 2, ..., Statement n.

 I 20.11.2003 I College I SG I 4

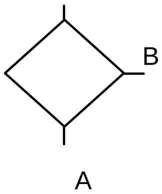
Program flowchart - symbols

Symbols	Statement	Description
	Input/output statement	This statement is used for input/output and is treated as a normal statement.

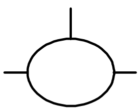
Program flowchart - symbols

Symbols	Statement	Description
	IF statement (without ELSE)	If the logic condition is met (true), statements A are executed, otherwise program execution is resumed at the next statement.

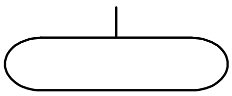
Program flowchart - symbols

Symbols	Statement	Description
	IF statement (with ELSE) or branch	If the logic condition is met (true), statements A are executed, otherwise statements B are executed.

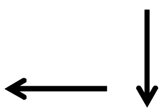
Program flowchart - symbols

Symbols	Statement	Description
	Branch	A branch/transfer without conditions

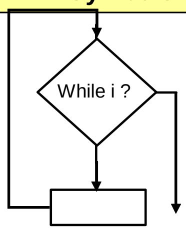
Program flowchart - symbols

Symbols	Statement	Description
	Terminal	Start or end of a process or program

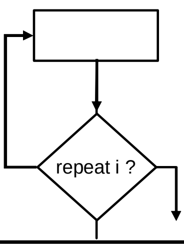
Program flowchart - symbols

Symbols	Statement	Description
	Flow lines	Link statements and operations

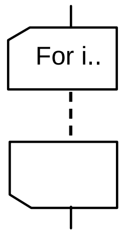
Program flowchart - symbols

Symbols	Statement	Description
	WHILE loop	The statements are executed while the condition is fulfilled. As soon as the condition is no longer met when execution of the loop is started again, program execution is resumed at the first statement after the loop.

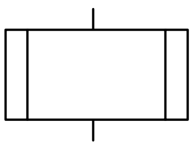
Program flowchart - symbols

Symbols	Statement	Description
	REPEAT loop	The statements are executed until the condition is fulfilled. As soon as the condition is met, program execution is resumed at the first statement after the loop. The statements are always executed at least once.

Program flowchart - symbols

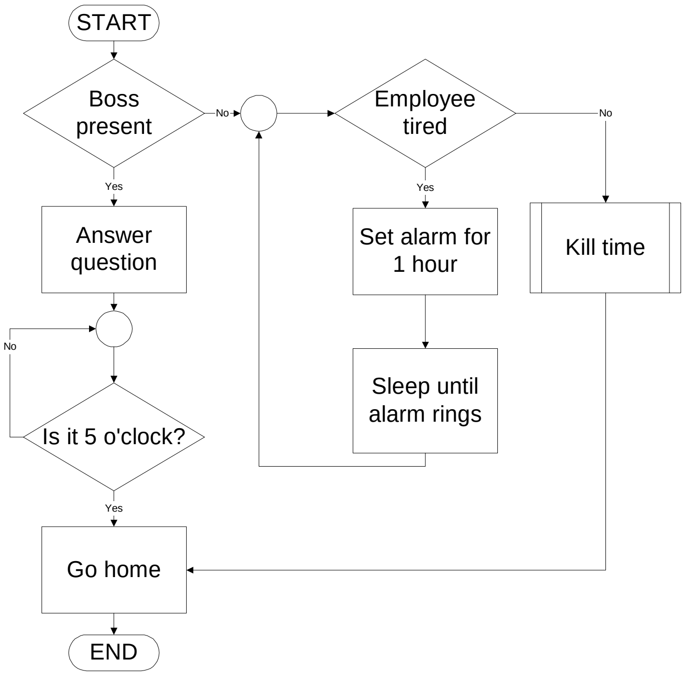
Symbols	Statement	Description
	FOR loop	The statements are executed a known number of times. The run variable i runs from the initial value (“from”) to the final value (“to”) with the specified increment (“step”). As soon as the run variable is greater than or equal to “to”, program execution is resumed at the next statement.

Program flowchart - symbols

Symbols	Statement	Description
	Subprogram calls	A certain sequence of statements is segregated from a certain position in the program and implemented as a separate subprogram that can then be called from the program.

Program flowchart example - main program

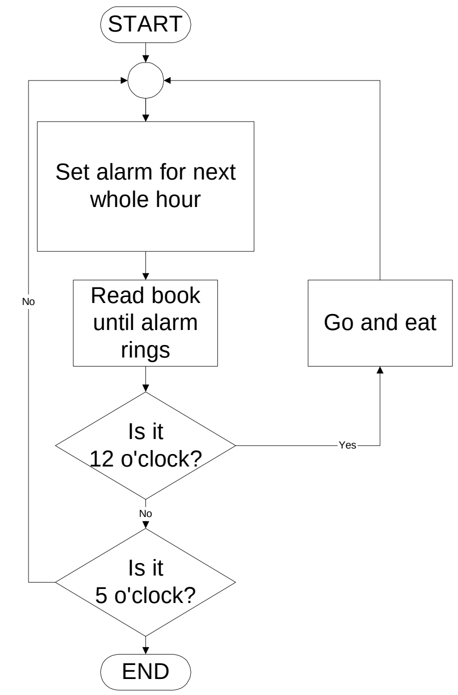
Program flowchart example: day in the life of an employee



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Program flowchart example - subprogram

Program flowchart example: day in the life of an employee



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

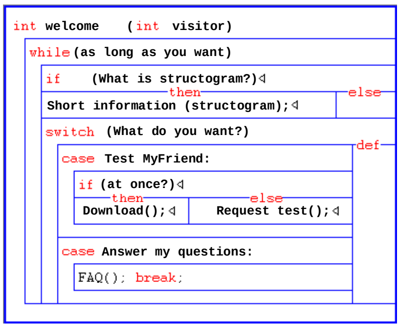
1.3. Structogram

Structograms

Graphic representation of the procedure:

➤ Structograms (DIN 66261)

Structograms (also called Nassi-Shneiderman diagrams) are one way of formulating abstract descriptions of algorithms. The program sequence is represented using standardized symbols that are not dependent on the program code used.



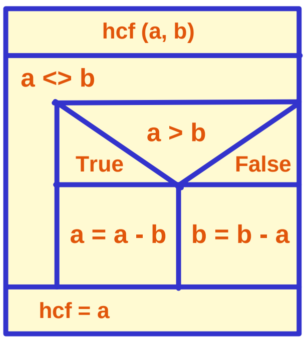
KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Structograms – comparison with programming language

Example of a structogram:

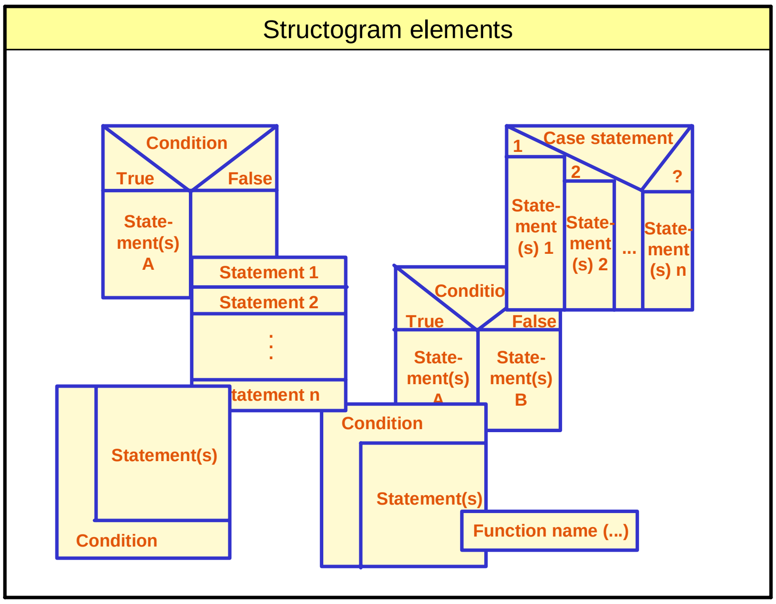
An algorithm is required for calculating the highest common factor of two numbers a and b. This is embedded in a function ‘hcf’ with two transfer parameters (a and b).

Structograms	KRL	Pascal

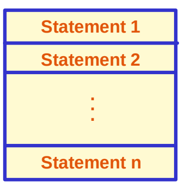
		function hcf (a,b:integer);
		while a<>b do
		If a>b
		then a:=a-b else b:=b-a;
		hcf:=a;

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

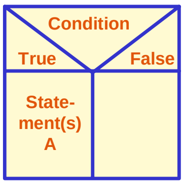
Structograms - overview of symbols



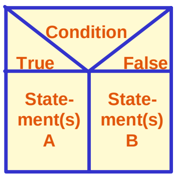
Structograms - symbols

Symbols	Statement	Description
	Statements in the program code	The statements are executed in the sequence Statement 1, Statement 2, ..., Statement n.

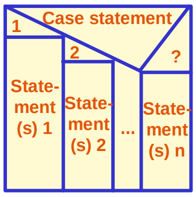
Structograms - symbols

Symbols	Statement	Description
	IF statement (without ELSE)	If the logic condition is met (true), statements A are executed, otherwise program execution is resumed at the next statement.

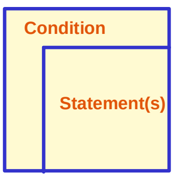
Structograms - symbols

Symbols	Statement	Description
	IF statement (with ELSE)	If the logic condition is met (true), statements A are executed, otherwise statements B are executed.

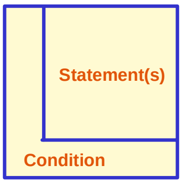
Structograms - symbols

Symbols	Statement	Description
	CASE statement	If CASE statement=1, statements 1 are executed. If CASE statement=2, statements 2 are executed, etc. In all other cases, statements n are executed.

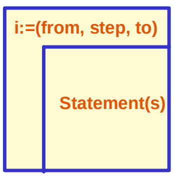
Structograms - symbols

Symbols	Statement	Description
	WHILE loop	The statements are executed while the condition is fulfilled. As soon as the condition is no longer met when execution of the loop is started again, program execution is resumed at the first statement after the loop.

Structograms - symbols

Symbols	Statement	Description
	REPEAT loop	The statements are executed until the condition is fulfilled. As soon as the condition is met, program execution is resumed at the first statement after the loop. The statements are always executed at least once.

Structograms - symbols

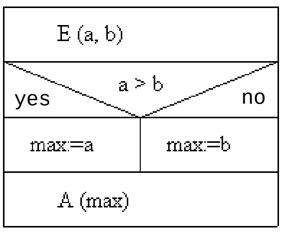
Symbols	Statement	Description
	FOR loop	The statements are executed a known number of times. The run variable i runs from the initial value (“from”) to the final value (“to”) with the specified increment (“step”). As soon as the run variable is greater than or equal to “to”, program execution is resumed at the next statement.

Structograms - symbols

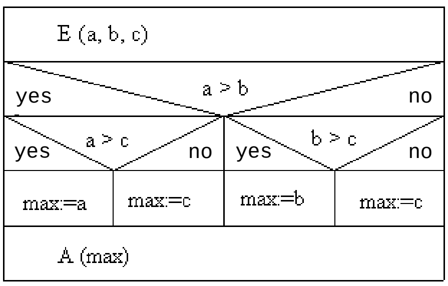
Symbols	Statement	Description
Function name (...)	Subprogram calls	A certain sequence of statements is segregated from a certain position in the program and implemented as a separate subprogram that can then be called from the program.

Structograms – example: selection

1. Simple selection



2. Selection

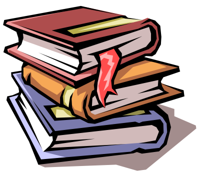


1.4. Programming methodology - Realization

Program planning representation

- This rough program flowchart serves as program specification on the one hand, and as program documentation on the other.

- The author of the rough flowchart should therefore always bear in mind that the documentation is primarily being created for others (end customers) rather than for his/her own use.



Program legibility

Transparency:

- Legibility makes it much easier to locate errors and to avoid them in the first place.

- Legibility makes it easier to understand and therefore also to modify programs.

- It is not sufficient to add comments to the text later. The following points must be considered during the programming itself.

Program legibility

Identification of data names:

- ➢ Only meaningful names should be used as data names.

- ➢ For the purposes of unambiguous identification, a prefix should be used (where appropriate), e.g. IN_IMM_READY.

Program legibility – Data name configuration

Identification of data names in the system:

1. Long texts

Outputs

	SYS	Ausgänge
	18	Output
	19	Gripper open
	20	Gripper close
	21	Output

Program legibility – Data name configuration

Identification of data names in the system:

2. Designation of BASE and TOOL

Basistyp (BASE_TYPE)

#	Typ	Name
1	undefiniert	
2	Offset	table
3	undefiniert	
4	undefiniert	
5	undefiniert	
6	undefiniert	
7	undefiniert	
8	undefiniert	
9	undefiniert	
10	undefiniert	
11	Offset	
12	undefiniert	
13	undefiniert	
14	undefiniert	
15	undefiniert	Ablageband
16	undefiniert	linke Ecke SGM

BASE designation

Werkzeugtyp (TOOL_TYPE)

#	Typ	Name
1	undefiniert	
2	Werkzeug	gripper
3	undefiniert	
4	undefiniert	
5	undefiniert	
6	undefiniert	
7	undefiniert	
8	undefiniert	
9	undefiniert	
10	undefiniert	
11	Werkzeug	
12	Werkzeug	
13	undefiniert	
14	undefiniert	
15	undefiniert	Vakuumgreifer 1
16	undefiniert	Zangengreifer 2

TOOL designation

Program legibility – Data name configuration

Identification of data names in the system:

2. Designation of BASE and TOOL

```
DEF LUFT ( )
...
PTP HOME  Vel= 100%  DEFAULT

PTP P3 CONT Vel= 100%  Tool[2]:Greifer  Base[2]: Tisch
PTP P4 CONT Vel= 100%  Tool[2]:Greifer  Base[2]: Tisch
PTP xP5
...
END
```


Program legibility – Data name programming

Identification of data names in KRL:

3. Using of signal-declaration

```
DEF PART_1 ( )
; Inputs / Outputs
SIGNAL machine_OK $IN[32]
SIGNAL Programnumber $IN[33] to $IN[40]
SIGNAL cycle_start $OUT[88]
…
IF machine_OK==TRUE THEN
cycle_start = TRUE
…
ENDIF
…
END
```


Program legibility

Indenting command sequences:

- In order to make explicit the structures within program modules, it is advisable to indent nested command sequences in the program text and write instructions at the same nesting depth directly below one another.

- The effect obtained is purely optical and serves merely to make the program more comprehensible to other people.

Program legibility

Indentation of command sequences in KRL

```
DEF INDENTION ( )
INT part, counter
INI
PTP HOME  Vel= 100%  DEFAULT
LOOP
   FOR counter = 1 TO 20
      part = part + 1
      ; Inlineforms are not able to indent !!!
PTP P3 CONT Vel= 100%  Tool[2]:Greifer  Base[2]: Tisch
PTP P4 CONT Vel= 100%  Tool[2]:Greifer  Base[2]: Tisch
      PTP xP5
   ENDFOR
…
ENDLOOP
END
```


Program legibility

Use of comments:

- Comments on their own cannot render a program legible; they can, however, significantly increase the legibility of a well-structured program.

- It is the responsibility of the programmer to ensure that the contents of the comments are up to date and correspond to the actual program instructions. If programs are modified, the comments must also be checked.

Program legibility

Use of comments in KRL:

The following comment commands can be selected:

File | Program | Configure | Monitor | Setup | Commands | Technology | Help

0 Last command
1 Motion ?
2 Moveparams ?
3 Logic ?
4 Analog output ?
5 Comment ?

0 Normal
1 Stamp

Any text line

Expanded text line

Program legibility – Comment: "NORMAL"

Text can be freely entered after selection of the command "NORMAL":

; Any text

Text box

Example:

```
...
INI
PTP HOME  Vel= 100%  DEFAULT
; Any text
PTP P1 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
...
```

When the command is selected again, the previously entered string will already be offered as a default text in the inline form.

Program legibility – Comment: "STAMP"

The name and a text can be freely entered after selection of the command "STAMP":

; 01.01.05 10:15 NAME	USER	CHANGES:	Any text

Name — Text box

Beispiel:

```
...
INI
PTP HOME  Vel= 100%  DEFAULT
; 01.01.05 10:15  NAME: User   CHANGES: Any text
PTP P1 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
...
```

With the Softkeys „New time", „New name" and „new text" you can jump to the choosed part and to actualise it.

Program legibility – Comment: "NORMAL EXPERT"

After switching to Expert mode, the programmer can turn any line into a comment by writing a "**;**" as the first character (exception: inline forms).

```
DEF CLEANING ( )
; ***************************************
; *** AUTOR Peter PAN             ***
; *** Last changing 05/05/05 ***
; ***************************************
INI
PTP HOME  Vel= 100%  DEFAULT
…
part = part + 1; part-counter increase
; drive to cleaning position
PTP P4 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
PTP xP5 ; Endposition cleaning
```

It is also possible to tag a comment onto the end of a command line using ";".

Program legibility

Use of FOLDs:

- Unlike normal editors, the KRC Editor allows a requirement-specific display of the program contents. The user only sees the important contents of a program, while at expert level the whole program is visible.

- The KUKA user interface uses so-called FOLDs to display a program clearly. This is done by enclosing the relevant declarations or instructions within the designations “*;FOLD*” and “*;ENDFOLD*”. FOLDs are closed by default with their contents hidden. FOLDs can be opened or closed using a menu.

Program legibility

Use of FOLDs – closed:

```
DEF CLEANING ( )
…
INI
IO_SETUP
PTP HOME  Vel= 100%  DEFAULT
…
PTP P21 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
PTP HOME  Vel= 100%  DEFAULT
END
```

USER FOLD

KUKA FOLD

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

Program legibility

Use of FOLD‘s – KUKA FOLD open

```
DEF REINIGEN ( )
…
INI
IO_SETUP
PTP HOME  Vel= 100%  DEFAULT
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
BAS(#PTP_DAT)
FDAT_ACT=FHOME
BAS(#FRAMES)
BAS(#VEL_PTP,100)
$H_POS=XHOME
PTP XHOME
…
PTP P21 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
PTP HOME  Vel= 100%  DEFAULT
END
```

KUKA FOLD open

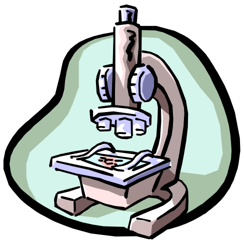
Program legibility

Use of FOLD‘s – USER FOLD open

```
DEF Bearbeitung ( )
…
INI
IO_SETUP
$OUT[12]=TRUE ;cell OK
part = 0 ; counter reset
glue = FALSE ; glueing off
position = 0 ; position-counter reset
PTP HOME  Vel= 100%  DEFAULT
…
PTP P21 CONT Vel= 100%  Tool[2]:Gripper  Base[2]: Table
PTP HOME  Vel= 100%  DEFAULT
END
```

USER FOLD open

Conversion into program structures



- As we have seen, not much is actually required for structured solutions to problems. The problem must simply be identified and broken down into steps. The individual steps can in turn be broken down into smaller steps.

- Depending on the nature of the problem, one of the following basic routines is required: conditional statements (branches), loops and data.

Conversion into program structures – Conditional statement, choice

- With [if (condition) ....; else; ...endif] statements, it is possible to recognize the one specific state that is important for continued program execution. These conditional statements are also called choices, branches or IF statements.

Syntax:

IF ... THEN

ELSE

ENDIF

Syntax:

SWITCH

CASE

ENDSWITCH

Conversion into program structures – Loops

- Loops can be used as a simple means of repeating part of a program. The number of repetitions can be specified in a counting loop or linked to a conditional statement.

Syntax:

- *FOR* *ENDFOR* (counting loop)
- *WHILE* *ENDWHILE* (rejecting loop)
- *REPEAT* *UNTIL* (non-rejecting loop)
- *LOOP* *ENDLOOP* (endless loop)

Conversion into program structures – General data

- Data are electronic saved, process able values or information
- There are simple data types, which have already a predefined valuation, like integer, floating-point number or logic state (true/false).
- Complex data types are e.g. structures (robot position)

Conversion into program structures – Programming

- An algorithm is a method for a schematic solution of a requirement. Therefore a robot programmer has the following routines:

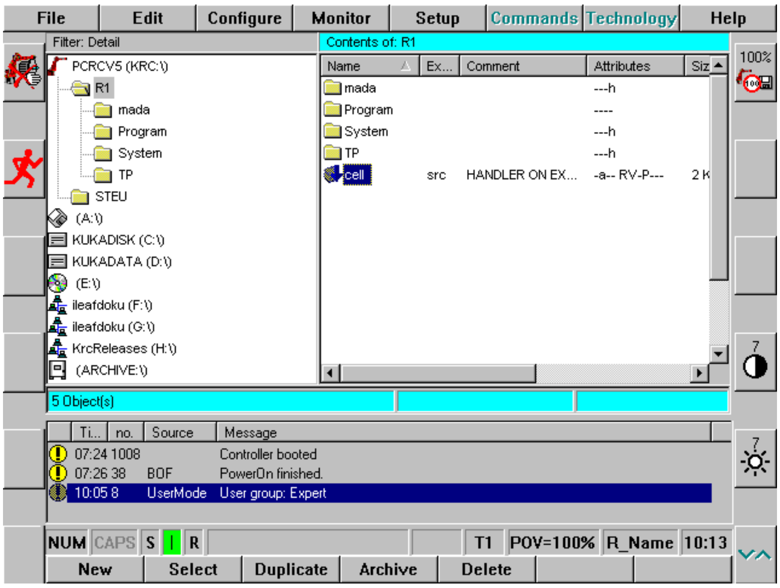
Syntax :

➢ *SUBPROG ( )*	Local or global subprogram
➢ *DEFFCT*	Function
➢ *INTERRUPT*	Process execution parallel to the robot program
➢ *TRIGGER*	Path-related switching actions

2. General inforamation about Expert level

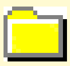
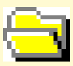
2.1. Navigator

Expert user group



 I 20.11.2003 I College I SG I 1

Symbols in the Navigator

Symbol	Meaning	Symbol	Meaning
	Folder closed		Module containing errors
	Folder open		SRC file containing errors
	Module		Dat file containing errors
	SUB file		
	SRC file		
	Dat file		

 I 20.11.2003 I College I SG I 2

Creating a new module

A KRL program can be made up of SRC and DAT files:
- SRC - contains program code
- DAT - contains specific program data

Cell	Skeleton program for control via a PLC
Expert	SRC and DAT file without a skeleton program
Expert Submit	SUB file without a skeleton program
Function	SRC file without a skeleton program
Module	SRC and DAT file with a skeleton program
Submit	SUB file with a skeleton program

Program run modes (1)

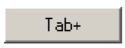
Run mode	Description
GO	**GO mode** All instructions in the progam are executed up to the end of the program without a STOP.
MSTEP	**Motion Step (motion block)** The program is executed one motion instruction at a time, i.e. with a STOP before each motion instruction. The program is executed without advance processing.
ISTEP	**Incremental Step (single block)** The program is executed step by step, i.e. with a STOP after each instruction (including blank lines). The program is executed without advance processing.

Program run modes (2)

Run mode	Description
PSTEP 	**Program Step** The program is executed step by step without advance processing. Subprograms are executed completely
CSTEP 	**Continuous Step (motion instruction)** Approximate positioning points are executed with advance processing, i.e. they are approximated. Exact positioning points are executed without advance processing and without a STOP after the motion instruction

The program run modes #PSTEP and #CSTEP can only be activated via the variable correction function ($PRO_MODE) and not using the status key.

Displaying machine data and version number



You can access further windows using the softkey “TAB”

2.2. Fold – Concept

Hiding program sections - FOLDS

- Program sections can be hidden in FOLDS.
- These program sections are processed during program execution.
- FOLDS are closed by default.
- The contents of FOLDS are not visible to the user.
- FOLDS can be nested.
- Sub-FOLDS are color-coded.

Color-coding of FOLDS:

- Closed FOLD
- Open FOLD
- Closed sub-FOLD
- Open sub-FOLD
- Contents of the FOLD

Hiding program sections - FOLDS

Syntax for creation of FOLD:

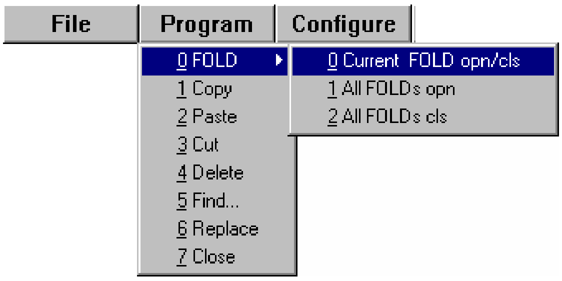
```
;FOLD Name
        Program line
        Program line
;ENDFOLD
```

```
int x
INI
;fold Home_position
$out[1]=true
counter=0
advance=35.0
;endfold
```

→

```
int x
INI
Home_position
```

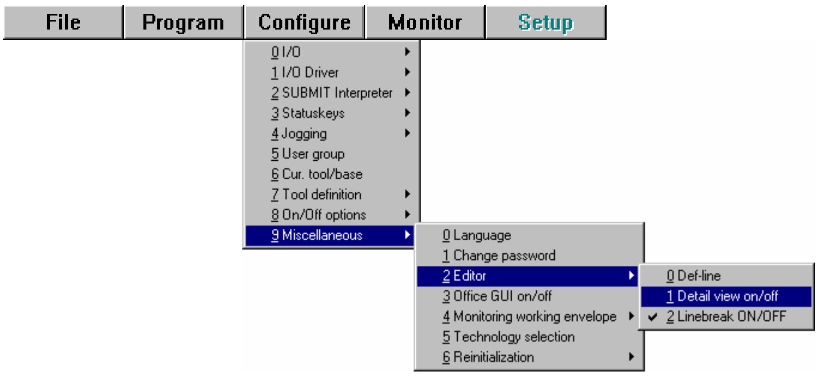
Hiding program sections - FOLDS




```
int x
INI
Outputs
```

```
int x
INI
Outputs
$out[1]=true
$out[2]=true
$out[3]=true
```


Hiding program sections - FOLDS




```
int x
INI
Outputs
$out[1]=true
$out[2]=true
$out[3]=true
```

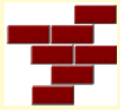
```
int x
INI
;fold Outputs
$out[1]=true
$out[2]=true
$out[3]=true
;endfold
```

3. Variables and declarations

3.1. Simple data types

Definition of “variables”

In addition to the use of constants (digits, characters, etc.), memory locations are also defined in the robot controller in which these values can be stored.



