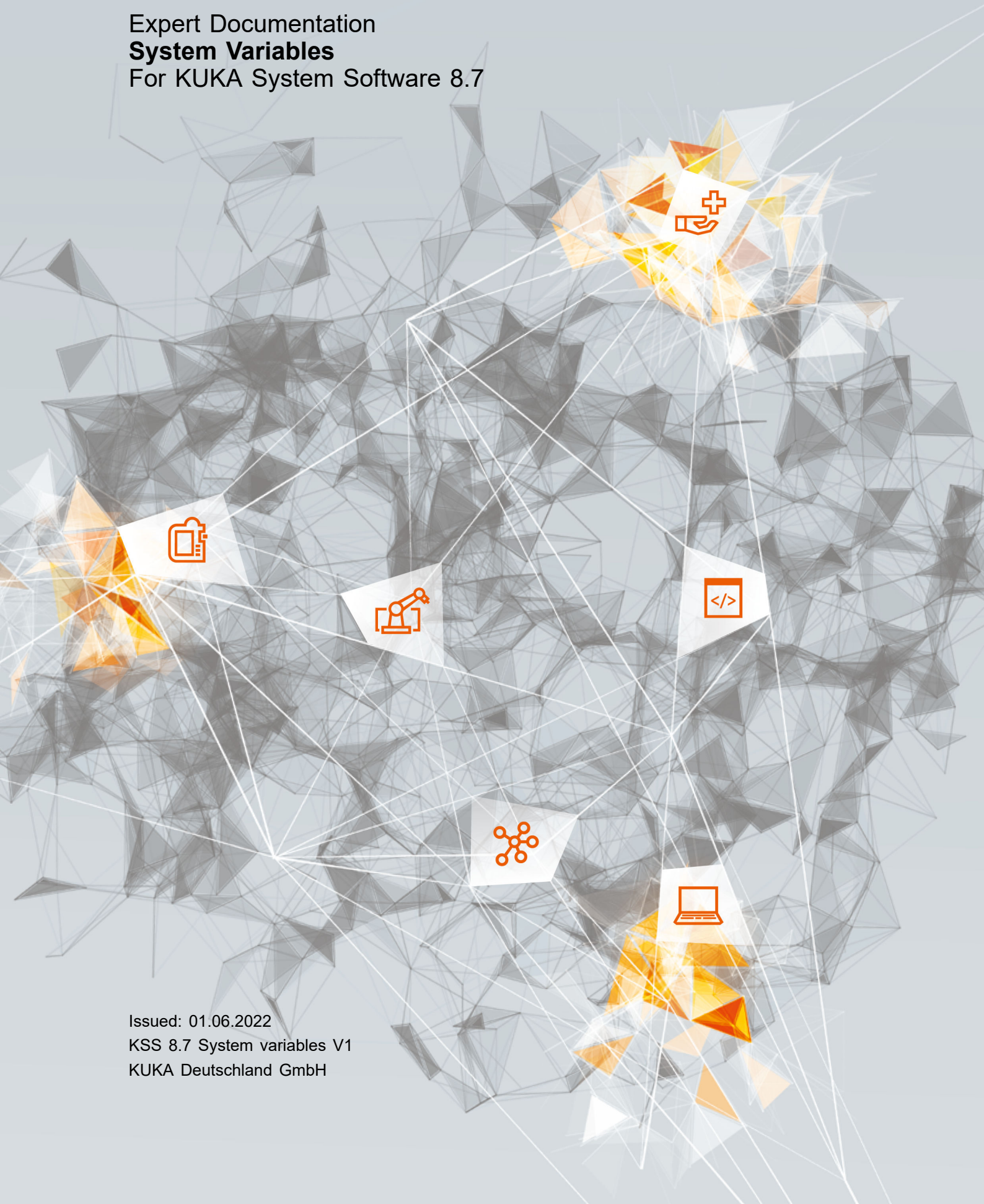
KUKA

Expert Documentation
System Variables
For KUKA System Software 8.7



Issued: 01.06.2022
KSS 8.7 System variables V1
KUKA Deutschland GmbH

KUKA Deutschland GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

KIM-PS5-DOC

Translation of the original documentation

Publication:	Pub KSS 8.7 System variables (PDF) en PB19967
Book structure:	KSS 8.7 System variables V1.1 BS17260
Version:	KSS 8.7 System variables V1

Contents

1 Introduction

1.1 Target group

This documentation is aimed at users with the following knowledge and skills:

- Advanced knowledge of the robot controller system
- Advanced KRL programming skills

For optimal use of KUKA products, we recommend the training courses offered by KUKA College. Information about the training program can be found at www.kuka.com or can be obtained directly from our subsidiaries.

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

- Documentation for the robot arm
- Documentation for the robot controller
- Documentation for the smartPAD-2 or smartPAD pro (if used)
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Spare parts overview in KUKA Xpert

Each set of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

These warnings are provided for safety purposes and **must** be observed.

DANGER

These warnings mean that it is certain or highly probable that death or severe injuries **will** occur, if no precautions are taken.

WARNING

These warnings mean that death or severe injuries **may** occur, if no precautions are taken.

CAUTION

These warnings mean that minor injuries **may** occur, if no precautions are taken.

NOTICE

These warnings mean that damage to property **may** occur, if no precautions are taken.

These warnings contain references to safety-relevant information or general safety measures.
These warnings do not refer to individual hazards or individual precautionary measures.

This warning draws attention to procedures which serve to prevent or remedy emergencies or malfunctions:

SAFETY INSTRUCTION
The following procedure must be followed exactly!

Procedures marked with this warning **must** be followed exactly.

Notices

These notices serve to make your work easier or contain references to further information.

Tip to make your work easier or reference to further information.

1.4 