This stored information can thus be retrieved at various points in the program for evaluation and further processing.

A variable is a fixed memory area, whose contents can be addressed via the variable name.

Example:

```
...
Part_counter = 37
...
```

Variable name

Content of the variable

Declaration of variables

Variable declarations always have the following syntax:

DECL ***Data_Type Variable_Name***

<u>Example:</u>

```
Def Main( )
DECL INT part_counter
INI
...
End
```

Variables and names

Variable names in KRL

- can have a maximum length of 24 characters.
- can consist of letters (A-Z), numbers (0-9) and the signs '_' and '$'.
- must not begin with a number.
- must not be a keyword.

Simple data types

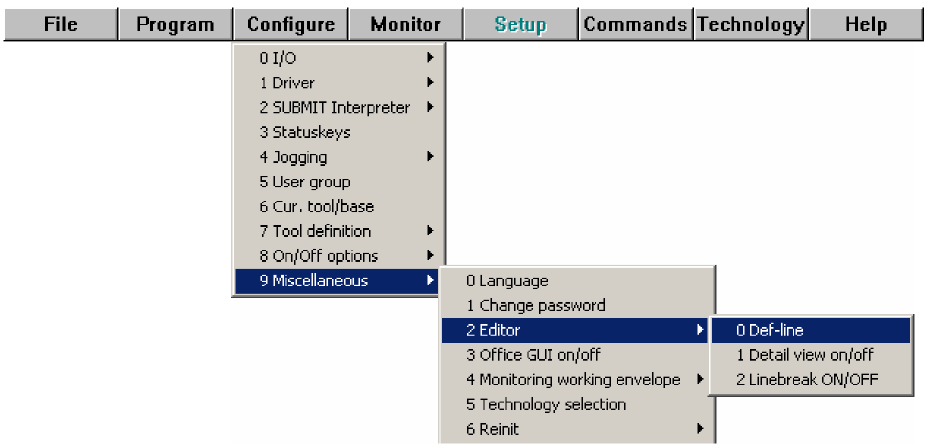
Data type	Integer	Real	Boolean	Character
Keyword	**INT**	**REAL**	**BOOL**	**CHAR**
Meaning	Integer	Floating-point number	Logic state	1 character
Range of values	-2^{31} … $(2^{31}-1)$	+/-1.1E-38 … +/-3.4E+38	TRUE, FALSE	ASCII character 1-255
Example	32	1.43	TRUE	"A"

Basic structure of a robot program

```
DEF NAME( )

    ;--- Declaration section ---
    ...

    ;--- Initialization section ---
    ...

    ;--- Instruction section ---
    ...

END
```


Activating the DEF line

- Variables must be declared before the INI line
- In order to set syntax before the INI line, the DEF line must be activated



→→

Declaration and value assignment

```
DEF MAIN_PROGRAM( )
          ;---  Declaration section  ---
DECL INT PART
          ;---  Initialization section  ---
INI                 ;Initialization of acceleration and velocity
          ;---  Instruction section  ---
PART=58             ;Value assignment, decimal
PART='B111010'      Value assignment, binary
PART='H3A'          ;Value assignment, hexadecimal
...
END
```

Value assignments are made in the program advance run!

Lifetime of variables

Variables declared ...

in the SRC file

- ... are only recognized in their own program.
- ... are not recognized in local subprograms.

```
DEF HP1( )
DECL INT CUBE
INI
CUBE=0
END
```

in the corresponding DAT file

- ... are only recognized in their own program, i.e. in the main program and in local subprograms.

```
DEFDAT HP1( )
DECL INT CUBE=0
...
ENDDAT
```

in the $CONFIG.DAT file

- ... are recognized in every program.
- ... can be called during program execution.

```
DEFDAT $CONFIG
DECL INT CUBE=0
...
ENDDAT
```

Declaration of constants

Constant declarations always have the following syntax and they could only declared in a data list:

DECL CONST *Datentyp Variable_Name*

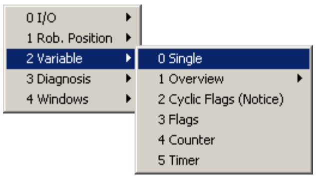
Beispiel:

```
Defdat  Main( )
Decl CONST REAL PI = 3.14159
...
End
```

You can not manipulate Constants in the SRC-file. It causes a error message.

3.2. Monitoring of variables

Variable display – Single

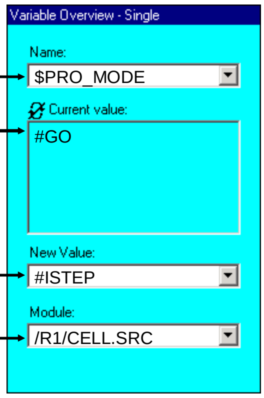


Current variable name; reachable over softkey „NAME“

Current value of the variable of the field „Name“

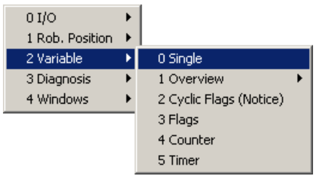
New value of the variable of the field „Name“

Folder of the program



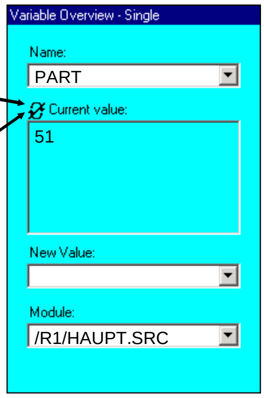
KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Variable display – Single



Current value of the variable will only updated, when you pressed in the field „Name“ the „Return-Key“.

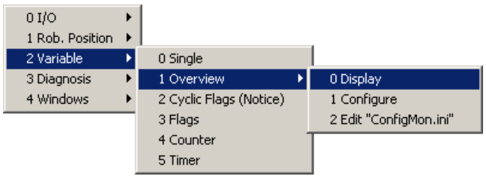
Current value of the variable will automatically be updated, when you pressed in the field „Name“ the „Shift+Return-Keys“ together.



Attention: Limited functionality with runtime variables

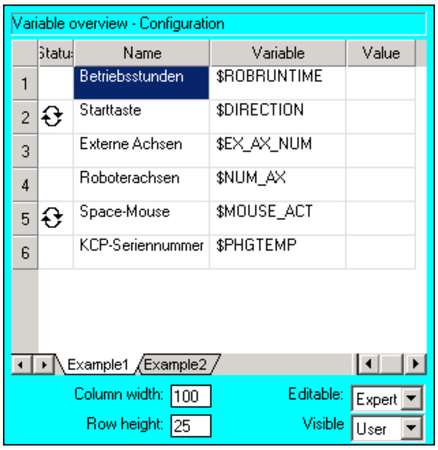
KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

Variable display – Configure

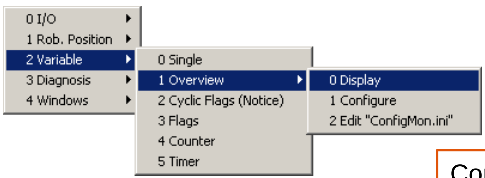


To use the variable in variable display it have to be:

- Declared in the $config.dat
- KUKA-System variable



Variable display – Configure

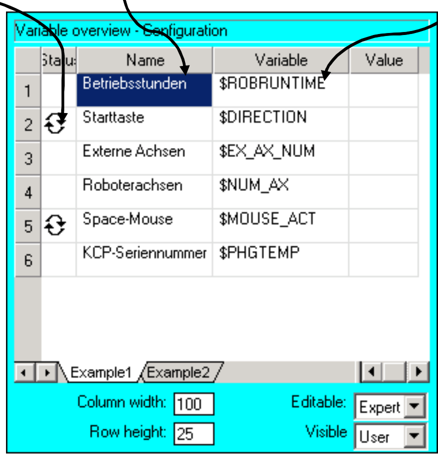


Continually refreshed

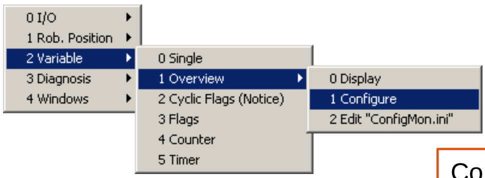
Description of the variable

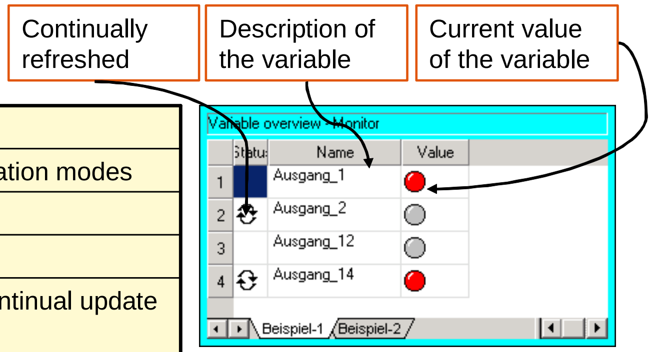
Variable name used

Softkeys	Meaning
Display	Switch to the display modes
Tab	Next group/worksheet
Jump	Jump to the group parameters (e.g. column width)
Insert	Open "Insert" menu (insert before/after group/line)
Delete	Delete a group/line
OK	Confirm, save and close configuration
Cancel	Close window without saving



Variable display – Display





Softkeys	Meaning
Configure	Switch to the configuration modes
Tab	Next group/worksheet
Refresh all	Static update
Start info / Cancel info	Activate/deactivate continual update
Edit	Change the name or value of the variable (writing access)
OK	Confirm, save and close configuration
Cancel	Close window without saving

3.3. Arrays

Arrays (one-dimensional)

Syntax:

Creation:
DECL *Data_Type* *Variable_Name[Number of array elements]*
Work:
Variable_Name[Index] = Value_Assignment

Example:

```
DEF MAIN_PROGRAM( )
DECL REAL measurement[3]

INI

measurement[1] = 17.5
measurement[2] = 35.7
measurement[3] = 67.2
...
END
```

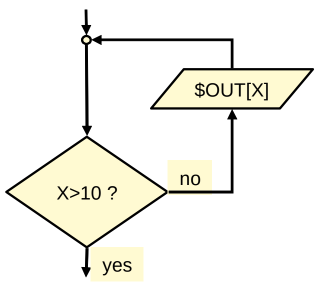

Simple counting loop

Syntax:

FOR *Counter* **=** *Start* **TO** *End*
  Statement
ENDFOR

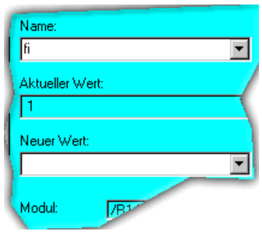
Example:

```
DEF INIT_OUTPUTS ( )
DECL INT COUNTER
INI

FOR COUNTER=1 TO 10
        ;Set output 1-10 to FALSE
        $OUT[x]=FALSE
ENDFOR
...
END
```



Example: Simple counting loop (1)

```
DEF MAIN_PROGRAM( )
DECL INT CELL[4]
DECL INT FI                 ;Array index
INI

FOR FI = 1 TO 4
  CELL[FI] = FI * 5
ENDFOR
...
END
```

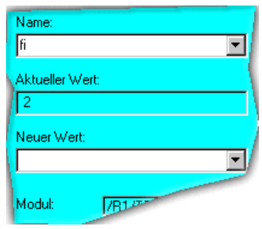


1 * 5 = 5

CELL	[1]	[2]	[3]	[4]
	5			

Example: Simple counting loop (2)

```
DEF MAIN_PROGRAM( )
DECL INT CELL[4]
DECL INT FI                 ;Array index
INI

FOR FI = 1 TO 4
  CELL[FI] = FI * 5
ENDFOR
...
END
```

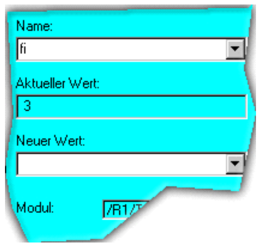


2 * 5 = 10

CELL	[1]	[2]	[3]	[4]
	5	10		

Example: Simple counting loop (3)

```
DEF MAIN_PROGRAM( )
DECL INT CELL[4]
DECL INT FI                ;Array index
INI

FOR FI = 1 TO 4
  CELL[FI] = FI * 5
ENDFOR
...
END
```



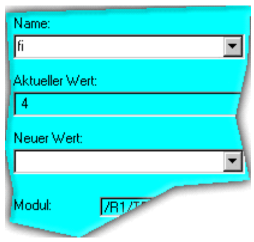
3 * 5 = 15

CELL				
	5	10	15	
	[1]	[2]	[3]	[4]

 I 20.11.2003 I College I SG I 5

Example: Simple counting loop (4)

```
DEF MAIN_PROGRAM( )
DECL INT CELL[4]
DECL INT FI                ;Array index
INI

FOR FI = 1 TO 4
  CELL[FI] = FI * 5
ENDFOR
...
END
```



FI = 5

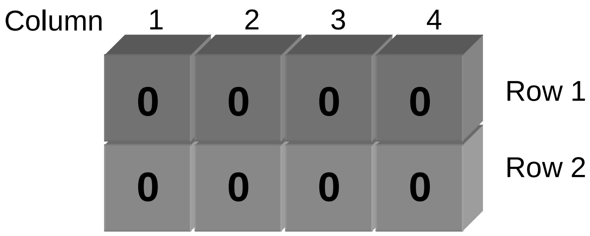
4 * 5 = 20

CELL				
	5	10	15	20
	[1]	[2]	[3]	[4]

 I 20.11.2003 I College I SG I 6

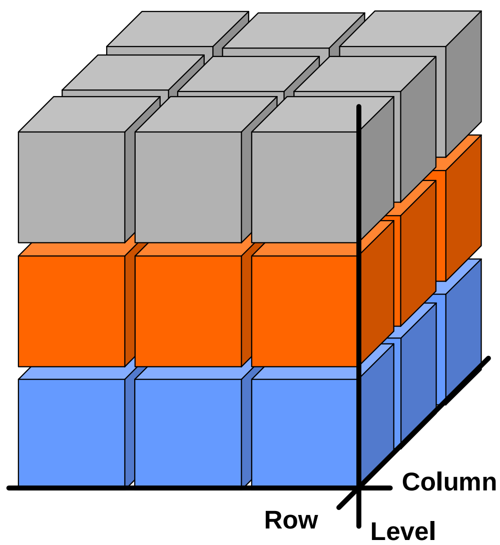
Two-dimensional array

```
DEF MAIN_PROGRAM( )
DECL INT MEAS_VALUE [2,4]
DECL INT ROW,COLUMN
INI
     ; --- Pre-assignment of an array ---
FOR ROW = 1 TO 2
  FOR COLUMN = 1 TO 4
     MEAS_VALUE [ROW,COLUMN] = 0
  ENDFOR
ENDFOR
...
END
```



Example: Three-dimensional array

BOOL MATRIX [3, 3, 3]



Three-dimensional array

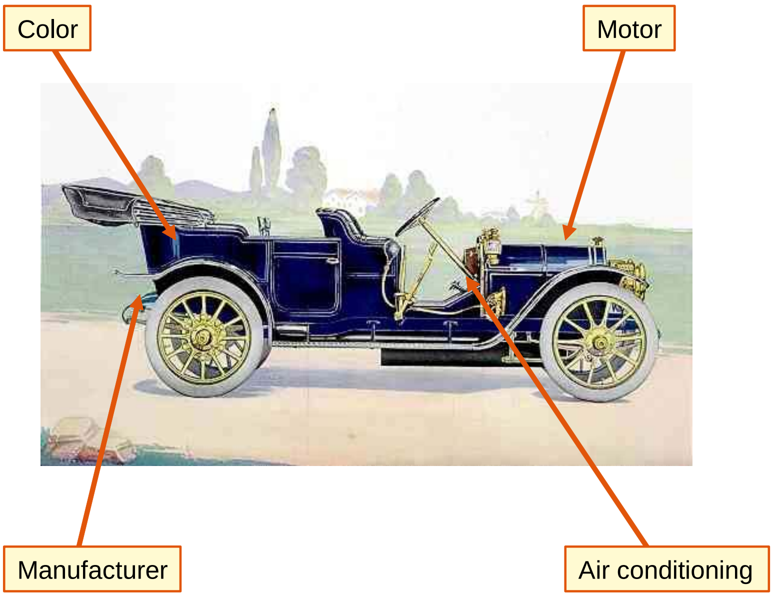
```
DEF MAIN_PROGRAM( )
BOOL MATRIX [3,3,3]
INT ROW, COLUMN, LEVEL
INI

FOR LEVEL = 1 TO 3
  FOR COLUMN = 1 TO 3
    FOR ROW = 1 TO 3
      MATRIX [ROW, COLUMN, LEVEL] = FALSE
    ENDFOR
  ENDFOR
ENDFOR
...

END
```

The keyword “DECL” may also be omitted when declaring **simple data types**.

3.4. Structures

Structures

Example: Exact definition of a car



Structures

- A structure is a combination of different data types.
- A structure is initialized with an aggregate.
- Not all the parameters have to be specified in an aggregate.
- The order of the parameters is insignificant.

User-defined structures should end in ...TYPE!

User-defined structure

Name of the structure

Data type

Variable name

```
DEF WELD ( )
STRUC WELDTYPE REAL V_WIRE, INT CHARAC, BOOL PULSE
DECL WELDTYPE SEAM1
INI
SEAM1={V_WIRE 0.7, CHARAC 5, PULSE FALSE}
...
SEAM1.PULSE=TRUE
...
END
```

Modification of a value in an aggregate using a point separator

Predefined structures

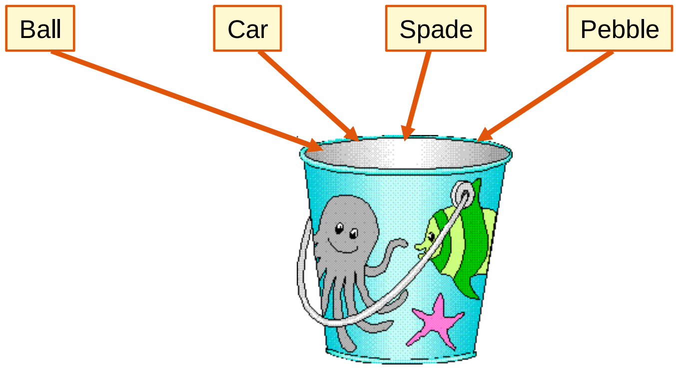
The following structures are predefined in the KUKA robot system:

STRUC	**AXIS**	REAL A1,A2,A3,A4,A5,A6
STRUC	**E6AXIS**	REAL A1,A2,A3,A4,A5,A6,E1,E2,E3,E4,E5,E6
STRUC	**FRAME**	REAL X,Y,Z,A,B,C
STRUC	**POS**	REAL X,Y,Z,A,B,C INT S,T
STRUC	**E6POS**	REAL X,Y,Z,A,B,C,E1,E2,E3,E4,E5,E6, INT S,T

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

3.5. Enumeration data types

Enumeration data type - ENUM



Enumeration data type

- An enumeration data type consists of a limited number of constants
- The constants are freely definable names
- The constants are defined by the programmer
- A variable can only take on the value of **one** of these constants

User-defined ENUM variables should end in **...TYPE**!

© Copyright by KUKA Roboter GmbH College

Enumeration data type

The following ENUM variable is predefined in the KUKA robot system:

ENUM MODE_OP T1, T2, AUT, EX, INVALID

```
DEF SEARCH ()
ENUM COUNTY_TYPE  German, English, French
DECL COUNTY_TYPE  LANGUAGE
INI
...
LANGUAGE = #German
...
LANGUAGE = #English
...
END
```

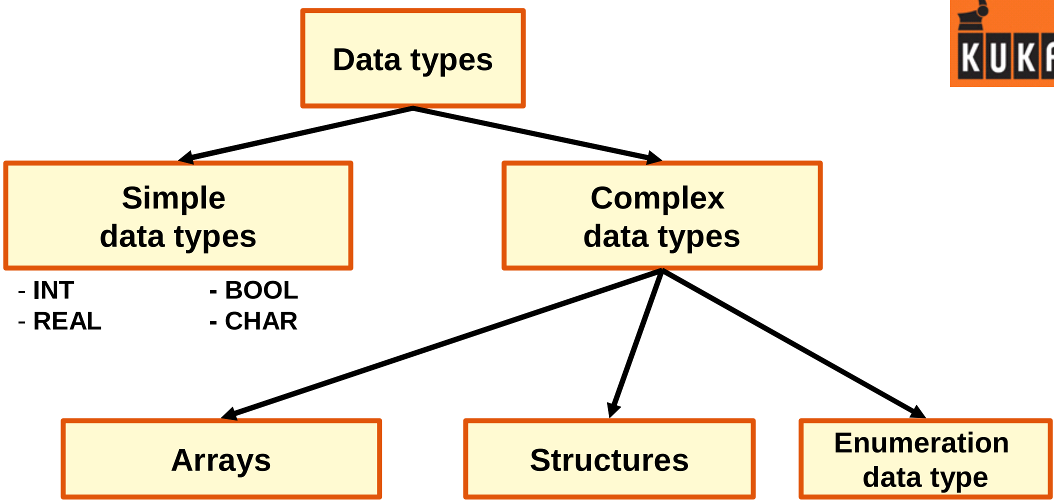
Generate enumerating variable

Declare enumerating variable

Value assignment

Value assignment

Overview of the data types



All array elements
- have the same data type
- have the same name
- differentiation only in their own index number

All structure elements
- could have different data types
- have different terms

A ENUM variable
- can have only one of the previously defined constants

4. Subprograms and functions

4.1. Definition of subprograms

Local subprograms

- A KRL file can consist of up to 255 local subprograms.
- The maximum nesting depth for subprograms is 20.
- Local subprograms are located after the main program and are identified by means of DEF and END.
- Local subprograms can be called repeatedly.
- Point coordinates are saved in the corresponding DAT list and are available for the entire file.
- Once a local subprogram has been executed, the program jumps back to the next command after the subprogram call.

Local subprograms

```
DEF Main_Program ( )
INI
...
Subprogram1 ( )
...
Subprogram2 ( )
...
END
DEF Subprogram1 ( )
...
Subprogram2 ( )
...
END
DEF Subprogram2 ( )
...
END
```

Global subprograms

- Global subprograms consist of a separate SRC file.
- Variables declared in global subprograms are not recognized in the main program.
- Variables declared in the main program are not recognized in the global subprogram.

```
DEF MAINPROG ( )
INI
...
GLOBAL_SP1()
...
END
```

```
DEF GLOBAL_SP1 ( )
INI
...
...
END
```


Return from the subprogram

RETURN

- The RETURN function ends the execution of the subprogram and causes the system to return to the calling module.

```
DEF MAINPROG ( )
INI
...
GLOBAL_SP1()
...
END
```

```
DEF GLOBAL_UP1 ( )
INI
...
IF value>15 THEN
RETURN
ENDIF
…
END
```

4.2. Transfer of parameter

Variable parameter transfer

Main_program ( )

X=21.0
Y=TRUE
Calc (X,Y)

Calc (……)

Variable parameter transfer

There are two different transfer options:

IN parameter
(call by value)

The value of the variable remains unchanged in the main program, *i.e. it continues to be processed with the old value from the main program*

OUT parameter
(call by reference)

The value of the variable is changed in the main program, *i.e. the value is taken from the subprogram*

Variable transfer to local subprograms

Syntax:

Program call:
Subprogram_Name **(*Transfer_Variable*)**
Subprogram header:
DEF *Subprogram_Name* (*Variable_Name: Call by*)

Example:

```
DEF MAIN ( )
REAL AMOUNT
INI
...
CALC (AMOUNT)
END
```

```
DEF CALC (SUM:OUT)
REAL SUM
...
END
```


Variable transfer to global subprograms

Syntax:

Program call:
Subprogram_Name **(*Transfer_Variable*)**
Subprogram header:
DEF *Subprogram_Name* (*Variable_Name: Call by*)

Example:

```
DEF MAIN ( )
REAL AMOUNT
INI
...
CALC (AMOUNT)
END
```

```
DEF CALC (SUM:OUT )
REAL SUM
INI
...
END
```

Example of variable transfer

```
DEF PROG_1 ( )
INT A, B
REAL C
A = 1
B = 2
C = 3.0

CALC (A, B, C)

HALT; A is now 11
      ; B is now 2
      ; C is now 3.0
END
```

Sequence is important!

```
DEF CALC (X1: OUT, X2: IN, X3: IN)
INT X1, X2
REAL X3
X1 = X1+10   ; X1 is now 11
X2 = X2+10   ; X2 is now 12
X3 = X3+10   ; X3 is now 13.0
END
```

<u>Two types of parameter list</u>
Call by value: **IN**
Call by reference: **OUT**

The use of parameter lists is necessary in both
LOCAL
and
GLOBAL
subprograms.

Array parameter transfer

- Arrays may **only** be transferred with parameter OUT (Call by reference)
- Arrays may only be transferred with the dimension specified, e.g. ARRAY_1D[ ] , ARRAY_2D[,] , ARRAY_3D[,,]

<u>Example:</u>

```
DEF MAIN ( )
CHAR NAME[8]
INI
...
SP1 (NAME[ ])
END
DEF SP1 (FIRST_NAME[ ]:OUT)
CHAR FIRST_NAME[ ]
...
END
```

4.3. Function

Functions

Input values → **Function**

Arithmetical algorithm

Return statement

→ Output value

Example: using camera correction position → calculate pickup position

Functions

- A function is a subprogram that returns a certain value to the main program.
- The program name is also the variable name of a certain data type.
- The value to be transferred must be returned by means of the RETURN(x) statement before the ENDFCT statement.

Syntax:

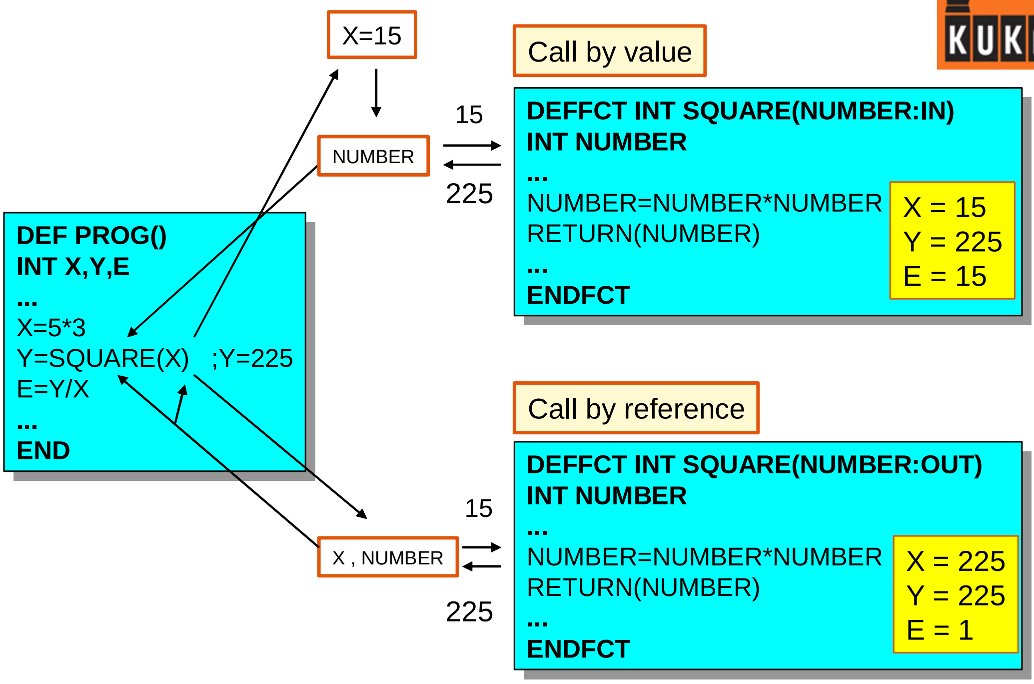
DEFFCT *Data_Type NAME_FUNCTION ( )*

...

RETURN(*x*)

ENDFCT

Function with IN and OUT parameters



5. Data lists

Data lists - local

Data lists are used for preparing program-specific and higher-level declarations (variables, coordinates, etc.)

Source file (SRC file)

```
DEF PROG_1 ( )
...
HALT
...
OTTO = 25
...
HALT
END
```

Value assignment

Data list (DAT file)

```
DEFDAT PROG_1
INT OTTO = 0
ENDDAT
```

Declaration and initialization

```
DEFDAT PROG_1
INT OTTO = 25
ENDDAT
```

The value remains unchanged after program execution

Data lists - global

Certain system files are global data lists, i.e. variables, signals and declarations are valid for all programs.

$CONFIG.DAT

```
DEFDAT $CONFIG
:
E6AXIS HOME {A1 0.0,A2 -90.0,...}
:
ENDDAT
```

$MACHINE.DAT

```
DEFDAT $MACHINE
:
$SOFTP_END[1] = 185.0
:
$SOFTP_END[6] = 350.0
:
ENDDAT
```

Importing variables

Variables defined in a data list can be imported into a “foreign” main program.

PROG_1.SRC

```
DEF PROG_1 ( )
...
HALT
...
OTTO = 25
...
END
```

PROG_1.DAT

```
DEFDAT PROG_1 PUBLIC
DECL INT OTTO = 0
...
ENDDAT
```

PROG_2.SRC

```
DEF PROG_2 ( )
IMPORT INT OTTO_2 IS /R1/PROG_1 .. OTTO
...
...
END
```

- The contents of the variable cannot be modified
- Make sure that the path is spelled correctly

Global variables

Variables defined in a data list can be made globally accessible to a “foreign” main program.

PROG_1.SRC

```
DEF PROG_1 ( )
...
HALT
...
OTTO = 25
...
END
```

PROG_1.DAT

```
DEFDAT PROG_1 PUBLIC
DECL  GLOBAL INT OTTO = 0
...
ENDDAT
```

PROG_2.SRC

```
DEF PROG_2 ( )
...
OTTO = 100
...
END
```

The contents of the variable can be modified

6. Data manipulation

Arithmetic operators

Basic arithmetic operations for the data types INTEGER and REAL:

Operator	Description
+	Addition or positive sign
-	Subtraction or negative sign
*	Multiplication
/	Division

Result of an arithmetic operation:

Operands	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

Program example

```
DEF ARITH()
;------- Declaration section --------
INT A,B,C
REAL K,L,M
;-------- Initialization section -------   ;All variables are invalid prior to initialization!
A = 2                                      ;A=2
B = 9.8                                    ;B=10
C = 7/4                                    ;C=1
K = 3.5                                    ;K=3.5
L = 0.1 E01                                ;L=1.0
M = 3                                      ;M=3.0
;------------ Main section-----------
A = A * C                                  ;A=2
B = B - 'HB'                               ;B=-1
C = C + K                                  ;C=5
K = K * 10                                 ;K=35.0
L = 10 / 4                                 ;L=2.0
L = 10 / 4.0                               ;L=2.5
L = 10 / 4.                                ;L=2.5
L = 10./ 4                                 ;L=2.5
C = 10./ 4.                                ;C=3
M = (10/3) * M                             ;M=9.0
END
```

Relational operators

Using relational operators, it is possible to form logic expressions. The result of a comparison is always of data type BOOL.

Operator	Description	Permissible data types
==	equal to	INT, REAL, CHAR, ENUM, BOOL
<>	not equal to	INT, REAL, CHAR, ENUM, BOOL
>	greater than	INT, REAL, CHAR, ENUM
<	less than	INT, REAL, CHAR, ENUM
>=	**greater than/equal to**	INT, REAL, CHAR, ENUM
<=	**less than/equal to**	INT, REAL, CHAR, ENUM

Example:

```
BOOL A,B
...
B = 10 < 3                                  ;B=FALSE
A = 10/3 == 3                               ;A=TRUE
B = ((B == A) <> (10.00001 >= 10)) == TRUE  ;B=TRUE
```


Logic operators

Logic operators are used for logic operations with Boolean variables, constants and simple logic expressions.

Operator	Operand number	Description
NOT	1	Inversion
AND	2	Logic AND
OR	2	Logic OR
EXOR	2	Exclusive OR

Example:

```
...
BOOL A,B,C
...
A = TRUE                             ;A=TRUE
B = NOT A                            ;B=FALSE
C = (A AND B) OR NOT (B EXOR NOT A)  ;C=TRUE
A = NOT NOT C                        ;A=TRUE
...
```

Truth table

Truth table for logic operations:

Operation		NOT A 1	A AND B &	A OR B ≥1	A EXOR B =1
A=TRUE	**B=TRUE**	FALSE	TRUE	TRUE	FALSE
A=TRUE	**B=FALSE**	FALSE	FALSE	TRUE	TRUE
A=FALSE	**B=TRUE**	TRUE	FALSE	TRUE	TRUE
A=FALSE	**B=FALSE**	TRUE	FALSE	FALSE	FALSE

Bit operators

Bit operators are used to perform logic operations on the individual bits of whole numbers.

Operator	Operand number	Description
B_NOT	1	Bit-by-bit inversion
B_AND	2	Bit-by-bit AND operation
B_OR	2	Bit-by-bit OR operation
B_EXOR	2	Bit-by-bit exclusive OR operation

As ASCII characters can also be addressed via the integer ASCII code, the data type of the operands may also be CHAR besides INT. The result is always of type INT.

Bit operators

Values	2^3	2^2	2^1	2^0	
	8	4	2	1	

Example:

1st number	0	1	0	1	=5
2nd number	1	1	0	0	=12
B_AND	0	1	0	0	=4
B_OR	1	1	0	1	=13
B_EXOR	1	0	0	1	=9

Bit-by-bit inversion does not simply involve all the bits being inverted. Instead, 1 is added to the operand and the sign is changed, e.g.:
B_NOT 10 = -11 or **B_NOT -10 = 9**

Priority of operators

Operators will be executed in order of priority.

Priority	Operator
1	NOT B_NOT
2	* /
3	+ -
4	AND B_AND
5	EXOR B_EXOR
6	OR B_OR
7	== <> < > >= <=

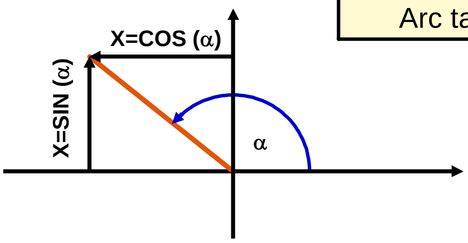
Example:

```
INT A,B
BOOL E,F
A = 4
B = 7
E = TRUE
F = FALSE
E = NOT E OR F AND NOT (-3 + A * 2 > B)      ;E=FALSE
```

Standard functions

Standard functions for calculating mathematical problems:

Description	Function
Absolute value	ABS (X)
Square root	SQRT (X)
Sine	SIN (X)
Cosine	COS (X)
Tangent	TAN (X)
Arc cosine	ACOS (X)
Arc tangent	ATAN2 (Y, X)

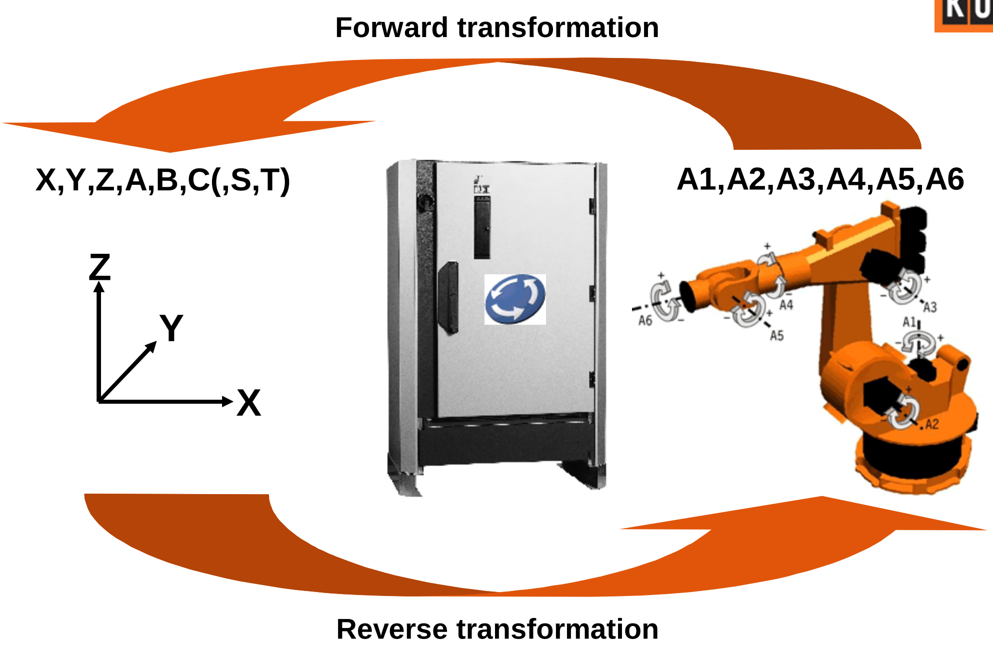


KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

7. Motion programming

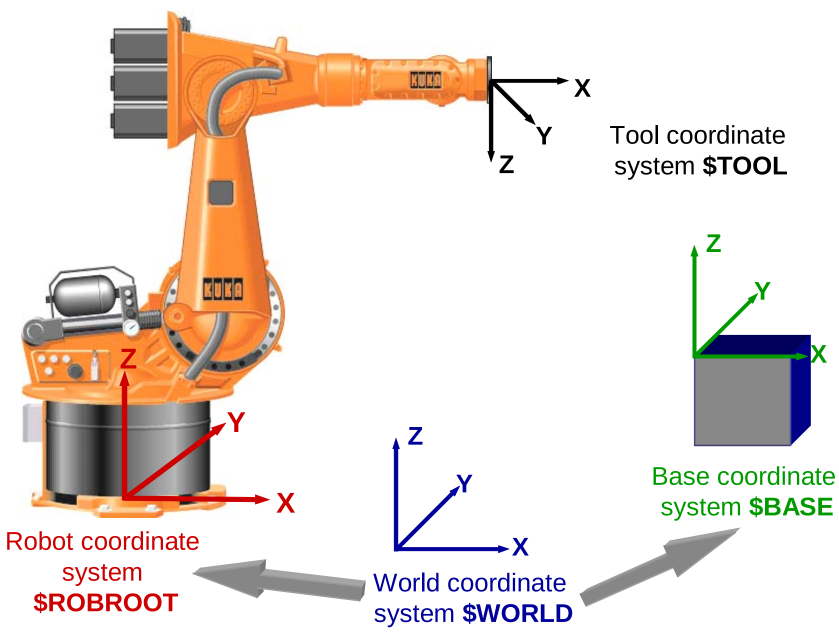
7.1. Coordinate systems

Coordinate transformation at interpolation cycle rate (12 ms)



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 20.11.2003 I College I SG I 1

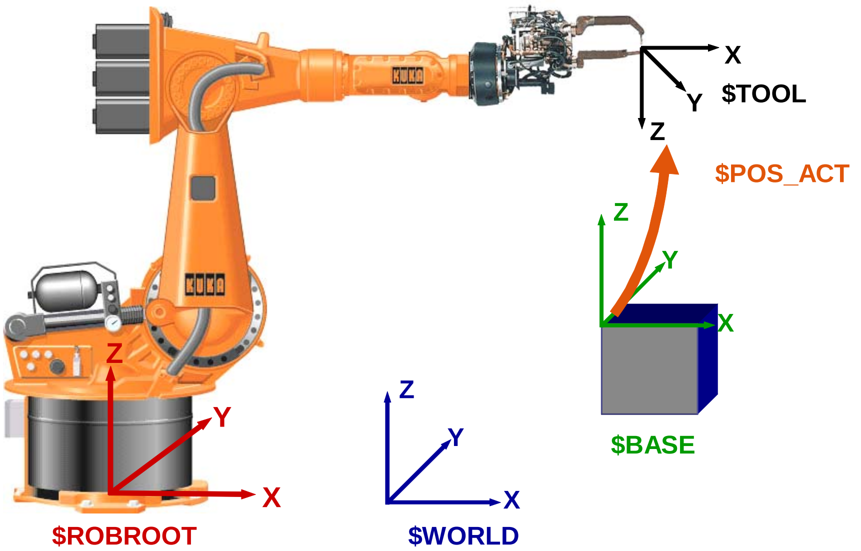
Cartesian coordinate systems



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 20.11.2003 I College I SG I 2

Kinematic sequence with base-related interpolation

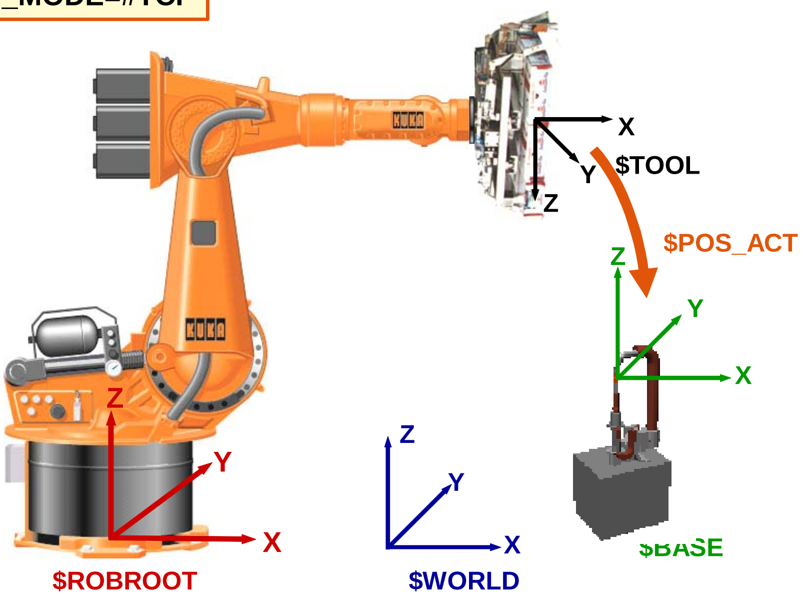
$IPO_MODE=#BASE



I 20.11.2003 I College I SG I 3

Kinematic sequence with tool-related interpolation

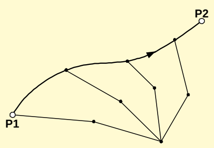
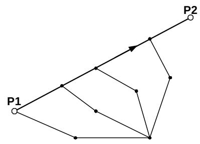
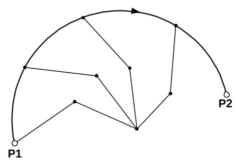
$IPO_MODE=#TCP



I 20.11.2003 I College I SG I 4

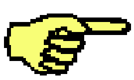
7.2. Axis motion – PTP

Overview of motion types

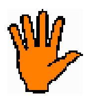
KUKA robot motion types (interpolation types)		
PTP	**LIN**	**CIRC**
Axis motion	*CP motion*	
Quickest motion between two points. The controller calculates the required angle differences.	The TCP of the currently active tool is moved along a straight line from the start point to the end point.	The TCP of the currently active tool is moved along a defined circular path.
		

BCO run

- Before a program can be executed, block coincidence (BCO) must first be established.
- This is done by holding down the Start key after selecting the program.
- The robot moves automatically at reduced velocity
- Once the robot has reached the programmed path, the program can be continued by pressing the Start key again

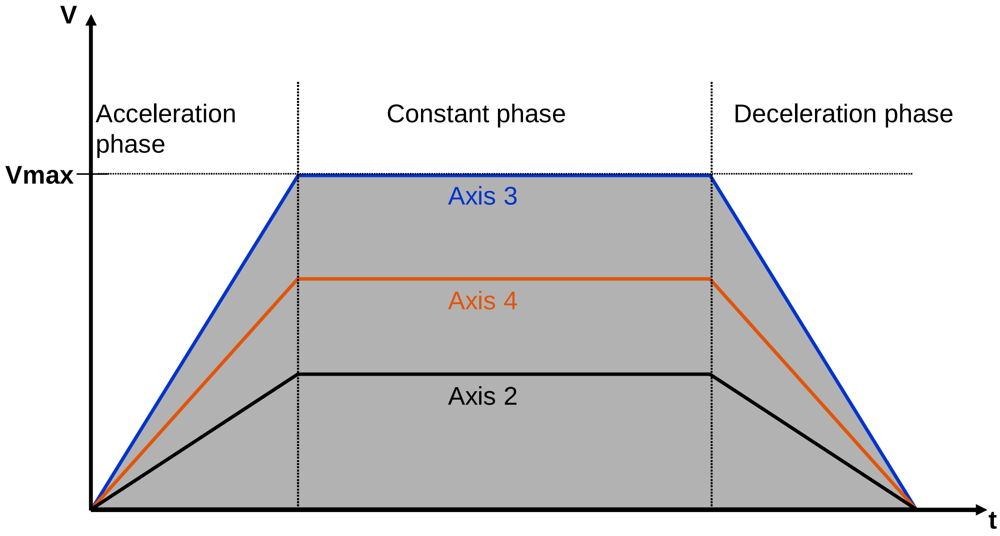


A HOME run is recommended for both the first and final motions as this represents an unambiguously defined uncritical position.



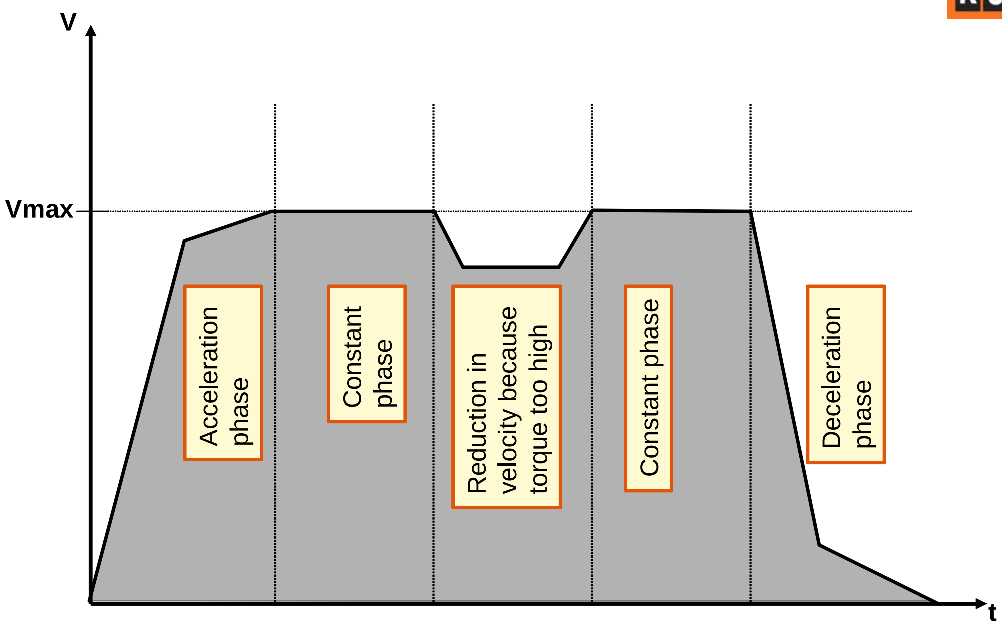
No BCO run is carried out in Automatic External mode!

PTP motion - simplified



PTP motion – higher motion profile

The higher motion profile is used as standard for PTP motions.



PTP motions - axis-specific

The axis velocities and accelerations must be initialized before a PTP motion can be executed. Values are specified in %.

```
DEF PTP_AXIS ( )
INT X
INI                        ;Initialization
...
FOR X=1 TO 6
  $ACC_AXIS[X]=70          ;Modification of the axis accelerations
  $VEL_AXIS[X]=70          ;Modification of the axis velocities
ENDFOR
...
PTP {AXIS: A1 0, A2 –90, A3 90, A4 0, A5 0, A6 0}
...
END
```

 I 20.11.2003 I College I SG I 5

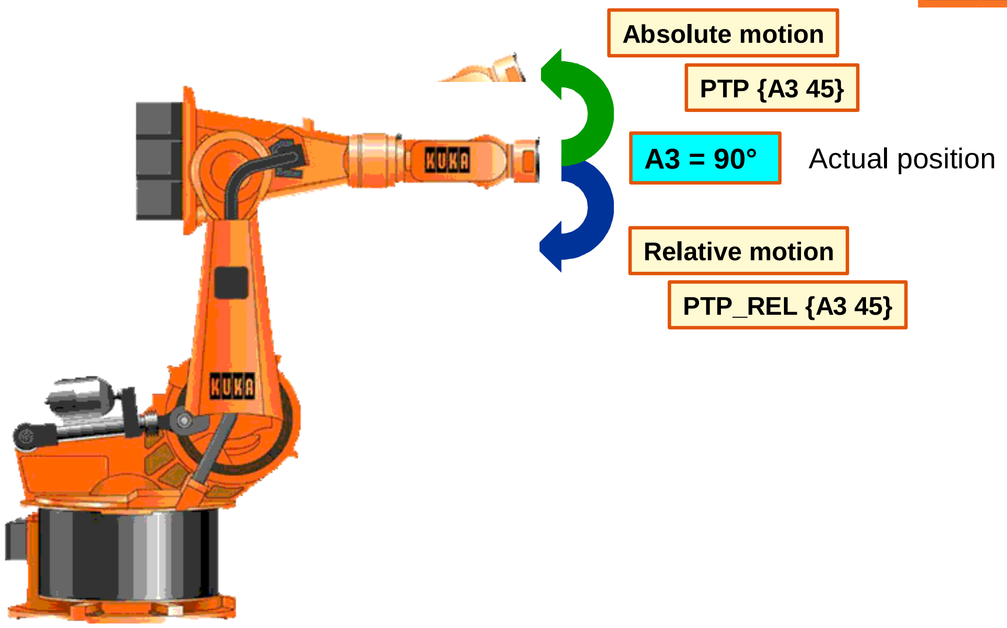
PTP motions - Cartesian

The base and tool coordinate systems must also be defined when entering the end point using Cartesian coordinates.

```
DEF PTP_AXIS ( )
INT X
INI
$BASE=BASE_DATA[1]  ;Setting the base coordinate system
$TOOL=TOOL_DATA[3]  ;Setting the tool coordinate system
FOR X=1 TO 6
  $ACC_AXIS[X]=70          ;Modification of the axis accelerations
  $VEL_AXIS[X]=70          ;Modification of the axis velocities
ENDFOR
...
PTP {POS:X 1025, Y 0, Z 1480, A 0, B 0, C 0}
...
END
```

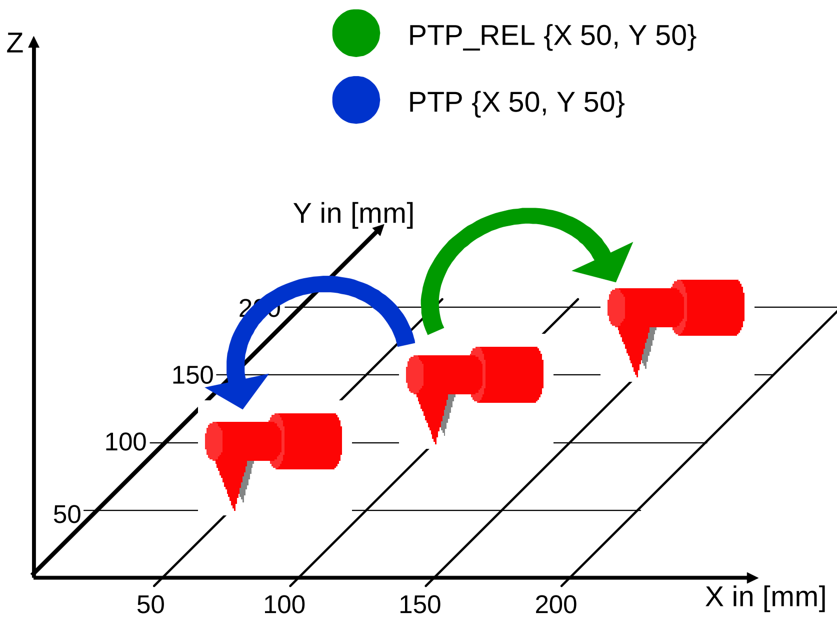
 I 20.11.2003 I College I SG I 6

Axis-specific absolute and relative motion



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

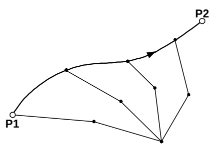
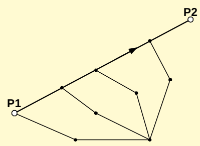
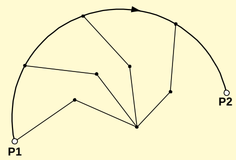
Cartesian absolute and relative motion



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

7.3. Continuous path motion – LIN und CIRC

Overview of motion types

KUKA robot motion types (interpolation types)		
PTP	**LIN**	**CIRC**
Axis motion	*CP motion*	
Quickest motion between two points. The controller calculates the required angle differences.	The TCP of the currently active tool is moved along a straight line from the start point to the end point.	The TCP of the currently active tool is moved along a defined circular path.
		

CP motions (CP = Continuous Path)

The path, swivel and rotational velocities and accelerations must be initialized before a CP motion can be executed.

	Variable name	Data type	Unit	Function
Velocity	**$VEL**.CP	REAL	m/s	Path velocity
	$VEL.ORI1	REAL	°/s	Swivel velocity
	$VEL.ORI2	REAL	°/s	Rotational velocity
Acceleration	**$ACC**.CP	REAL	m/s²	Path acceleration
	$ACC.ORI1	REAL	°/s²	Swivel acceleration
	$ACC.ORI2	REAL	°/s²	Rotational acceleration

Path-related absolute and relative motion

When the initialization sequence of the basic package is called, the default settings for the velocities and rates of acceleration of CP motions are also preset to the maximum values defined in the machine data or $CONFIG.DAT.

```
DEFDAT $CONFIG
...
; CP - MOVEMENTS
;---------------------------------
DECL CIRC_TYPE DEF_CIRC_TYP=#BASE
REAL DEF_VEL_CP=2.00
REAL DEF_VEL_ORI1=200.0
REAL DEF_VEL_ORI2=200.0
REAL DEF_ACC_CP=2.3
REAL DEF_ACC_ORI1=100.0
REAL DEF_ACC_ORI2=100.0
...
ENDDAT
```


Path-related absolute and relative motion

The first motion in a program must be a PTP motion.

```
DEF CPMOTION ( )
INI
  $BASE=BASE_DATA[1]          ;Setting the base coordinate system
  $TOOL=TOOL_DATA[3]          ;Setting the tool coordinate system
  $VEL.CP=1.0                 ;Modification of velocities
  $VEL.ORI1=150
  $VEL.ORI2=100
  $ACC.CP=1.7                 ;Modification of accelerations
  $ACC.ORI1=80
  $ACC.ORI2=80
  ...
  PTP HOME
  LIN {X 1050, Z 900}         ;Absolute coordinate specification
  LIN_REL {X 100, Y 250, Z 125} ;Relative distance specification
  ...
END
```

Circular motion

CIRC *Aux_Point* , *End_Point* , **CA** *Circular_Angle*

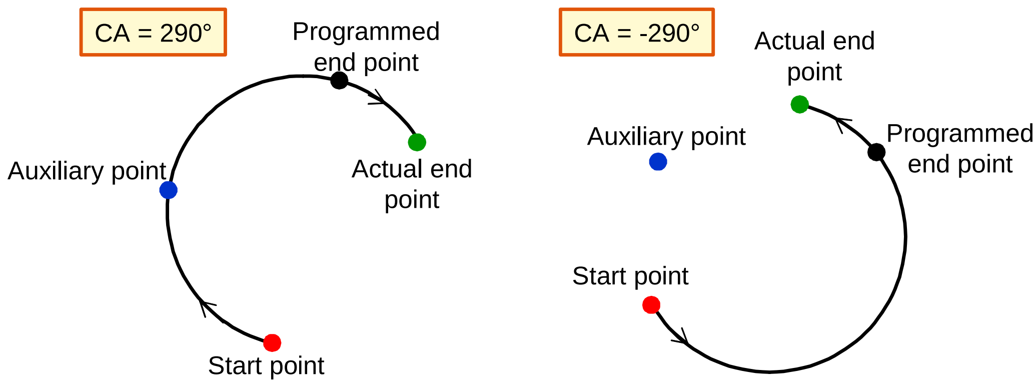
Argument	Explanation
Aux_Point	Auxiliary point coordinate – specified by means of an aggregate or the point name of a point in the data list
End_Point	End point coordinate – specified by means of an aggregate or the point name of a point in the data list
Circular_ Angle	CA = Circular Angle Circular angle in degrees

When specifiying the auxiliary point or end point using an aggregate, make sure the tool and base coordinate systems are assigned correctly!

Circular motion

CA > 0°	Circle is executed in the direction programmed: start point – auxiliary point – end point
CA < 0°	Circle is executed in the opposite direction from that programmed: start point – end point – auxiliary point

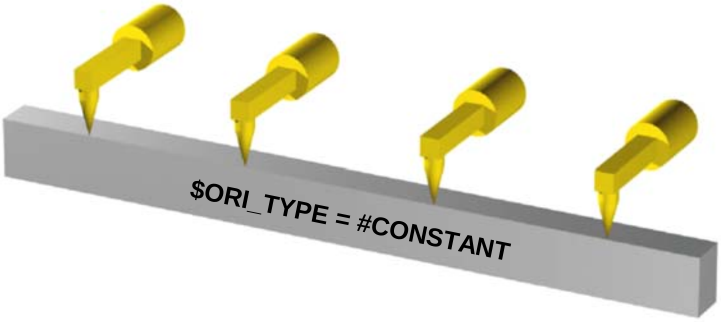
The orientation angles at the “calculated” destination point are identical with the orientation angles at the destination point, that was “touched up”.



7.4. Orientation control

Orientation control with LIN motions

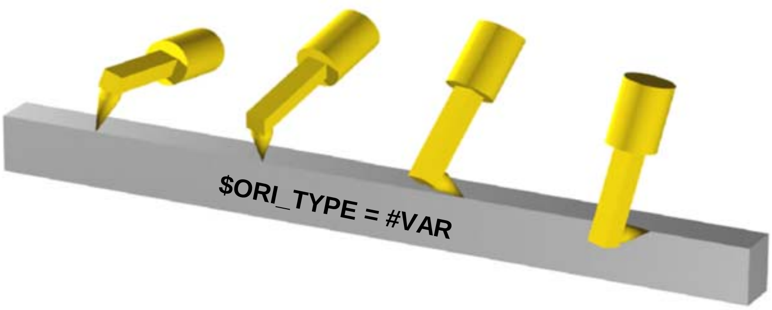
Variable	Value	Description
$ORI_TYPE	*#CONSTANT*	The orientation remains constant during the CP motion. The programmed orientation is disregarded for the end point



 I 20.11.2003 I College I SG I 1

Orientation control with LIN motions

Variable	Value	Description
$ORI_TYPE	*#VAR*	During the CP motion the orientation changes continuously to the orientation of the end point



 I 20.11.2003 I College I SG I 2

Orientation control with LIN motions

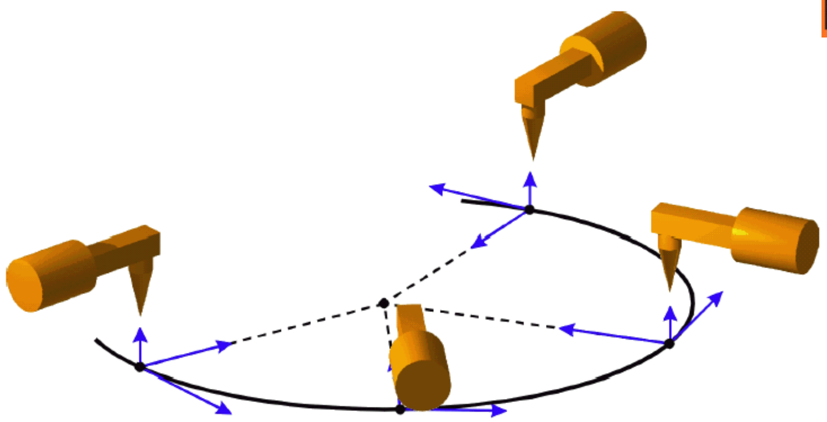
Variable	Value	Description
$ORI_TYPE	*#JOINT*	During the path motion, the orientation of the tool changes continuously from the start position to the end position. This is done by linear transformation of the wrist axis angles. The problem of the wrist singularity can be avoided using this option as there is no orientation control by rotating and pivoting the tool direction.

Variables for orientation control with circular motions

Variable	Value	Description
$ORI_TYPE	*#CONSTANT*	The orientation remains constant during the circular motion. The programmed orientation is disregarded for the end point
$ORI_TYPE	*#VAR*	During the circular motion the orientation changes continuously to the orientation of the end point
$CIRC_TYPE	*#BASE*	Space-related orientation control during the circular motion
$CIRC_TYPE	*#PATH*	Path-related orientation control during the circular motion

The variable “$CIRC_TYPE” **is meaningless** in the case of a linear transformation of the wrist axis angles with “$ORI_TYPE = #JOINT”.

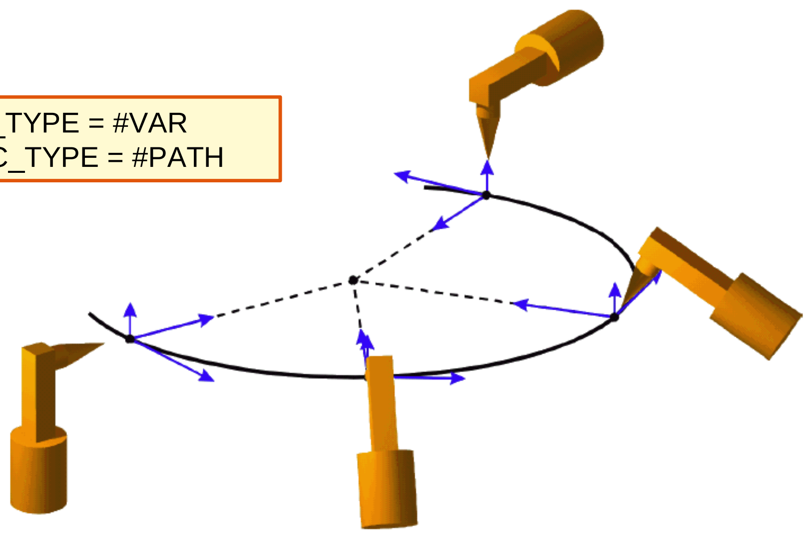
Orientation control with circular motions



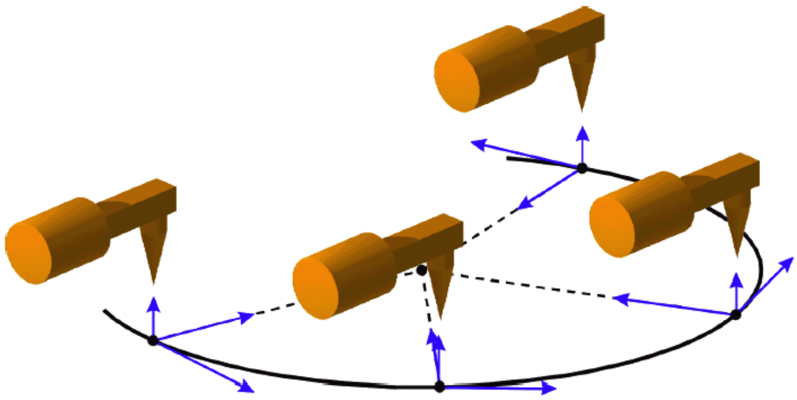
$ORI_TYPE = #CONSTANT
$CIRC_TYPE = #PATH

Orientation control with circular motions

$ORI_TYPE = #VAR
$CIRC_TYPE = #PATH

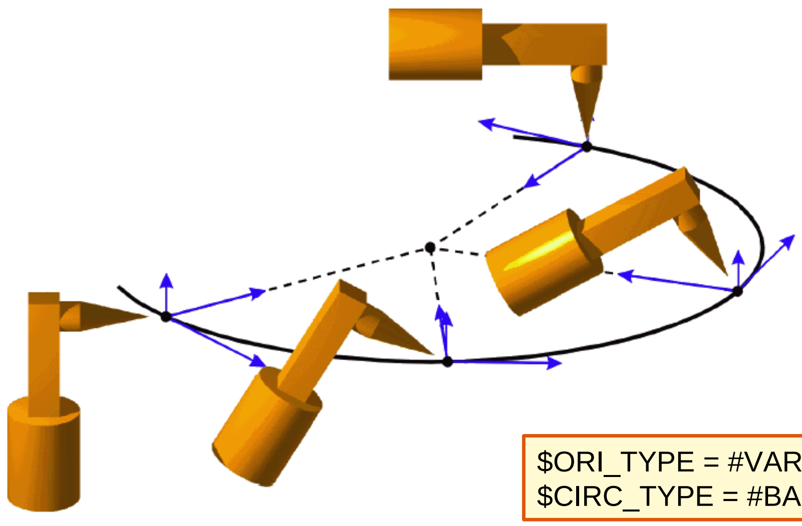


Orientation control with circular motions



$ORI_TYPE = #CONSTANT
$CIRC_TYPE = #BASE

Orientation control with circular motions



$ORI_TYPE = #VAR
$CIRC_TYPE = #BASE

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

7.5. Advance run

Computer advance run

What is the computer advance run?

The **main run pointer** (white bar), which can be seen on the graphical user interface when the program is running, always indicates the block that is currently being processed. The **advance run pointer**, on the other hand, is not visible and **precedes** the **main run pointer** by three motion blocks (default setting).

What is the function of the advance run pointer?

In order to be able to calculate the path, e.g. of an approximation motion, it is necessary to read the path planning data using the advance run pointer. It is not only motion data that are processed, however, but also arithmetical data and commands for controlling the periphery.

How is the advance run pointer influenced?

Instructions and data that influence the periphery (e.g. input/output instructions) trigger an advance run stop. If the advance run pointer is stopped, approximate positioning cannot be carried out.

Advance run pointer

The variable $ADVANCE specifies the maximum number of motion blocks the advance run may process ahead of the program pointer. The actual advance run is dependent, however, on the capacity of the computer.

Syntax:

$ADVANCE = *Value*

Value	Explanation
0	Approximation not possible, every point is positioned exactly
1	Minimum value permitting approximation
3	Default
5	Maximum value

Position of the advance run pointer

```
DEF mov( )
INT X, SPEED
INI

PTP HOME  Vel= 100 % DEFAULT

$ADVANCE=1

LIN P1  Vel= 0.5 m/s CPDAT1 Tool[1]:stift1 Base[0]

SPEED=SPEED*1.2 +0.25

FOR x = 1 to 6
    $VEL_AXIS[x]=75
    $ACC_AXIS[x]=80
ENDFOR

PTP XP2

PTP HOME  Vel= 100 % DEFAULT
END
```

Main run pointer

Advance run pointer

Automatic advance run stop

Instructions and data which affect the peripheral equipment (e.g. I/O instructions), or which are based on the current state of the robot, trigger an advance run stop. This is necessary in order to guarantee the correct sequence of statements and robot motions.

Statements	HALT WAIT ANIN ON/OFF	PULSE ANOUT ON/OFF
Customary system variables	$IN[x] $ANIN[x]	$OUT[x] $ANOUT[x]
Imported variables	All, when accessed	

A complete table of commands that trigger an automatic advance run stop can be found in the documentation (see Documentation CD)

CONTINUE

Syntax: **CONTINUE**

In applications where this advance run stop should be prevented, the command CONTINUE must be programmed immediately before the relevant instruction. The controller then allows the advance run to continue. The effect of this command is limited to the next program line (even if this line is empty!!).

Approximate positioning using CONTINUE

Syntax: **CONTINUE**

The condition or value assignment is polled by the advance run pointer, but it is still possible to approximate the preceding point

```
DEF EXIT_PRO ( )
...
PTP XP9 C_PTP
CONTINUE
WAIT FOR $IN[14] == TRUE
...
END
```

7.6. Approximate positioning

Approximate positioning with PTP commands

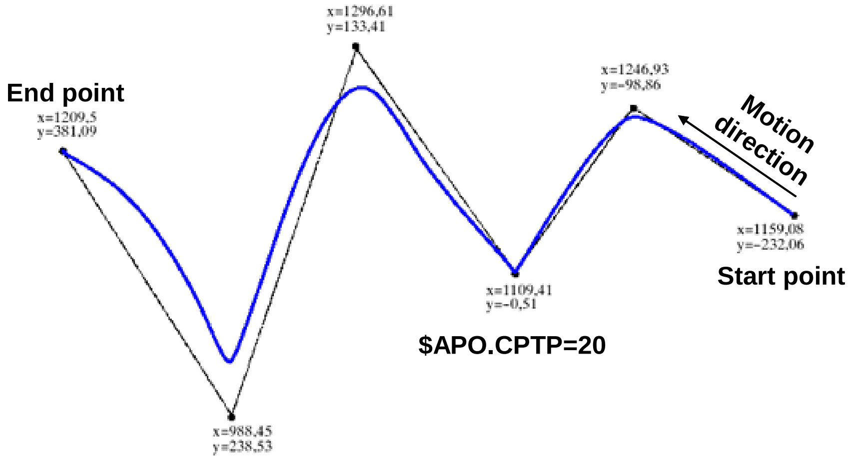
PTP *Point_Name (or coordinate)* **C_PTP**

- The size of the approximate positioning range is set using the variable **$APO.CPTP** (in %)
- The maximum approximation distance is limited in the machine data to 90° for all axes

```
DEF PTP_AXIS_CPTP ( )
...
$APO.CPTP = 50
...
PTP XHOME
PTP {AXIS: A1 80, A2 –40, A3 0, A4 0, A5 0, A6 -90} C_PTP
PTP XPOINT1 C_PTP
...
END
```


Example of PTP-PTP approximate positioning

```
PTP {POS:X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C 162.65,S 2,T 10}
PTP {POS:X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66,S 2,T 10} C_PTP
PTP {POS:X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95,S 2,T 10}
$APO.CPTP=20
PTP {POS:X 1296.61,Y 133.41,Z 715,A 150.32,B 55.07,C 130.23,S 2,T 11} C_PTP
PTP {POS:X 988.45,Y 238.53,Z 715,A 114.65,B 50.46,C 84.62,S 2,T 11} C_PTP
PTP {POS:X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41,S 2,T 11}
```



Approximate positioning with CP motions

Variable	Unit / Data type	Description	Keyword
$APO.CDIS (Distance)	mm REAL	The distance from the end point must be less than the value $APO.CDIS	C_DIS
$APO.CORI (Orientation)	° REAL	The dominant orientation angle must be less than the value $APO.CORI	C_ORI
$APO.CVEL (Velocity)	% INT	The velocity in the deceleration phase to the end point must be less than the value $APO.CVEL	C_VEL

Approximate positioning with CP motions

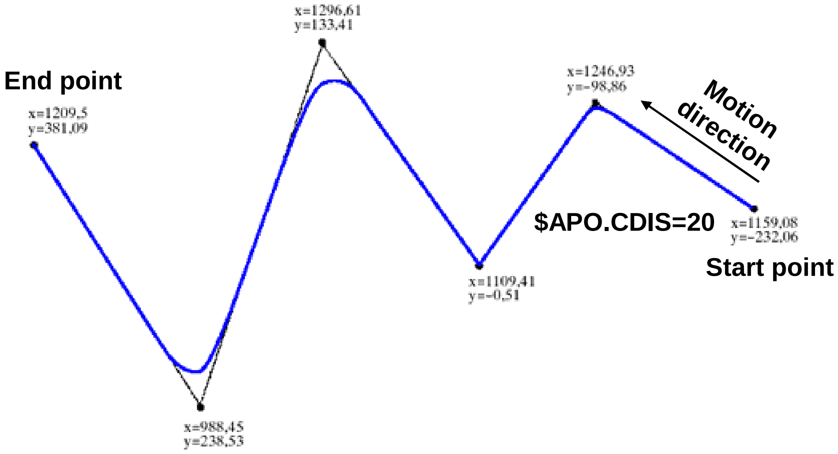
LIN *Point_Name (or coordinate) Approximate_Positioning_Criterion*

The size of the approximate positioning range is set using the variables **$APO.CDIS, $APO.CVEL** and **$APO.CORI**

```
DEF LIN_CIRC_CONT ( )
…
$APO.CVEL = 50
$APO.CDIS = 20
$APO.CORI = 20
…
PTP XHOME
LIN_REL {X -500, Z 250, A 50, C 90} C_VEL
CIRC XHP, XZP, CA 270 C_ORI
LIN {X 1050, Y 120, Z 1000, A 0, B 0, C 0} C_DIS
…
END
```

Example of LIN-LIN approximate positioning

```
PTP {POS: X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C 162.65,S 2,T 10}
$APO.CDIS=20
LIN {X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66} C_DIS
LIN {X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95}
LIN {X 1296.61,Y 133.41,Z 714.99,A 150.32,B 55.07,C 130.23} C_ORI
LIN {X 988.45,Y 238.53,Z 714.99,A 114.65,B 50.46,C 84.62} C_VEL
LIN {X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41}
```

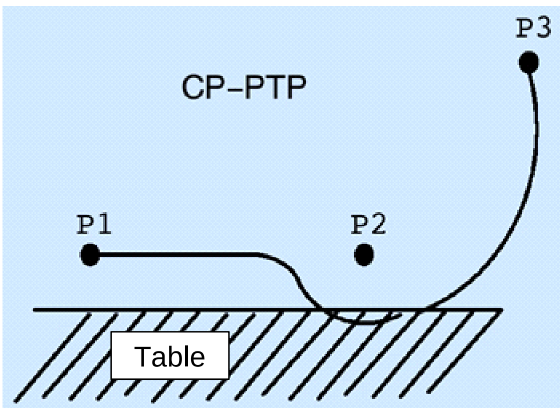


PTP-CP approximate positioning

CP motion (CP = Continuous Path) = LIN or CIRC

Program:

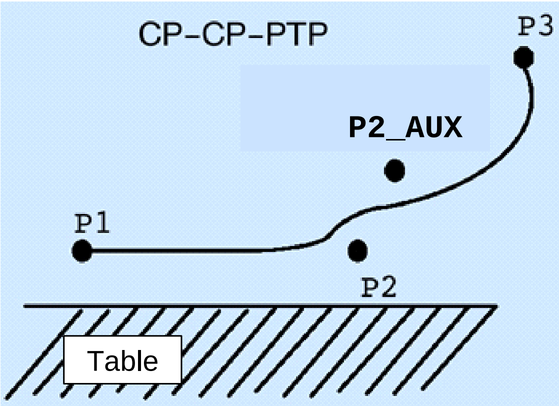
```
...
LIN P1
LIN P2 C_DIS
PTP P3
...
```



The problem here is that the approximate positioning motion is unpredictable. There is the possibility of a collision with the table.

PTP-CP approximate positioning

In order to counter the above problem, while avoiding exact positioning, an additional CP motion (P2_AUX) must be inserted.



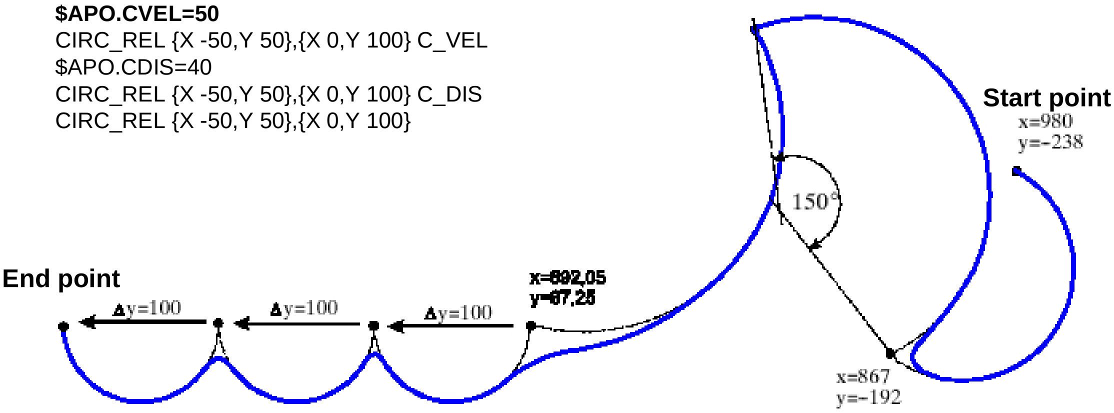
Program:

```
...
LIN P1
LIN P2 C_DIS
LIN P2_AUX C_DIS
PTP P3
...
```

 I 20.11.2003 I College I SG I 7

Example of CIRC-CIRC approximate positioning

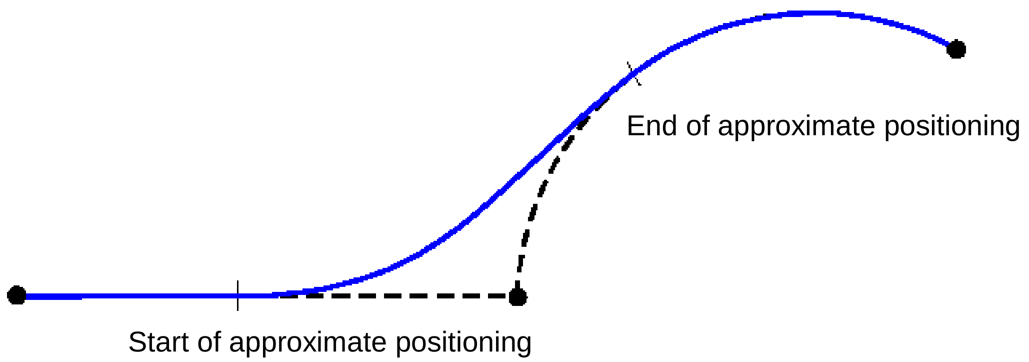
PTP {POS: X 980,Y -238,Z 718,A 133,B 66,C 146,S 6,T 50}
$APO.CDIS=20
CIRC {X 925,Y -285,Z 718},{X 867,Y -192,Z 718,A 155,B 75,C 160} **C_DIS**
$ORI_TYPE=#CONST
CIRC {X 982,Y -221,Z 718,A 50,B 60,C 0},{X 1061,Y -118,Z 718,A -162,B 60,C 177}, CA 150 **C_ORI**
…
CIRC {X 963.08,Y -85.39,Z 718},{X 892.05,Y 67.25,Z 718.01,A 97.34,B 57.07,C 151.11} **C_ORI**
$APO.CVEL=50
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_VEL
$APO.CDIS=40
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_DIS
CIRC_REL {X -50,Y 50},{X 0,Y 100}



 I 20.11.2003 I College I SG I 8

Approximate positioning contour with LIN-CIRC blocks

The approximate positioning path consists of two parabolic segments, which have a tangential transition between each other and also to the individual blocks.



KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

I 20.11.2003 I College I SG I 9

7.7. Status and turn

Status and Turn

The entries “S” and “T” in a POS structure serve to select a specific, unambiguously defined robot position where several different axis positions are possible for the same point in space.

The specification of Status and Turn is only evaluated for PTP motions. For this reason, the first motion in a program must be a PTP motion.

```
DEFDAT MAIN_PROGRAM ( )

DECL POS XPOINT1={X 900, Y 0, Z 800, A 0, B 0, C 0, S 6, T 27}
DECL FDAT FPOINT1...
...

ENDDAT
```

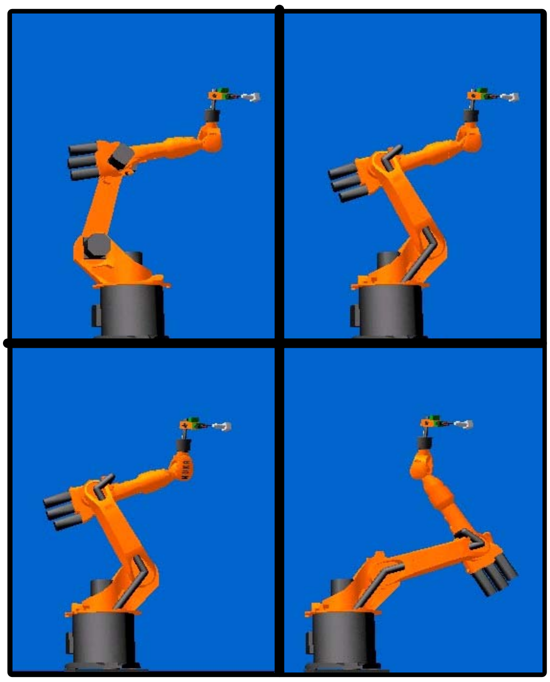
STATUS — S 6

TURN — T 27

Status and Turn

Status = 1
Turn = 46

Status = 2
Turn = 43



Status = 6
Turn = 59

Status = 4
Turn = 63

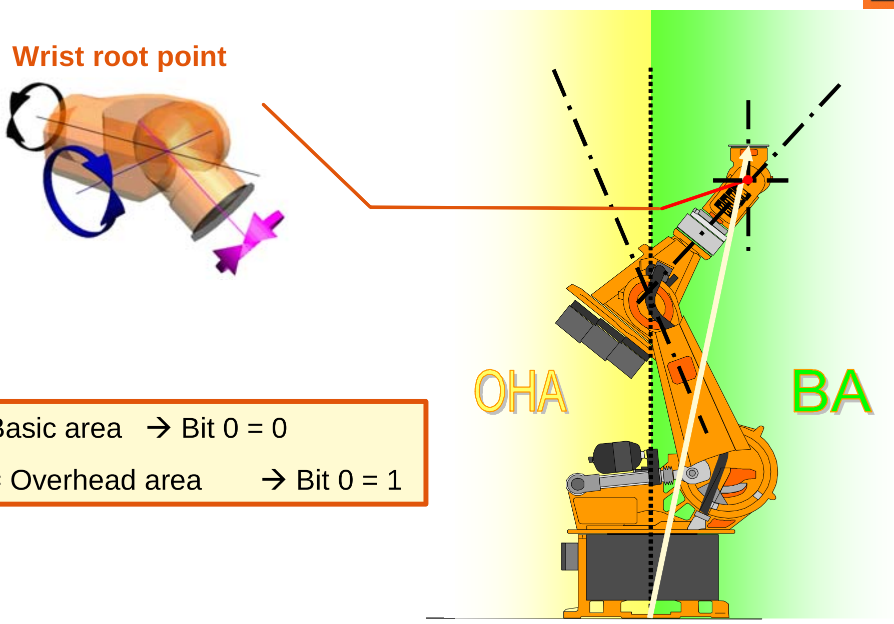
Turn

With rotational axes, the individual bits determine the sign before the axis value in the following way:
Bit x = 0: angle of axis x ≥ 0°
Bit x = 1: angle of axis x < 0°

Value	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A6 ≥ 0°	A5 ≥ 0°	A4 ≥ 0°	A3 ≥ 0°	A2 ≥ 0°	A1 ≥ 0°
1	A6 < 0°	A5 < 0°	A4 < 0°	A3 < 0°	A2 < 0°	A1 < 0°

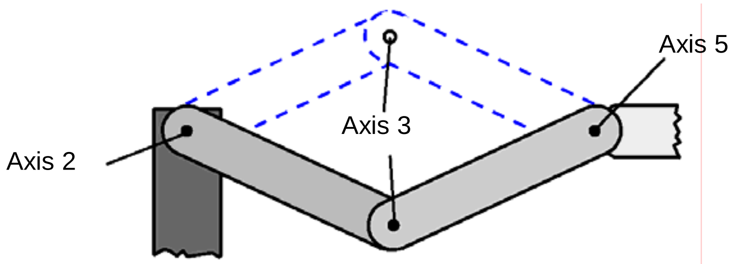
Example:
Specification T 11 → T'B010011'
Axes A1, A2 and A5 negative; axes A3, A4 and A6 positive

Status – BIT 0: Position of the wrist root point

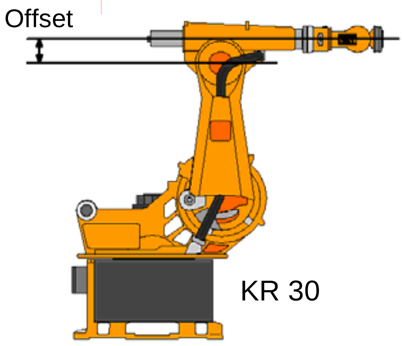


BA = Basic area → Bit 0 = 0
OHA = Overhead area → Bit 0 = 1

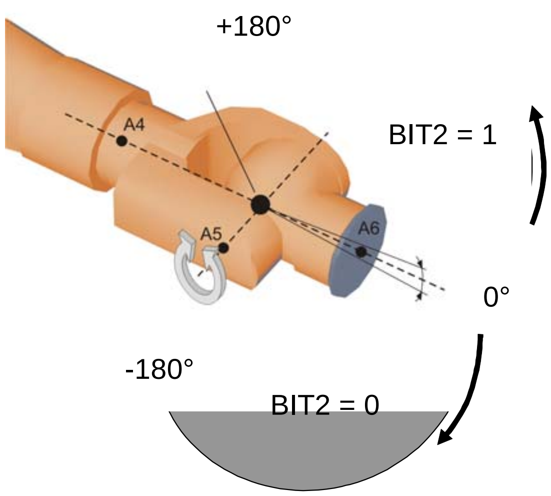
Status – BIT 1: Arm configuration



Bit 1 specifies the position of the arm. The setting of the bit is dependent on the robot model in use. For robots whose axes 3 and 4 intersect, the following applies:
Bit 1 has the value 0, if axis 3 < 0°, otherwise bit 1 = 1.
For robots with an offset between axis 3 and axis 4 (e.g. KR 30), the angle at which the value of bit 1 changes depends on the size of this offset.



Status – BIT 2: Wrist configuration



Bit 2 specifies the configuration of the wrist. This is dependent on the position of axis 5.

8. System variables

System variables and system files

- The KRC supports the concept of **predefined system variables** and **system files**.
- System variables can be read and written in the same way as any other variable (with limitations due to the type of data).
- It is possible to write separate data lists with data definitions and initializations if requried.
- The system contains predefined data lists which cannot be generated or deleted, e.g.:

$MACHINE.DAT	➡	Adapt controller to robot
$CUSTOM.DAT	➡	Configure control functions
$CONFIG.DAT	➡	Predefined data list
$ROBCOR.DAT	➡	Robot-specific data

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I **20.11.2003 I College I SG I** 1

Starting and stopping timers

The system variables $TIMER[1]…$TIMER[32] serve the purpose of measuring time sequences (20 on the display). A timing process is started and stopped by means of the system variables $TIMER_STOP[1]…$TIMER_STOP[16]:

Syntax:

$TIMER_STOP[No] = *Value*

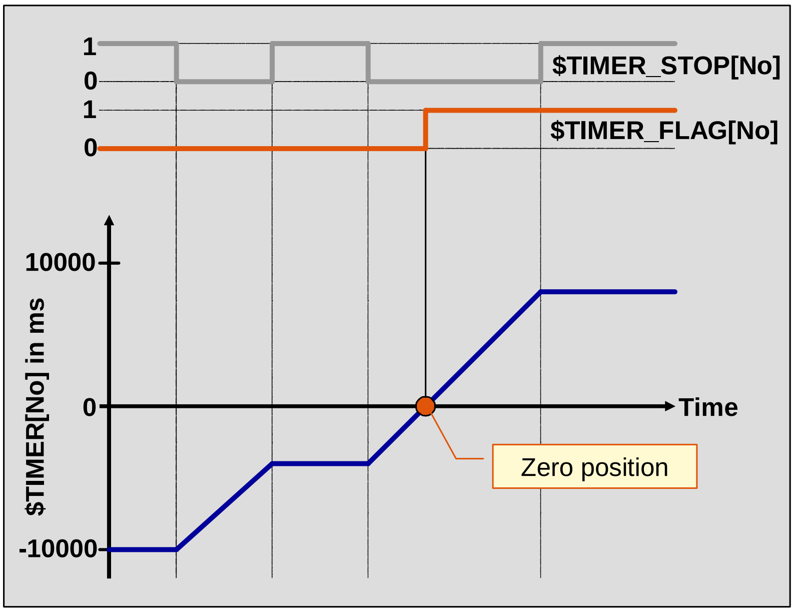
Argument	Type	Explanation
Value	BOOL	FALSE: Start timer TRUE: Stop timer

Example:

```
$TIMER[5] = 0              ;Reset timer 5 by assigning a value
$TIMER_STOP[5] = FALSE     ;Start timer 5
...
$TIMER_STOP[5] = TRUE      ;Stop timer 5
```

KUKA Roboter GmbH, Hery-Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I **20.11.2003 I College I SG I** 2

Timer flag



 I 20.11.2003 I College I SG I 3

Flags

Flags are 1-bit memories and are used as global markers. Flags are static.

Syntax:

$FLAG[*No*] = *Value*

Argument	Type	Explanation
No.	INT	Flag number, range of values: 1…1024 (999 on the display)
Value	BOOL	FALSE: Reset flag TRUE: Set flag

Example:

```
$FLAG[1] = FALSE                ;Reset flag 1
```

 I 20.11.2003 I College I SG I 4

Cyclical flags

Cyclical flags are also 1-bit memories, but are cyclically active at the robot level only.

Syntax:

$CYCFLAG[*No*] = *Value*

Argument	Type	Explanation
No.	INT	Cyclical flag number, range of values: 1…256 (32 on the display)
Value	BOOL	FALSE: Reset cyclical flag TRUE: Set cyclical flag

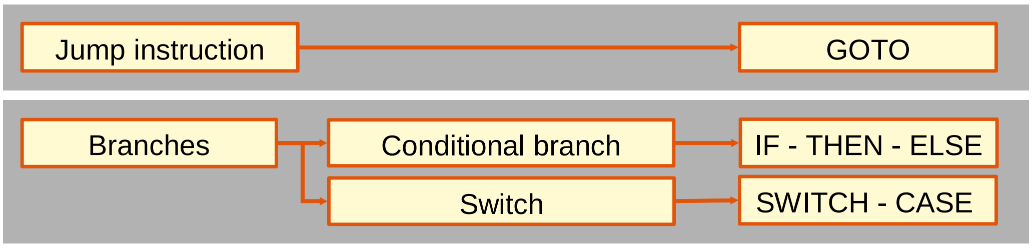
Example:

```
$CYCFLAG[1] = $IN[2] AND $IN[13]     ;Boolean expression is cyclically
                                      evaluated and updated.
```


9. Program execution control

9.1. Jump instruction and branches

Program execution control - Overview



The controller does not detect logic errors!

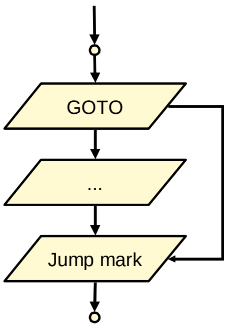
Jump instruction

<u>Syntax:</u>

GOTO *Marker*

GOTO statements can very quickly lead to a loss of structural clarity within a program and should therefore be used as little as possible

```
DEF PRG ( )

...
GOTO Mark1
...
...
Mark1:
...
END
```

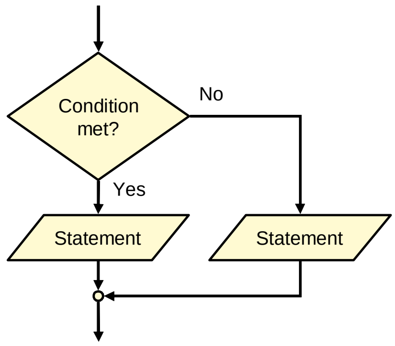


Conditional branch

Syntax:

```
IF  Execution_Condition  THEN
        Statement
ELSE
        Statement
ENDIF
```

```
...
IF $IN[22]==TRUE THEN
      PTP HOME
ELSE
      $OUT[17]=TRUE
      $OUT[18]=FALSE
      PTP HOME
ENDIF
...
```

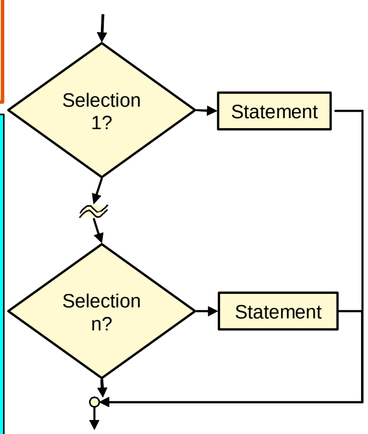


Verteiler

Syntax:

```
SWITCH Variable (INT or ENUM)
    CASE 1
        Statement
    CASE 2
        Statement
    DEFAULT
ENDSWITCH
```

```
...
SWITCH PROG_NO
  CASE 1
      Part1 ( )      ; if Prog_No = 1
  CASE 2
      Part2 ()       ; if Prog_No = 2
  DEFAULT
      ERROR_SP( ) ; all other values
ENDSWITCH
...
```



9.2. Loops

Program execution control - Overview

- Loops
  - Endless loop → LOOP
  - Counting loop → FOR - TO - STEP
  - Conditional loops → WHILE, REPEAT - UNTIL

The controller does not detect logic errors!

Loops - general

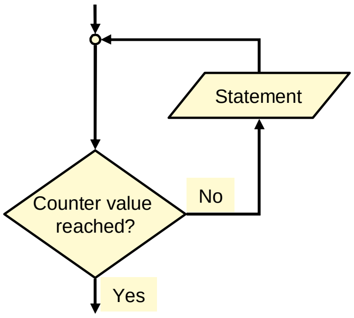
1. Loops are required for repeating program instructions.
2. A distinction is made between counting loops and conditional loops.
3. A jump into the loop from outside is not allowed and is refused by the controller (error messages).
4. The nesting of loops is a further programming technique.

Counting loop

Syntax:

FOR *Counter* **=** *Start* **TO** *End* **STEP** *Increment*
Statement
ENDFOR

```
E6POS POSITION[4,4]
INT X, Y
...
FOR X=1 TO 4 STEP 1
   FOR Y=4 TO 1 STEP -1
         POSITION[X,Y]=P1
         POSITION[X,Y].X=P1.X+50
         POSITION[X,Y].Y=P1.Y+50
   ENDFOR
ENDFOR
```



I 20.11.2003 I College I SG I 3

Rejecting loop

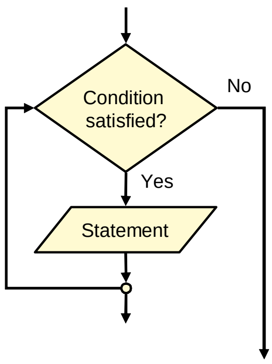
Syntax:

WHILE *Condition*

Statement

ENDWHILE

```
...
WHILE $IN[22]==TRUE

      $OUT[17]=TRUE
      $OUT[18]=FALSE
      PTP HOME

ENDWHILE
...
```



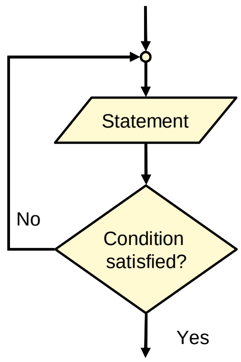
I 20.11.2003 I College I SG I 4

Non-rejecting loop

Syntax:

```
REPEAT
    Statement
UNTIL Condition
```

```
...
REPEAT
    $OUT[17]=TRUE
    $OUT[18]=FALSE
    PTP HOME
UNTIL $IN[22]==TRUE
...
```

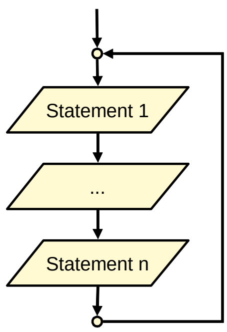


Endless loop

Syntax:

```
LOOP
    Statement 1
    ...
    Statement n
ENDLOOP
```

```
...
PTP HOME
LOOP
    LIN XP1
    LIN XP2
    LIN XP4
ENDLOOP
PTP HOME
...
```



Premature termination of loop execution

Syntax:

EXIT

Any loop can be terminated prematurely by using the EXIT statement.

```
DEF EXIT_PRO ( )
   PTP HOME Vel=100% DEFAULT
    LOOP                              ;Start of endless loop
     LIN P1 Vel=0.4 m/s CPDAT
       IF $IN[1] == TRUE THEN          ;Terminate when
        EXIT                           ;input 1 set
       ENDIF
    LIN P2 Vel=0.4 m/s CPDAT
   ENDLOOP                      ; End of endless loop
   PTP HOME Vel=100% DEFAULT
END
```

9.3. Wait and Halt function

Wait statements

Syntax: **WAIT FOR** *Condition*

```
DEF EXIT_PRO ( )
...
WAIT FOR $IN[14] == TRUE  ;waits until input 14 is set
...
END
```

- If the logical expression “Condition” is already TRUE when WAIT is called, program execution is not stopped (**an advance run stop is triggered, however**).
- If “Condition” is FALSE, program execution is stopped until the expression takes the value TRUE.

Wait statements

Syntax: **WAIT FOR** *Time*

```
DEF EXIT_PRO ( )
...
WAIT SEC 5                 ;waits 5 seconds
...
WAIT SEC Time*0.5+0.25     ;waits calculated time

END
```

“Time” is an arithmetic REAL expression which can be used to specify the number of seconds for which program execution is to be interrupted. If the value is negative, the program does not wait.

Stopping the program

Syntax: **HALT**

HALT statements are used to stop programs. The last motion instruction to be executed will, however, be completed.

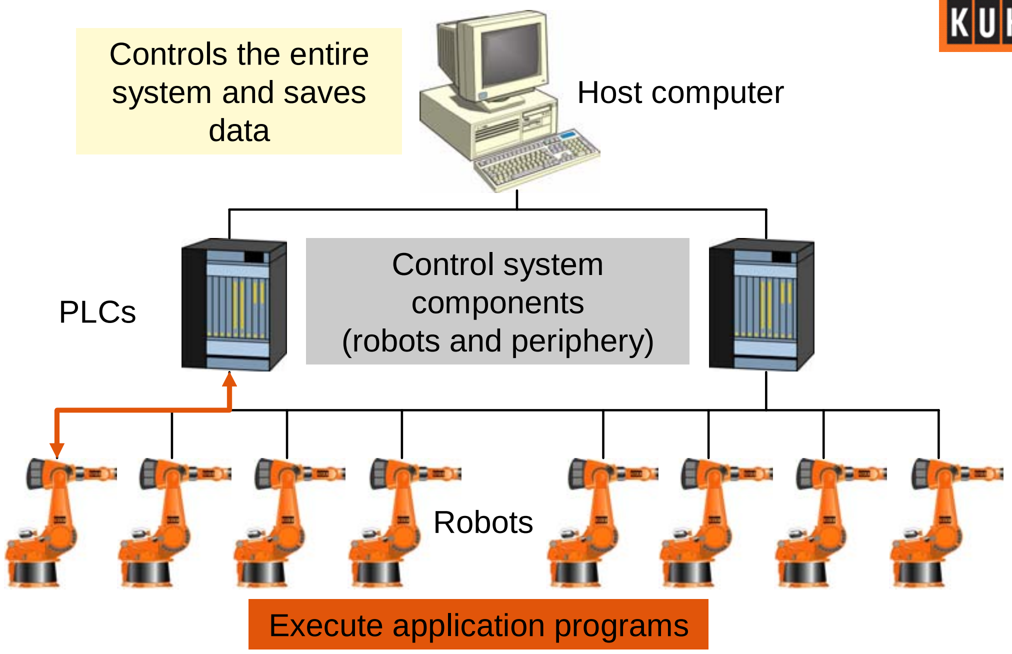
The program can be resumed by pressing the START key.

Special case: In the case of interrupt programs, the routine is processed completely (advance run pointer).

10. Automatik External function

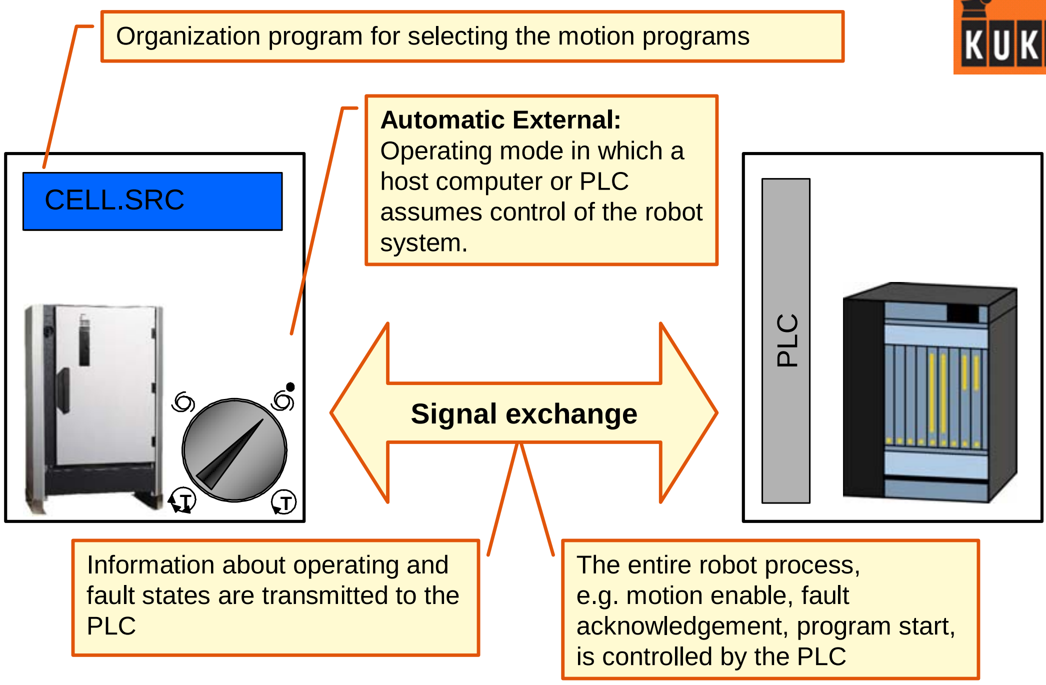
10.1. Definition and sequence

System structure



 I 08.03.2005 I College I AC I 1

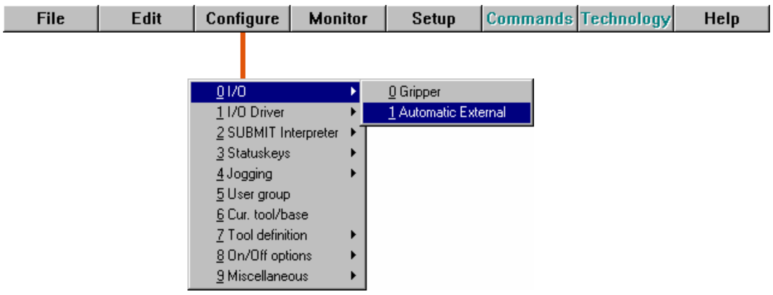
System structure: KRC - PLC



 I 08.03.2005 I College I AC I 2

Configuring the interface

The signals of the Automatic External interface must be assigned physical inputs and outputs of the robot controller.



Automatic External inputs

Functional description

Variable name

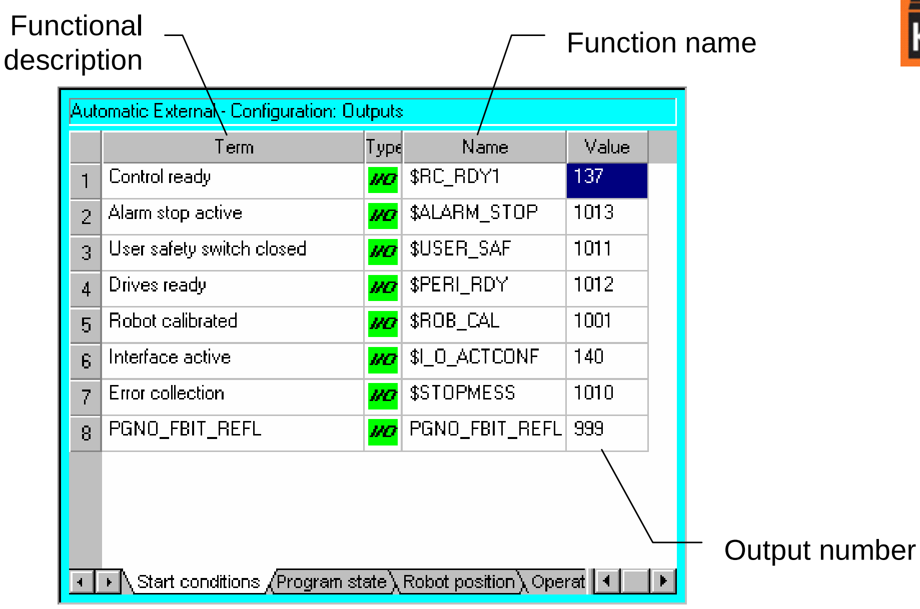
Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	1026
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	1026
10	Drives off (invers)	I/O	$DRIVES_OFF	1025
11	Drives on	I/O	$DRIVES_ON	140
12	Activate interface	I/O	$I_O_ACT	1025

Var Variable
I/O Input

Input number or variable value

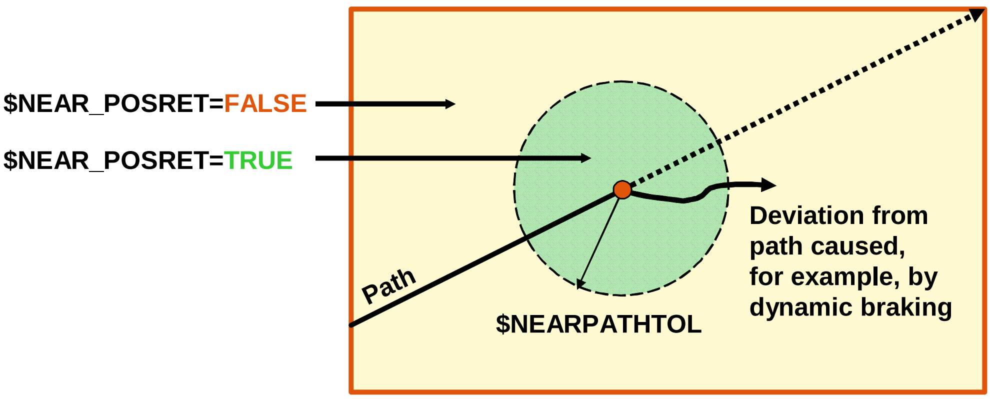
Automatic External outputs - Start condition



	Term	Type	Name	Value
1	Control ready	I/O	$RC_RDY1	137
2	Alarm stop active	I/O	$ALARM_STOP	1013
3	User safety switch closed	I/O	$USER_SAF	1011
4	Drives ready	I/O	$PERI_RDY	1012
5	Robot calibrated	I/O	$ROB_CAL	1001
6	Interface active	I/O	$I_O_ACTCONF	140
7	Error collection	I/O	$STOPMESS	1010
8	PGNO_FBIT_REFL	I/O	PGNO_FBIT_REFL	999

$NEAR_POSRET

The signal **$NEAR_POSRET** can be used to determine whether or not the robot is situated within a sphere about the position saved in $POS_RET*). The radius of the sphere can be set in the file $CUSTOM.DAT using the system variable $NEARPATHTOL.



***) $POS_RET:** Position at which the robot left the path (●)

Organization program: CELL.SRC

Position check

Function for program number transfer

- Delete semicolon
- Substitute program name for EXAMPLEx

If program number invalid, restart program with $EXT_START

```
...
INIT
BASISTECH INI
CHECK HOME
PTP HOME Vel= 100 % DEFAULT
AUTOEXT INI
 LOOP
   P00 (#EXT_PGNO,#PGNO_GET,DMY[],0 )

     SWITCH PGNO

        CASE 1
        P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
        CASE 2
        P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
        CASE 3
        P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )

        DEFAULT
        P00 (#EXT_PGNO,#PGNO_FAULT,DMY[],0 )

      ENDSWITCH
  ENDLOOP
```

 I 08.03.2005 I College I AC I 7

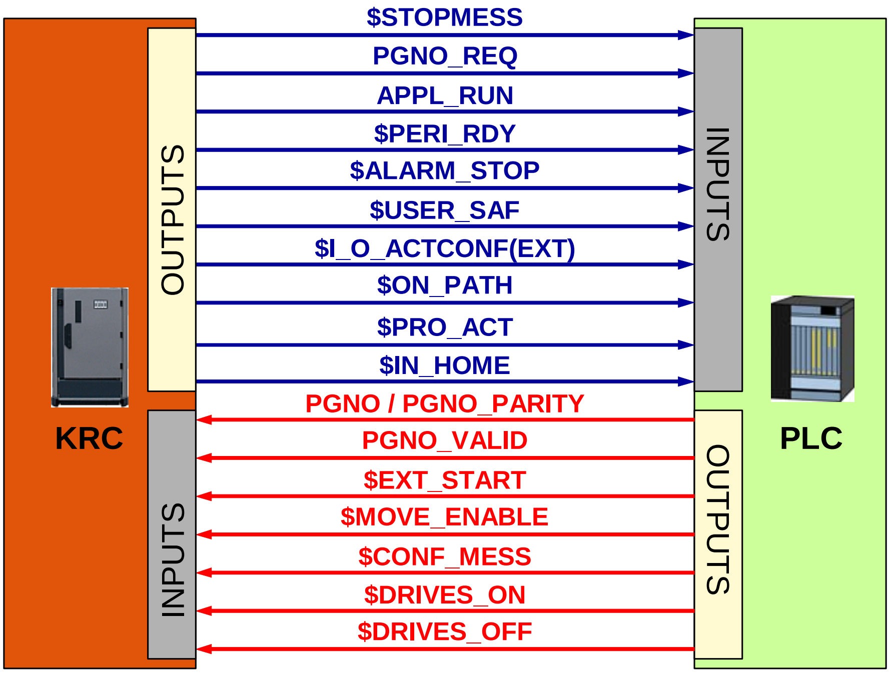
Expanding CELL.SRC

- Select CASE branch with SHIFT + cursor key and insert as new copied branch
- Proceed in the same way for each subsequent program
- Modify program number and corresponding program name

```
LOOP

   P00 (#EXT_PGNO,#PGNO_GET,DMY[],0 )
   SWITCH PGNO

     CASE 2
     P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
     PROG2 ( )
     CASE 3
     P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
     PROG3 ( )

     DEFAULT
     P00 (#EXT_PGNO,#PGNO_FAULT,DMY[],0 )
   ENDSWITCH

ENDLOOP
```

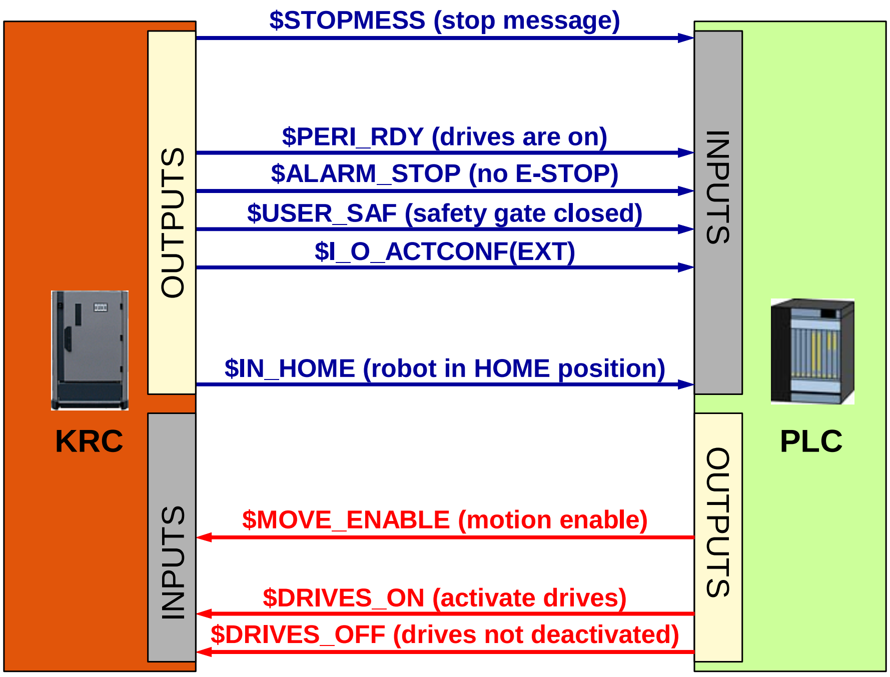
 I 08.03.2005 I College I AC I 8

"Automatic External" interface example (not complete)



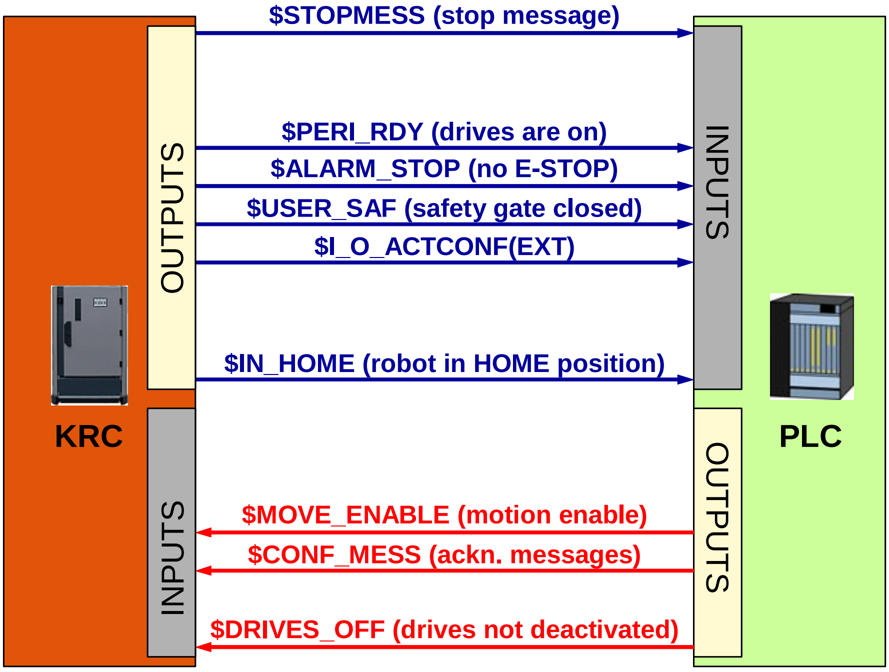
KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 08.03.2005 I College I AC I 9

Signal diagram example: Precondition



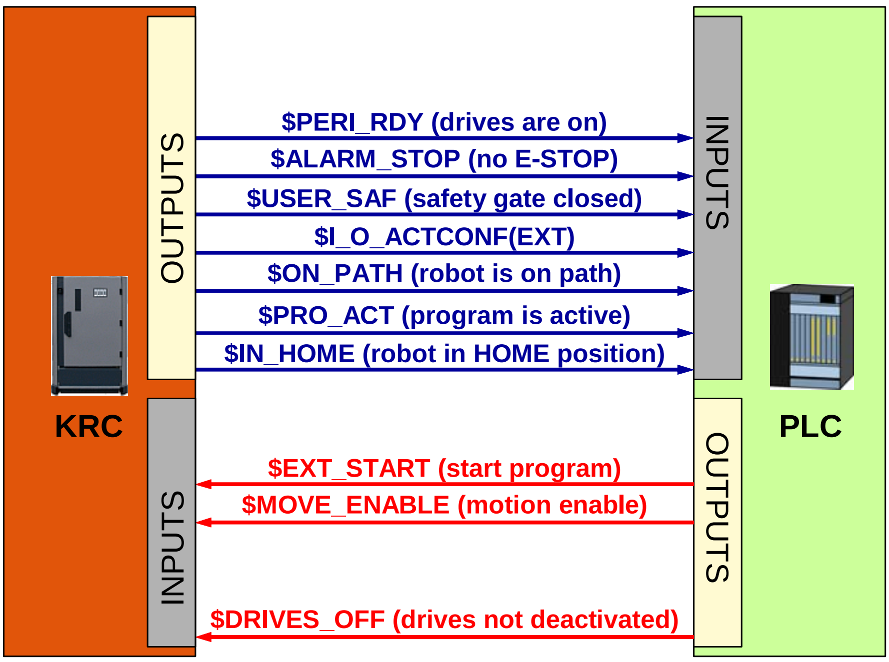
KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 08.03.2005 I College I AC I 10

Signal diagram example: Acknowledge messages



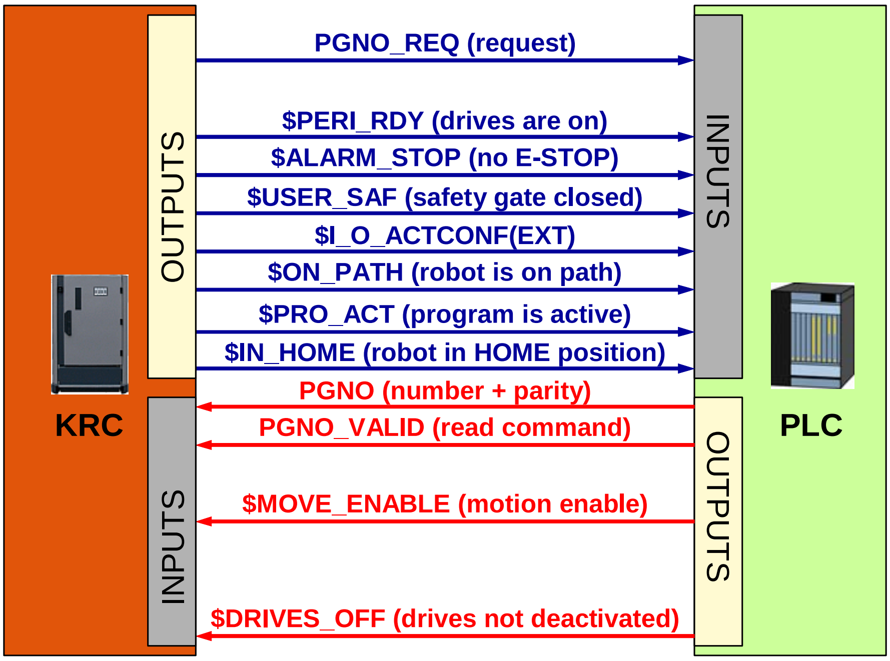
I 08.03.2005 I College I AC I 12

Signal diagram example: Start program



I 08.03.2005 I College I AC I 14

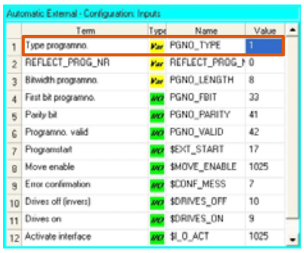
Signal diagram example: Transfer program number



KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 08.03.2005 I College I AC I 16

10.2. Configuration inputs

Program number format: PGNO_TYPE


Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

PGNO_TYPE – Program number type

This value determines the format in which the program number sent by the host computer is read.

PGNO_TYPE	Read as ...	Meaning	Example
1	Binary number	Sent as binary coded **integer**	Click
2	BCD value	Sent as binary coded **decimal**	Click
3	“1 of n” *1	Sent as “**1 of n**” coded value	Click
*1	When using this transmission format, the values of PGNO_REQ, PGNO_PARITY and PGNO_VALID are not evaluated and are thus of no significance.		

Display format of the program number: BINARY

Binary coded integer: **PGNO_TYPE=1**

Input:	...	E8	E7	E6	E5	E4	E3	E2	E1
Values:		2^7= 128	2^6= 64	2^5= 32	2^4= 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1
Example:		0	1	1	0	0	0	0	1

Prog. no.: 97 =0 + 64 + 32 + 0 + 0 + 0 + 0 + 1

Display format of the program number: BCD

Binary coded decimal: **PGNO_TYPE=2**

		2nd decimal place				1st decimal place		
Input: ...	E8	E7	E6	E5	E4	E3	E2	E1
Values:	8	4	2	1	8	4	2	1
Example:	0	0	1	0	0	1	1	1
Result:	2				7			
Prog. no.:	27							

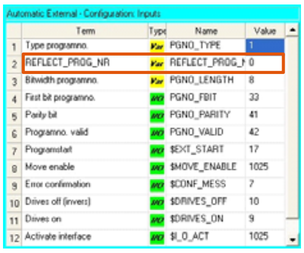
Display format of the program number: "1 of n"

"1 of n" coded integer: **PGNO_TYPE=3**

Input (max. 16): ...	E8	E7	E6	E5	E4	E3	E2	E1
Prog. no. 1 :	0	0	0	0	0	0	0	**1**
Prog. no. 2 :	0	0	0	0	0	0	**1**	0
Prog. no. 3 :	0	0	0	0	0	**1**	0	0
Prog. no. 4 :	0	0	0	0	**1**	0	0	0
Prog. no. 5 :	0	0	0	**1**	0	0	0	0
Prog. no. 6 :	0	0	**1**	0	0	0	0	0
Prog. no. 7 :	0	**1**	0	0	0	0	0	0
Prog. no. 8 :	**1**	0	0	0	0	0	0	0

Only one input may be "TRUE" at any time.
All other input combinations result in an invalid program number.

Mirroring the program number: REFLECT_PROG_NR



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

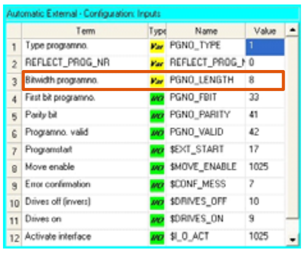
REFLECT_PROG_NR – REFLECT_PROG_NR

This option allows you to decide whether or not the program number should be mirrored in a definable output area.

REFLECT_PROG_NR	Function
0	Deactivated
1	Activated

The output of the signal starts with the output defined using “PGNO_FBIT_REFL”.

Program number length in bits: PGNO_LENGTH



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

PGNO_LENGTH – Program number length in bits

Its value determines the number of bits defining the program number sent by the host computer.

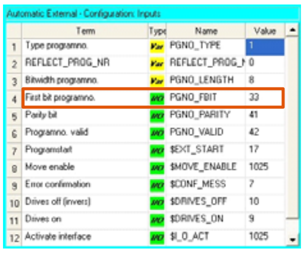
<u>Range of values:</u> PGNO_LENGTH = 1...16

<u>Example:</u> PGNO_LENGTH = **4**

the external program number is **four bits** long

While PGNO_TYPE has the value 2 (program number read as BCD value), only 4, 8, 12 and 16 are permissible values for the number of bits.

First bit in the program number: PGNO_FBIT



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

PGNO_FBIT – First bit in program number

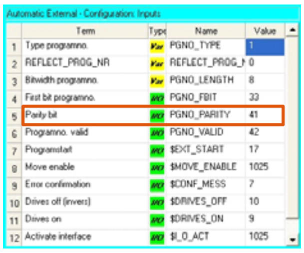
Input representing the first bit of the program number.

Range of values: PGNO_FBIT = 1...4096

Example: PGNO_FBIT = **5**

the external program number begins with the **input $IN[5]**

Parity bit: PGNO_PARITY



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

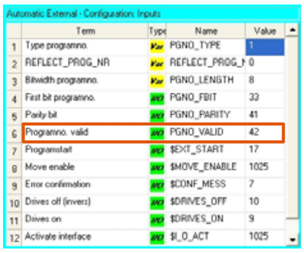
PGNO_PARITY – Parity bit

Input to which the parity bit is transferred from the host computer.

Input	Function
Negative value	Odd parity
0	No evaluation
Positive value	Even parity

While PGNO_TYPE has the value 3 (program number read as “1 of n” value), PGNO_PARITY is NOT evaluated.

Program number valid: PGNO_VALID



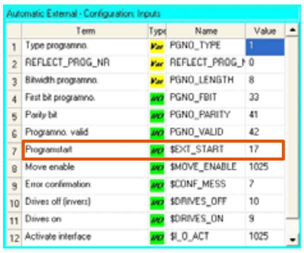
PGNO_VALID – Program number valid

Input to which the command to read the program number is transferred from the host computer. If the variable PGNO_TYPE has the value 3, PGNO_VALID is not evaluated.

Input	Function
Negative value	Number is transferred at the falling edge of the signal
0	Number is transferred at the rising edge of the signal on the EXT_START line
Positive value	Number is transferred at the rising edge of the signal

KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 08.03.2005 I College I AC I 6

Program start: $EXT_START



$EXT_START – Program start

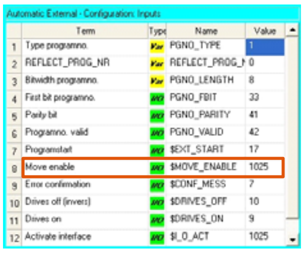
If the I/O interface is active, this input can be set to start or continue a program.

Only the rising edge of the signal is evaluated.

There is no BCO run in Automatic External mode, so there is no program stop at the first programmed position.

KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de I 08.03.2005 I College I AC I 7

Motion enable: $MOVE_ENABLE



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

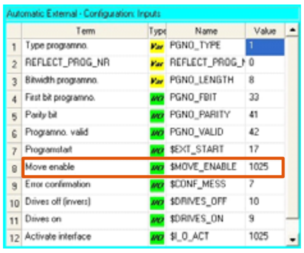
$MOVE_ENABLE – Motion enable

This input is used by the host computer to check the robot drives.

Signal	Function
TRUE	Jogging and program execution are possible
FALSE	All drives are stopped and all active commands inhibited

If the drives have been switched off by the host computer, the message "GENERAL MOTION ENABLE" appears in the message window of the KCP. It is only possible to move the robot again once this message has been reset and another external start signal has been given.

MOVE_ENABLE monitoring: $CHCK_MOVENA



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

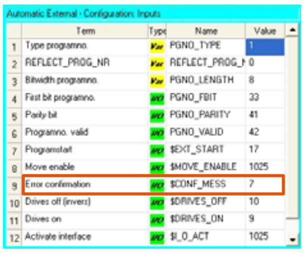
$MOVE_ENABLE – Motion enable

If the variable $CHCK_MOVENA has the value "FALSE", MOVE_ENABLE can be bypassed. The value of the variable can only be changed in the file **"C:\KRC\Roboter\KRC\Steu\MaDa\OPTION.DAT"**.

Signal	Function
TRUE	MOVE_ENABLE monitoring is activated
FALSE	MOVE_ENABLE monitoring is deactivated

In order to be able to use MOVE_ENABLE monitoring, $MOVE_ENABLE must have been configured with the input "$IN[1025]".
Otherwise, "$CHCK_MOVENA" has no effect whatsoever.

Error acknowledgement: $CONF_MESS



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

$CONF_MESS – Error acknowledgement

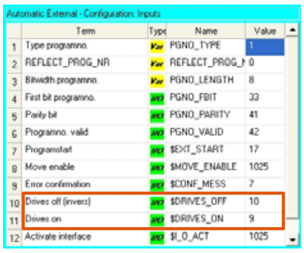
Setting this input enables the host computer to reset (acknowledge) error messages automatically.

Only the rising edge of the signal is evaluated.

Acknowledgement of the error messages is only possible once the cause of the error has been eliminated.

KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de

Drives on/off: $DRIVES_ON / $DRIVES_OFF



Automatic External - Configuration: Inputs

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

DRIVES_OFF (inverse)

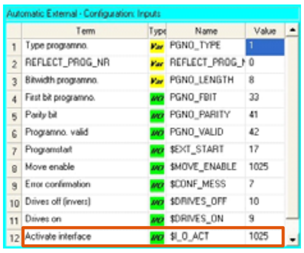
With a low-level pulse of at least 20 ms duration at this input, the host computer can switch off the robot drives.

DRIVES_ON

With a high-level pulse of at least 20 ms duration at this input, the host computer can switch on the robot drives.

KUKA Roboter GmbH, Hery Park 3000, D-86368 Gersthofen, Tel.: +49 (0) 8 21/45 33-1906, Fax: +49 (0) 8 21/45 33-2340, http://www.kuka-roboter.de
© Copyright by KUKA Roboter GmbH College

Drives on/off: $DRIVES_ON / $DRIVES_OFF


Automatic External - Configuration: Inputs

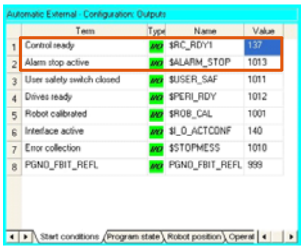
	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	REFLECT_PROG_NR	Var	REFLECT_PROG_N	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	17
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	7
10	Drives off (invers)	I/O	$DRIVES_OFF	10
11	Drives on	I/O	$DRIVES_ON	9
12	Activate interface	I/O	$I_O_ACT	1025

$I_O_ACT – Interface

The I/O interface can be activated by setting this system variable to TRUE ($IN[1025])

10.3. Configuration outputs

Output RC_RDY1 & ALARM_STOP



Automatic External - Configuration: Outputs

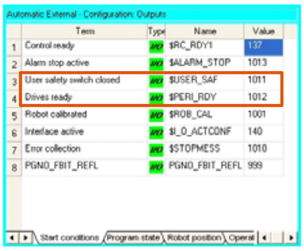
	Term	Type	Name	Value
1	Control ready		$RC_RDY1	137
2	Alarm stop active		$ALARM_STOP	1013
3	User safety switch closed		$USER_SAF	1011
4	Drives ready		$PERI_RDY	1012
5	Robot calibrated		$ROB_CAL	1001
6	Interface active		$I_O_ACTCONF	140
7	Error collection		$STOPMESS	1010
8	PGNO_FBIT_REFL		PGNO_FBIT_REFL	999

Start condition

RC_RDY1 – Controller ready
Ready for program start

ALARM_STOP – E-Stop circuit closed
This output is reset in the event of an EMERGENCY STOP event.

Output USER_SAF & PERI_RDY



Automatic External - Configuration: Outputs

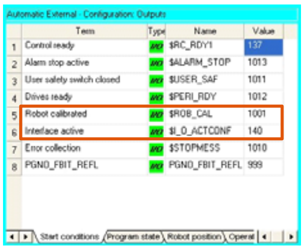
	Term	Type	Name	Value
1	Control ready		$RC_RDY1	137
2	Alarm stop active		$ALARM_STOP	1013
3	User safety switch closed		$USER_SAF	1011
4	Drives ready		$PERI_RDY	1012
5	Robot calibrated		$ROB_CAL	1001
6	Interface active		$I_O_ACTCONF	140
7	Error collection		$STOPMESS	1010
8	PGNO_FBIT_REFL		PGNO_FBIT_REFL	999

Start condition

USER_SAF – Operator safety device closed
This output is reset if the safety fence monitoring switch is opened (in AUTO mode) or an enabling switch is released (in TEST mode).

PERI_RDY – Drives ready
By setting this output, the robot controller communicates to the host computer the fact that the robot drives are switched on.

Output ROB_CAL & I_O_ACTCONF



Automatic External - Configuration: Outputs

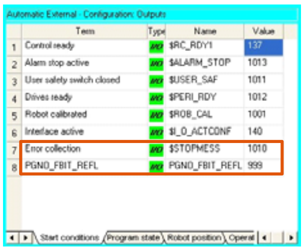
	Term	Type	Name	Value
1	Control ready		$RC_RDY1	137
2	Alarm stop active		$ALARM_STOP	1013
3	User safety switch closed		$USER_SAF	1011
4	Drives ready		$PERI_RDY	1012
5	Robot calibrated		$ROB_CAL	1001
6	Interface active		$I_O_ACTCONF	140
7	Error collection		$STOPMESS	1010
8	PGNO_FBIT_REFL		PGNO_FBIT_REFL	999

Start condition

ROB_CAL – Robot mastered
Signal is FALSE as soon as an axis has been unmastered

I_O_ACTCONF – Interface active
The Automatic External interface is active. This signal is set to TRUE as soon as the operating mode is switched to EXT.

Output STOPMESS & PGNO_FBIT_REFL



Automatic External - Configuration: Outputs

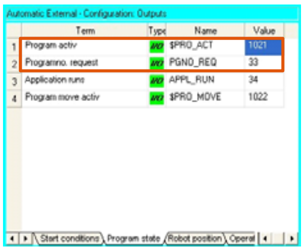
	Term	Type	Name	Value
1	Control ready		$RC_RDY1	137
2	Alarm stop active		$ALARM_STOP	1013
3	User safety switch closed		$USER_SAF	1011
4	Drives ready		$PERI_RDY	1012
5	Robot calibrated		$ROB_CAL	1001
6	Interface active		$I_O_ACTCONF	140
7	Error collection		$STOPMESS	1010
8	PGNO_FBIT_REFL		PGNO_FBIT_REFL	999

Start condition

STOPMESS – Collective fault
This output is set by the robot controller in order to communicate to the host computer any message occurring which requires the robot to be stopped. (e.g. EMERGENCY STOP, Motion enable, Operator safety, Command velocity, etc.)

PGNO_FBIT_REFL
Mirrored output representing the first bit of the program number. If a program selected by the PLC is deselected by the user, the output area starting with PGNO_FBIT_REFL is set to "FALSE". In this way, the PLC can prevent a program from being restarted manually.

Output PRO_ACT & PGNO_REQ

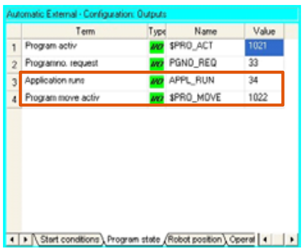


Program status

PRO_ACT – Program active
This output is always set if a process or the program execution is active at robot level. The process is therefore active as long as a program or an interrupt is being processed. Program processing is set to the inactive state at the end of the program only after all pulse outputs and all triggers have been processed.

PGNO_REQ – Program number request
A change of signal at this output requests the host computer to send a program number. If the variable PGNO_TYPE has the value 3, PGNO_REQ is **not** evaluated.

Output APPL_RUN & PRO_MOVE

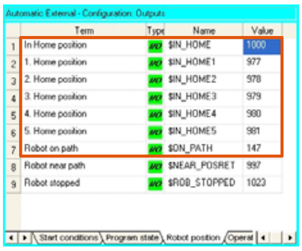


Program status

APPL_RUN – Application running
By setting this output, the robot controller communicates to the host computer the fact that a program is currently being executed.

PRO_MOVE – Program motion active
Means that a synchronous axis is moving, including in jog mode. The signal is thus the inverse of $ROB_STOPPED.

Output IN_HOME & ON_PATH



Automatic External - Configuration: Outputs

	Term	Type	Name	Value
1	In Home position		$IN_HOME	1000
2	1. Home position		$IN_HOME1	977
3	2. Home position		$IN_HOME2	978
4	3. Home position		$IN_HOME3	979
5	4. Home position		$IN_HOME4	980
6	5. Home position		$IN_HOME5	981
7	Robot on path		$ON_PATH	147
8	Robot near path		$NEAR_POSRET	997
9	Robot stopped		$ROB_STOPPED	1023

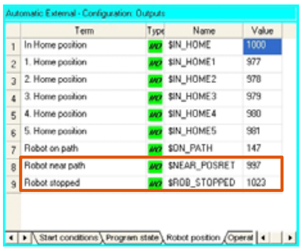
Start conditions | Program state | Robot position | Operat

Robot position

IN_HOME – In HOME position
This output communicates to the host computer whether or not the robot is in its HOME position.

ON_PATH – Robot on the path
This output remains set as long as the robot stays on its programmed path. The output ON_PATH is set after the BCO run. This output remains set until the robot leaves the path, the program is reset or block selection is carried out. The ON_PATH signal has no tolerance window, however; as soon as the robot leaves the path the signal is reset.

Output NEAR_POSRET & ROB_STOPPED



Automatic External - Configuration: Outputs

	Term	Type	Name	Value
1	In Home position		$IN_HOME	1000
2	1. Home position		$IN_HOME1	977
3	2. Home position		$IN_HOME2	978
4	3. Home position		$IN_HOME3	979
5	4. Home position		$IN_HOME4	980
6	5. Home position		$IN_HOME5	981
7	Robot on path		$ON_PATH	147
8	Robot near path		$NEAR_POSRET	997
9	Robot stopped		$ROB_STOPPED	1023

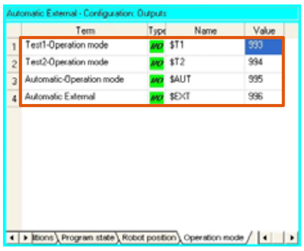
Start conditions | Program state | Robot position | Operat

Robot position

NEAR_POSRET – Robot near the path
This signal, NEAR_POSRET, allows the host computer to determine whether or not the robot is situated within a sphere about the position saved in $POS_RET. The host computer can use this information to decide whether or not the program may be restarted.

ROB_STOPPED – Robot not in motion
This signal is set if all the axes are in position, i.e. the robot is stationary. In the event of a WAIT command, this output is set during the wait.

Operating mode output



Operating mode

Operating modes: T1 – T2 – AUT – EXT
These outputs are set when the corresponding operating mode is selected.

11. Submit – Interpreter

SPS.SUB - Meaning and purpose

The KRC has two internal tasks that run in parallel:

- Robot interpreter
- **Controller interpreter**

- Robot operator and control tasks
- Runs independently of the selected robot program
- Cannot be used with time-critical applications
- Examples: monitoring a cooling circuit or safeguard

This renders the use of an additional PLC unnecessary for smaller applications as these tasks can be accommodated by the KRC.

SPS.SUB - Programming

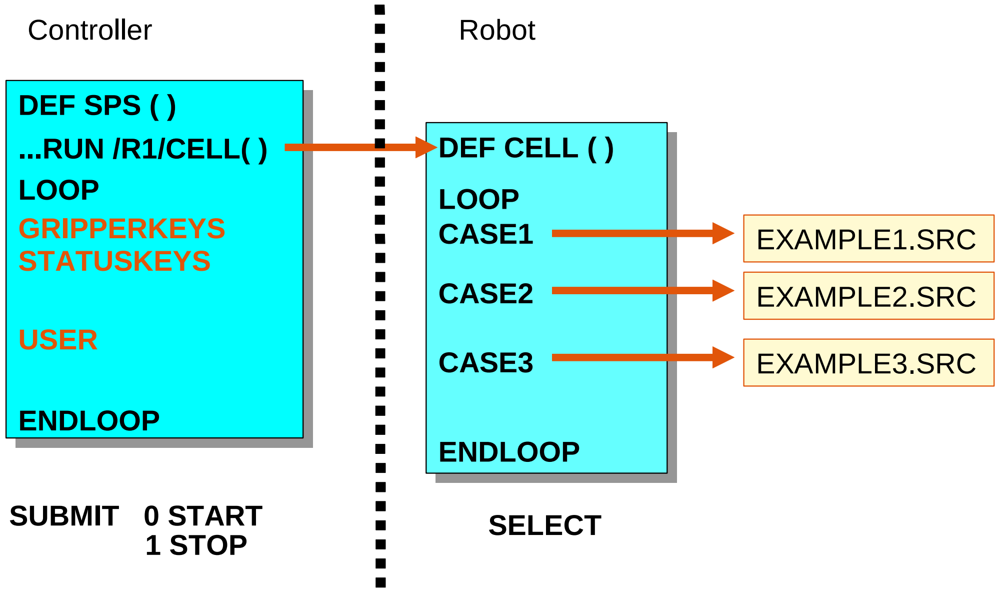
```
DEF SPS ( )

DECLARATIONS

INI

LOOP
GRIPPERKEYS
STATUSKEYS

USER

ENDLOOP
```

Declaration section for SUBMIT

Initialization section for SUBMIT

Submit interpreter programs that run continuously in the background

SPS.SUB - Programming



SPS.SUB - Features

- Only controller commands (e.g. $OUT[x]) can be used in SPS.SUB, no motion commands.
- Commands which are only to be executed once after running up the system can be inserted before the LOOP.
- Within the LOOP, calculations, monitoring or other actions can be executed parallel to the robot process.
- The LOOP is run cyclically (PLC philosophy) and should not be held up by wait commands.

Caution: SPS.SUB must be stopped before it is edited.

Frequently-used system variables in the Submit interpreter (1)

$MODE_OP

Value	*Explanation*
#T1	Robot controller is in T1 mode
#T2	Robot controller is in T2 mode
#AUT	Robot controller is in Automatic mode
#EX	Robot controller is in Automatic External mode
#INVALID	Robot controller has no defined state

Frequently-used system variables in the Submit interpreter (2)

$OV_PRO = *Value*

Argument	*Data type*	*Explanation*
Value (%)	INT	Program override value

Programming example:
If the programmed velocity cannot be reached, output 2 remains set to 'FALSE':

```
...
IF (($MODE_OP == #T1) OR ($OV_PRO < 100)) THEN
$OUT[2] = FALSE
ENDIF
...
```

12. Exercises

Exercise: Creating a program sequence 180
Exercise: Recapitulation of TOOL/BASE calibration 183
Exercise: Navigator (Expert) and endless loop................................ 186
Exercise: Simple data types 188
Exercise: Arrays with simple data types and counting loop.............. 191
Exercise: Creating a structure – Box 194
Exercise: Creating an enumeration type - ENUM color 196
Exercise: Subprograms with parameter transfer 198
Exercise: Data manipulation - Position calculation 201
Exercise: Motion programming – Palletizing / Depalletizing 205
Exercise: System variable “Timer” – Cycle time measurement 208
Exercise: Use of loops 211
Exercise: Program optimization / Program intelligence 214
Exercise: Automatic External.. 216

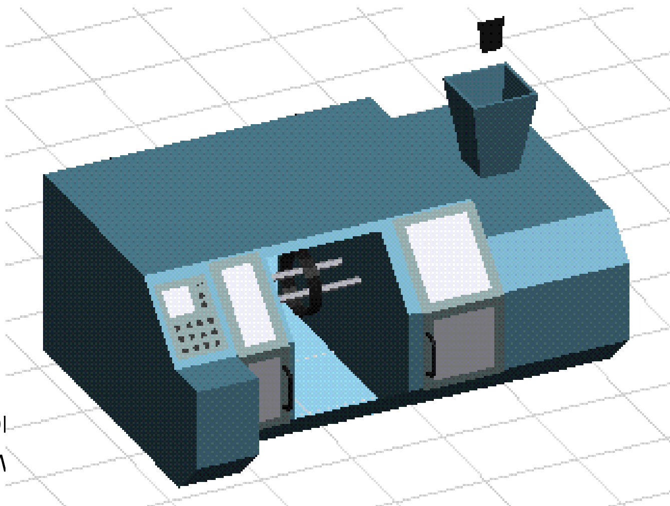
Exercise: Creating a program sequence

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Break down the overall task into subtasks
- Refine a rough breakdown
- Create a program flowchart

Prerequisites:

- Theoretical knowledge of the chapter “Programming methodology”



Necessary equip

- Office PC w

Reading materials:

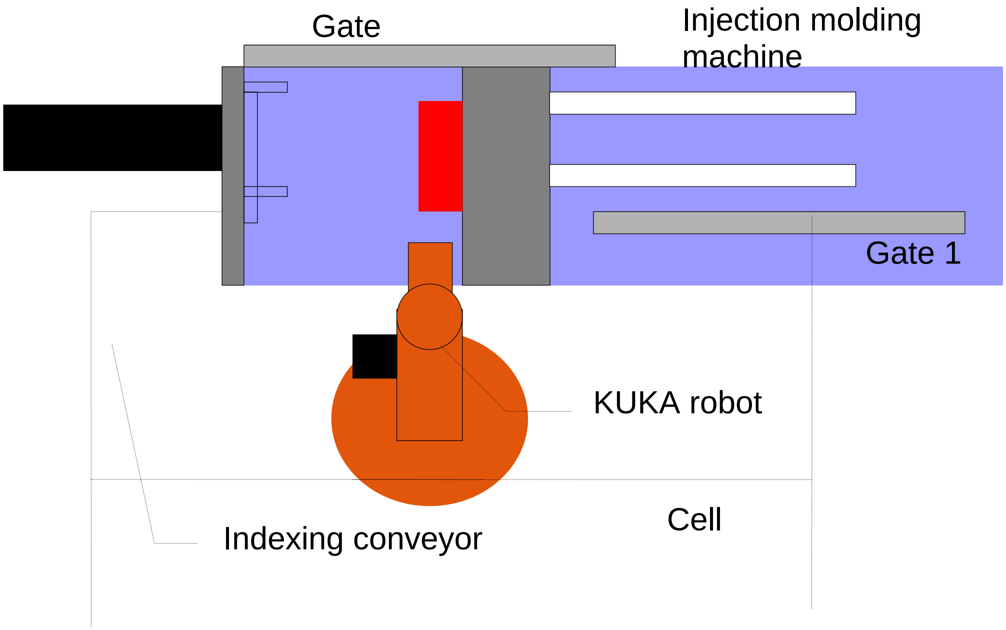
Training documentation:

- Workbook “Advanced Robot Programming” Release 5.x – Chapter “Programming methodology”

Task:

System description

The robot's task is to remove plastic components from an injection molding machine. The components are gripped using suction cups and stacked up on an indexing conveyor next to the injection molding machine.



Procedure

Once the injection molding machine (IMM) has manufactured a part, gate 1 is opened. After checking the limit switch, the robot moves to the unloading position and grips the part. The ejector pushes the part out of the mold. The robot now moves out of the machine and the ejector is retracted. As soon as the robot has moved safely out of the machine, gate 1 can be closed and a new part can be manufactured.

The finished part is now set down in the free space on the indexing conveyor. The indexing conveyor is then moved until the setdown position is free again.

Input/output configuration

;+++++++++++ **OUTPUTS** ++++++++++++++	
CLOSE TOOL GATE 1	$OUT[1]
RETRACT EJECTOR	$OUT[2]
ADVANCE EJECTOR	$OUT[3]
VACUUM ON	$OUT[18]
VACUUM OFF	$OUT[19]
INDEXING CONVEYOR ON	$OUT[20]
;+++++++++++++ **INPUTS** +++++++++++++	
TOOL GATE 1 OPEN	$IN[1]
TOOL GATE 1 CLOSED	$IN[2]
EJECTOR IS RETRACTED	$IN[3]
EJECTOR FORWARD POSITION	$IN[4]
MACHINE IN AUTOMATIC	$IN[5]
VACUUM ACHIEVED	$IN[9]
INDEXING CONVEYOR FREE	$IN[10]

Tasks

- Break the task down into appropriate program modules
- Refine your preliminary task break-down
- Create a program flowchart

What you should now know:

1. What demands are placed on a program?

   ....
   ....
2. What is meant by “requirements specification”?

   ....
   ....
3. What are the advantages of good legibility/structuring of a program?

   ....
   ....
4. What types of loop are available in the KUKA controller?

   ....
   ....
5. What must be taken into consideration when using comments?

   ....
   ....

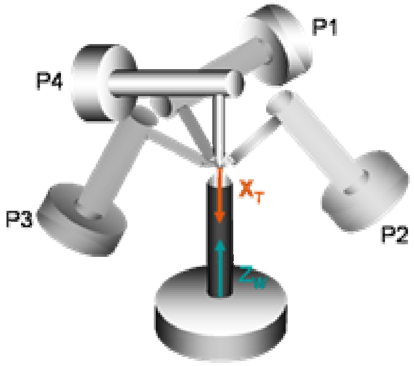
Exercise: Recapitulation of TOOL/BASE calibration

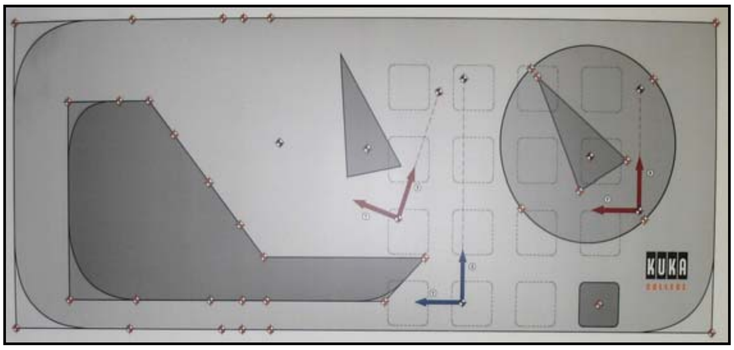
Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Tool calibration using the XYZ – 4-Point and World 5D methods
- Tool calibration by entering numerical values
- Base calibration using the 3-point method

Prerequisites:

- Theoretical knowledge of tool calibration
- Theoretical knowledge of base calibration





Necessary equipment:

- Pen 1 from the pen magazine

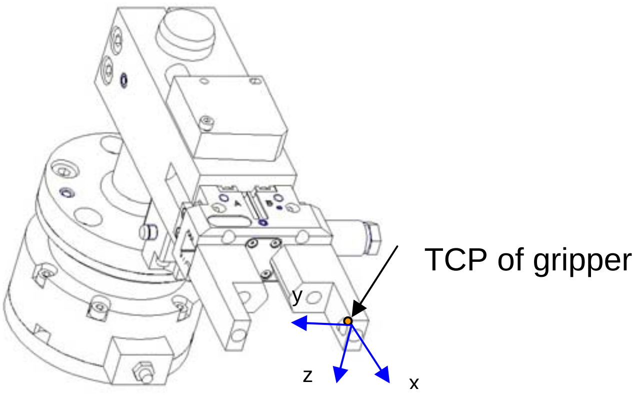
Reading materials:
Standard documentation:

- Operating Handbook, chapter “Start-up”

Task:

Calibrating the gripper

- Calibrate the TCP of the gripper using “numeric input”. Assign the tool number 3 and the name “gripper”. The calibration data are: X=132.05, Y=171.3, Z=173.0, A=45°, B=0°, C=0°



Calibration of pen 1

- Use the uppermost pen from the pen magazine and clamp it manually in the gripper. Calibrate the pen using the XYZ – 4-Point and ABC World 5D methods. Use the tool number 2 and the name “pen 1”.

Calibration of the blue BASE

- BASE number 2 is calibrated using pen 1 and the 3-point method. The name “blue base” is used.

What you should now know:

1. What are the advantages of using a “work base” on the table?

   ....
   ....
   ...
2. Where is the reference point for the calibration of a tool?

   ....
   ....
3. How many different BASE systems can KUKA software V5.x manage?

   ....
   ....
4. What is the difference between the ABC-World 5D and 6D methods?

   ....
   ....
5. Where is there an uncalibrated base (on delivery)?

   ....
   ....

Exercise: Navigator (Expert) and endless loop

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Creation of modules at Expert level
- Linking of individual modules in a main program
- Use of an endless loop

Prerequisites:

- Theoretical knowledge of the Navigator at Expert level
- Theoretical knowledge of the use of global subprograms
- Theoretical knowledge of an endless loop

Fetch cube

Set down cube

Necessary equipment:

- Training cell

Reading materials:

Standard documentation:

- Operating Handbook, chapter “Start-up”

Training documentation:

- Workbook, chapter “Navigator”
- Workbook, chapter “Subprograms and functions – Definition”
- Workbook, chapter “Program execution control – Loops”

Task:

Procedure

- Remove a cube from the cube magazine. The start and end position is the HOME position. Next, place the cube back into the magazine. Both processes are then to be linked and left to run continuously. Optimize the HOME position if necessary.

Task

- Create two appropriate modules (Setdown_cube/Fetch_cube).
- These are to be linked in the module MAGAZINE.
- Before commencing programming, create a program flowchart
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What is the difference between selecting MODULE and EXPERT when creating a program?

   ………………………………………………………………………………………
   ………………………………………………………………………………………
2. What is meant by the program run mode ISTEP?

   ………………………………………………………………………………………
   ………………………………………………………………………………………
3. What is the difference between SELECTING a program and OPENING it?

   ………………………………………………………………………………………
   ………………………………………………………………………………………
4. What does the syntax for a “Fold” look like?

   ………………………………………………………………………………………
   ………………………………………………………………………………………
5. What are the consequences of subsequently changing the HOME position?

   ………………………………………………………………………………………
   ………………………………………………………………………………………

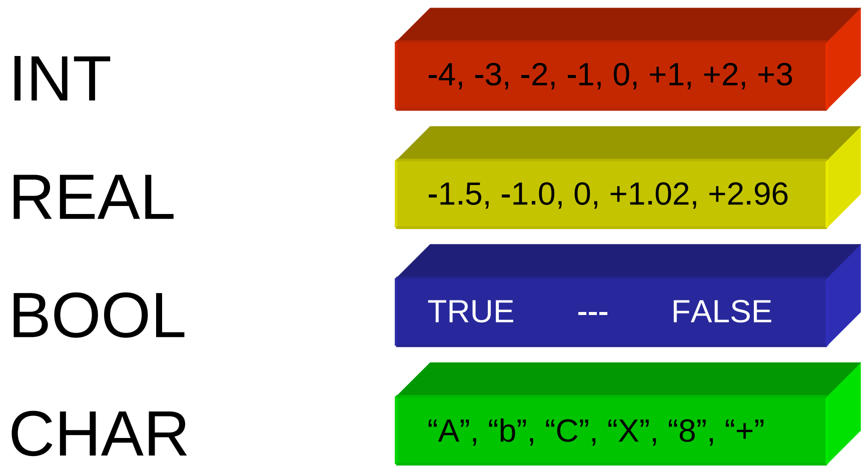
Exercise: Simple data types

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Using simple data types
- Declaration, initialization and use of variables
- Correct use of variables with regard to their lifetime

Prerequisites:

- Theoretical knowledge of simple data types and how they are handled



Necessary equipment:

- Training cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Variables and declarations – Simple data types”

Task:

Description

Expand the programs you have created to include the following:

- A variable that counts the number of cubes transported since the last program selection.
- A variable that saves the absolute number of all cubes transported so far.
- A variable that adds up the combined weight (in kg) of all cubes transported. A cube weighs 0.329 kg.
- A variable that is set to TRUE while the cube is being fetched and is otherwise set to FALSE.
- A variable that contains the letter “O” when the gripper is open and the letter “C” when the gripper is closed. In the undefined state, the variable is assigned the value “X”.

Use appropriate variable names and data types. It is also important to declare the variable in the correct place.

Task

- Define where your variables are declared.
- Expand your existing program flowchart to include these new variables.
- Observe the different initializations of the variables.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What is the maximum length of a variable name?

   ....
   ....
2. What simple data types are there?

   ....
   ....
3. Where are variables declared in the SRC file?

   ....
   ....

   What is the lifetime of a variable declared in **`$CONFIG.DAT`**?

   ....
   ....
4. How do you declare a floating-point number with the name PAUL in the DAT file with the value 138.74?

   ....
   ....

Exercise: Arrays with simple data types and counting loop

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Declare and initialize arrays with simple data types
- Process individual array elements
- Work with the variable display (Configure/Show)

Prerequisites:

- Knowledge of simple data types
- Theoretical knowledge of arrays
- Theoretical knowledge of the FOR loop
- Theoretical knowledge of the variable display

Pre-initialization:

1	2	3	4	5	6	7	8	9	10	11	12
O	O	O	O	O	O	O	O	O	O	O	O

Program execution:

1	2	3	4	5	6	7	8	9	10	11	12
X	X	X	X	X	X	O	O	O	O	O	O

6 cubes set down

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter "Expert Programming"
- Operating Handbook, chapter "Operator control"

Training documentation:

- Workbook, chapter "Variables and declarations – Arrays"

<u>Task:</u>

Description

Expand the program you have created to include the following:

- Create a one-dimensional array of size 12. The contents of the individual boxes should be the letters "O" and "X".
- Before the program is executed, the pre-defined content for all boxes should be the letter "O"
- Once the first cube has been set down, the first box should receive the letter "X". The same procedure is to apply for each subsequent setdown operation.
- Check the current array index and the contents of the array.

Task

- Define where your variables are declared.
- Expand your existing program flowchart to include this array.
- Create a new group to display your required array variables.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. How is the size of an array defined?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

2. What is the difference between declaring an array in the SRC file and declaring it in the DAT file?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

3. What error message is generated if the array index is exceeded?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

4. Declare a 3-dimensional array with the name Cabinet_price. The cabinet price is calculated from the components length, width and depth. The range includes 5 different lengths, 3 different widths and 2 different depths.

   ……………………………………………………………………………………
   ……………………………………………………………………………………

5. What can be activated in the variable display by means of the softkey “Start info”?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

Exercise: Creating a structure – BOX_Type

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Create your own structures (declaration, initialization)
- Work with the point separator

Prerequisites:

- Theoretical knowledge of simple data types and structures

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Variables and declarations – Structures”

Task:

Task

Create a program with the name “Structure”.

- Generate a structure with the name “BOX_Type”. This structure is to be able to store the following variables:
    - Length
    - Width
    - Height
    - Painted (yes/no)
    - Number of bottles
- Assign the structure BOX_Type to the variable CRATE and initialize it with the following start values:
    - Length 25.2
    - Width 18.5
    - Height 5.0
    - Contents: 4 bottles
    - Not painted

- In the first operation, a further 8 bottles are added to the crate. In the second step, the CRATE is painted. Plan a wait time of 3 seconds between steps one and two.
- Display the contents of the variable CRATE during program execution.

What you should now know:

1. What is a structure?

   ……………………………………………………………………………………
   ……………………………………………………………………………………
2. What predefined KUKA structures are there?

   ……………………………………………………………………………………
   ……………………………………………………………………………………
3. What is the difference between a POS structure and an E6POS structure?

   ……………………………………………………………………………………
   ……………………………………………………………………………………
4. Name a FRAME structure you already know.

   ……………………………………………………………………………………
   ……………………………………………………………………………………
5. What is a point separator?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

Exercise: Creating an enumeration type - ENUM color

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Create your own ENUM variables (declaration)
- Work with ENUM variables

Prerequisites:

- Theoretical knowledge of simple data types and structures
- Theoretical knowledge of simple ENUM variables

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Variables and declarations – ENUM”

Task:

Task

Expand your program “Structure” to include the following:

- Generate an ENUM variable containing the colors red, yellow, green and blue.
- Create a variable with the name Lamp and assign it the color blue.
- In the second step, assign the lamp the color yellow.
- Expand your “BOX_Type” structure to include the aggregate “Color” and assign the following colors to the CRATE: first red, then yellow, and finally green.
- Display the contents of the variable CRATE during program execution.

What you should now know:

1. Who defines the number of constants and their names in the enumeration type ENUM?

   ....
   ....

2. When is the symbol “#” used?

   ....
   ....

3. What is wrong in the following declaration?

   ```
   ENUM Day_type mo, di, mi, do, fr, sa, so
   ```

   ....
   ....

Exercise: Subprograms with parameter transfer

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Subprograms with transfer of parameters

Prerequisites:

- Knowledge of working with subprograms
- Theoretical knowledge of parameter transfer
- Theoretical knowledge of functions

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Subprograms and functions”

Task:

Please complete the results following program execution.

Main program:

```
DEF var_transf( )
INT A, B, C, D, E, F
A=50
B=48
C=200
D=300
E=8
F=5

CALC (C, D, A, F, B, E)
;A = ....................     B = ....................          C = ....................
;D = ....................     E = ....................          F = ....................

END
```

Subprogram:

```
DEF CALC (X1:OUT, X2:IN, X3:OUT, X4:IN, X6:IN, X5:OUT)
INT X1, X2, X3, X4, X5, X6

; Calculation
X1 = X1+211          ; X1 = ....................
X2 = X2-312          ; X2 = ....................
X3 = X2 +X3          ; X3 = ....................
X4 = X4 * X5         ; X4 = ....................
X5 = X5 * 8          ; X5 = ....................
X6 = X6 - 40         ; X6 = ....................

END
```

What you should now know:

1. What is the maximum nesting depth for subprograms?

   ....
   ....

2. What DAT file is accessed by a local subprogram?

   ....
   ....

3. Where is a variable declared if it is to be recognized both in the global subprogram and in the main program?

   ....
   ....

4. What command is used to transfer a return value from a function to the main program?

   ....
   ....

5. What is the difference, in parameter transfer, between “IN parameter” and “OUT parameter”?

   ....
   ....

Exercise: Data manipulation - Position calculation

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Work with arrays
- Work with structures and the point separator
- Calculate position coordinates
- Use nested FOR loops

Prerequisites:

- Knowledge of arrays, structures and the FOR loop
- Theoretical knowledge of data manipulation

Necessary equipment:

- Training robot cell

Reading materials:

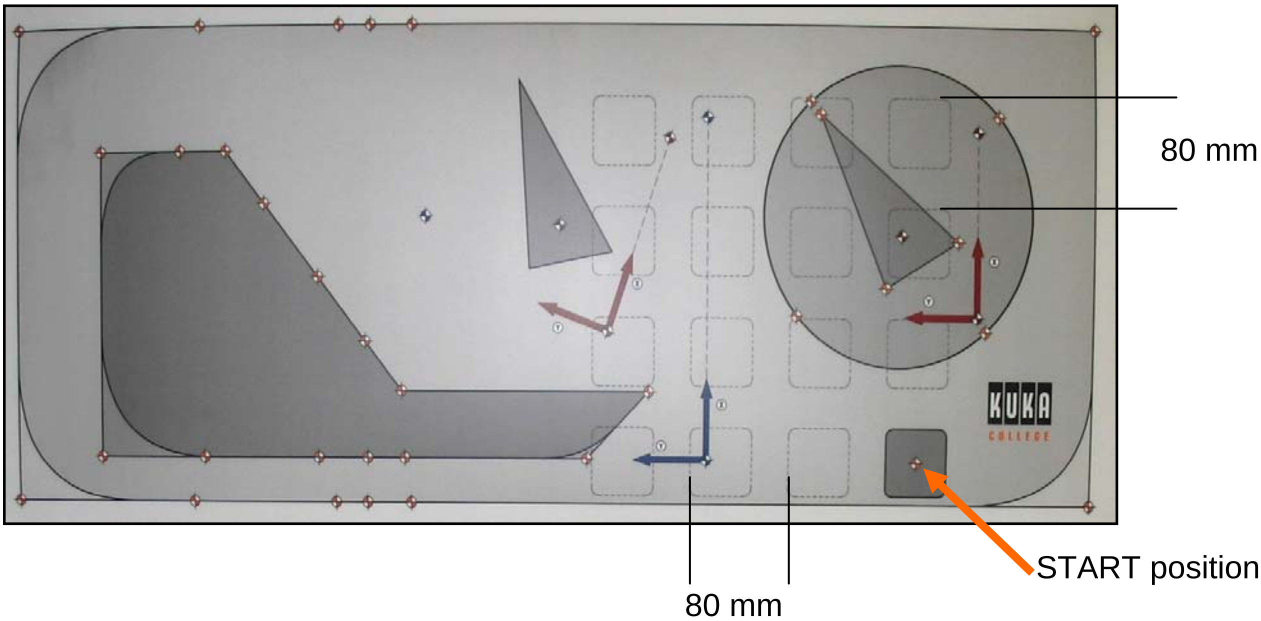
Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Variables and subprograms”
- Workbook, chapter “Data manipulation”

Task:



1. Subtask: Calculation of setdown position

   a. Description

      Starting from the START position indicated in gray, calculate all the other positions on the table. Cubes will later be set down in the calculated positions. The distances between the individual setdown positions are 80 mm on all sides. Teach the START position by moving the robot, with the cube clamped in place, to the position indicated in gray.

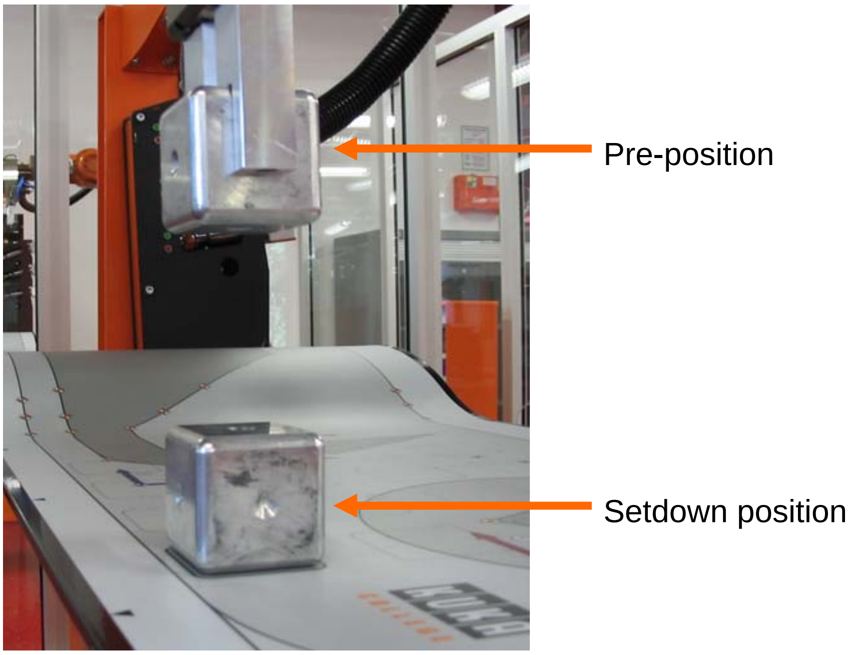
   b. Tasks

      - Create a program flowchart for the task described.
      - Create suitable variables for the program “CALCULATION”. Use a module for program creation.
      - Initialize your variables with appropriate start values.
      - Calculate the 16 setdown positions on the worktable.

2. Subtask: Calculation of pre-position

   a. Description

      Additional positions are required for subsequent setdown operations. In addition to the setdown positions, a pre-position is required for each cube. This position is 100 mm above the actual position of the cube on the table.



   b. Tasks

      - Expand your existing program flowchart to include this new function.
      - Expand your program “CALCULATION” to include this pre-position (declaration, initialization, calculation).

What you should now know:

1. Calculate the values for A, B, C, X, Y, Z:

```
INT X, Y, Z
REAL A, B, C
A= 4.5
B= -2
C= 4.1
X= 2.5
Y= 4
Z= 0.1
A= A * B + Y                  ;A= ...............
B= B * Z + X                  ;B= ...............
C= C + 4/3                    ;C= ...............
X= X + 2.5 * 4                ;X= ...............
Y= Y - 10.0/4                 ;Y= ...............
Z= 14 - Z * C + A             ;Z= ...............
```

2. Calculate the values for O, P, Q:

```
INT O, P, Q
O= 11 B_OR 44                 ;O= ...............
P= 21 B_AND 18                ;P= ...............
Q= 53 B_EXOR 35               ;Q= ...............
```

3. Calculate the values for H, I, J, K:

```
BOOL T, F, H, I, J, K
T= TRUE
F= FALSE
H= (7/3 >2) AND T             ;H= ...............
I= (17-4*3>10) EXOR NOT T     ;I= ...............
J=NOT (T OR F) AND (F OR T)   ;J= ...............
K=NOT T OR F AND F OR T       ;K= ...............
```

Exercise: Motion programming – Palletizing / Depalletizing

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Program motions without using inline forms.
- Move the robot to calculated target coordinates.

Prerequisites:

- Theoretical knowledge of motion programming without inline forms

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

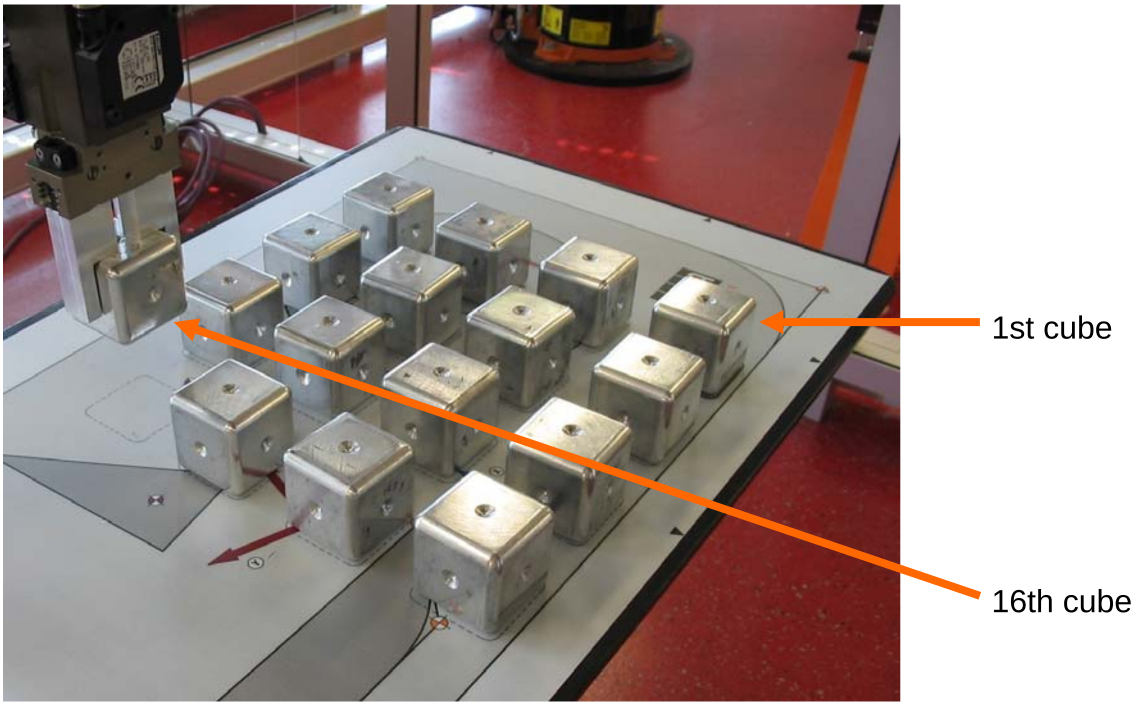
Training documentation:

- Workbook, chapter “Motion programming”

Task:

Description

Duplicate your program CALCULATION with the new name “Pallet”.
Now that you have calculated the 16 setdown positions, it only remains to fetch the cube from the magazine and set it down in the corresponding position on the table. Once all 16 cubes have been set down, they should automatically be cleared away from the table. The cube that was set down first is also to be cleared away first.



Tasks

- Create a program flowchart for palletizing/depalletizing. A block with calculation is sufficient for calculation of the coordinates.
- Create a program for the entire operation.
- “Pack” your calculation in suitable Folds.
- Make appropriate use of existing modules.
- Check the different variables in the variable display.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What is set using the system variable $IPO_MODE?

   ....
   ....
2. What is specified as CA in circular motion?

   ....
   ....
3. What can be done using the command “CONTINUE”?

   ....
   ....
4. What must be taken into consideration when using the “CONTINUE” command?

   ....
   ....
5. What information is conveyed by Turn ‘T22’?

   ....
   ....

Exercise: System variable “Timer” – Cycle time measurement

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Work with a timer (initialize, start, stop).
- Optimize cycle times.

Prerequisites:

- Knowledge of the system variable $TIMER[x].

Necessary equipment:

- Training cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Expert Programming”

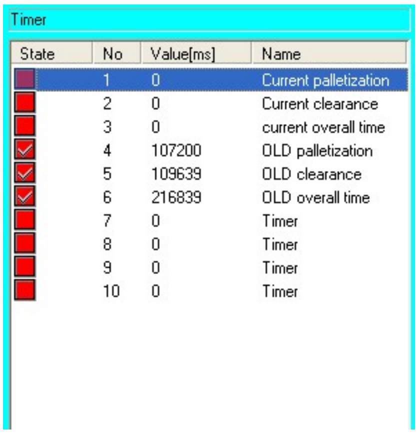
Training documentation:

- Workbook, chapter “System variables”

Task:

Description

Duplicate your program “Pallet” and rename it “Time_measurement”. You should record three times. Timer 1 is to time the palletization and timer 2 is to time the clearing operation. Timer 3 calculates the overall time of the two operations. Make sure that you do not start and stop the timer with the advance run.



Task

- Use $TIMER[1] for timer 1 and also save the final value in $TIMER[4]. Assign the description of timer 1 the name “Current palletization” and assign the saved value the name “OLD palletization”.
- $TIMER[2] is used for timer 2 and the value is saved in $TIMER[5]. Here, once again, appropriate names are to be assigned in the display.
- $TIMER[3] is used for timer 3 and the value is saved in $TIMER[6]. Here, once again, appropriate names are to be assigned in the display.
- Try to optimize your program.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. How many timers does the KUKA controller have and how are they started?

   ....
   ....

2. Give an example of a variable in `$CONFIG.DAT`.

   ....
   ....

3. Where are the variables for the software limit switches located?

   ....
   ....

4. What advantages does a CYCFLAG have over a FLAG?

   ....
   ....

5. How many CYCFLAGs are available in the KUKA controller?

   ....
   ....

Exercise: Use of loops

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Substitute FOR loops with other loops
- Use WHILE or REPEAT/UNTIL loops

Prerequisites:

- Theoretical knowledge of the functionality of the different loop techniques in programming

Necessary equipment:

- Training robot cell

Reading materials:
Standard documentation:

- Programming Handbook, chapter “Expert Programming”

Training documentation:

- Workbook, chapter “Program execution control”

Task:

Description

In the program section “Palletization”, change the double FOR loop into two WHILE loops. During depalletization, the existing loops are to be converted into REPEAT/UNTIL loops.

Task

- Duplicate your program “Pallet” with the new name “Pal_loop”.
- Update your existing program flowchart for the new loops.
- Create new counting variables for your loops. Make sure that your variables are correctly initialized or suitably reset.
- Modify the way the cubes are cleared from the table so that the last cube cleared is the first one to be picked up and set back down again.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. In connection with SWITCH/CASE, there is also a "DEFAULT" statement.

   What is the purpose of the "DEFAULT" statement?

   ....
   ....
2. What command can you use with the FOR loop to set the increment?

   ....
   ....
3. What loops can be left using the "EXIT" command?

   ....
   ....
4. Which component can be omitted when using a branch?

   a. IF

   b. THEN

   c. ELSE

   d. ENDIF
5. What is wrong in the following example program?

```
IF $IN[14]==FALSE THEN
     $OUT[12]=TRUE
     GOTO MARK1
ELSE
     $OUT[12]=FALSE
     GOTO MARK2
WHILE $IN[17]==TRUE
     PTP P1
     MARK1:
     ...
ENDWHILE
MARK2:
…
PTP HOME
```

   ....
   ....

Exercise: Program optimization / Program intelligence

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Program subprogram calls
- Program an endless loop

Prerequisites:

- Knowledge of the Navigator at Expert level
- Knowledge of subprogram and loop programming

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter "Expert Programming"

Training documentation:

- Workbook, chapter "Variables and declarations"
- Workbook, chapter "Program execution control"

Task:

Description

Optimize your program "Pal_loop" by expanding it so that you can deselect it at any time and resume program execution – on restarting the program – from the point at which it was stopped.

Task

- Define in your program sequence whether you are palletizing or depalletizing.
- Evaluate the state of the gripper (e.g: open $IN[26] / closed with cube $IN[29]) and resume the program accordingly.
- Check during initialization of the basic values whether this is necessary, or whether you wish to continue working with the saved values.
- Update your existing program flowchart for this new function.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. Which variable is responsible for the advance run pointer?

   ……………………………………………………………………………………
   ……………………………………………………………………………………
2. How is a one-dimensional array transferred?
   a. SUBPRG(VALUE(( ))
   b. SUBPRG(VALUE[255])
   c. SUBPRG(VALUE[ ])
   d. SUBPRG(VALUE[x])
3. In which structure can you save the position of a robot with a two-axis positioner (2 external axes)?
   a. POS
   b. E6POS
   c. AXIS
   d. E6AXIS
   e. FRAME
4. What is the correct syntax for a “non-rejecting loop”?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

Exercise: Automatic External

Aim of the exercise:
On successful completion of this exercise, you will be able to carry out the following activities:

- Targeted integration of a robot program into Automatic External operation
- Adaptation of the “Cell” program
- Configuration of the Automatic External interface
- Recognition of sequence in Automatic External mode

Prerequisites:

- Knowledge of how to edit the “Cell” program
- Knowledge of how to configure the Automatic External interface
- Theoretical knowledge of the signal sequence in Automatic External

Necessary equipment:

- Training robot cell

Reading materials:

Standard documentation:

- Programming Handbook, chapter “Configuration”

Training documentation:

- Workbook, chapter “Automatic External interface”

Task:

Description:

You have individual functioning modules and these are to be made available to the Automatic External interface.

Task

- Configure the Automatic External interface to the requirements of your control panel.
- Expand your Cell program by any three modules, the functions of which you have verified beforehand.
- Use buttons to simulate the functionality of the PLC.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

<u>Photograph:</u> Control panel at KUKA College for controlling Automatic External

What you should now know:

1. What is the precondition for PGNO_REQ not to be evaluated?

   ....
   ....
2. What signal is used to activate the drives and what must be taken into consideration?

   ....
   ....
3. Which Automatic External interface variable also influences jogging?

   ....
   ....
4. Which Fold in the CELL program checks the HOME position?

   ....
   ....
5. What are the preconditions for Automatic External mode?

   ....
   ....
   ....
   ....
   ....
   ....
   ....
   ....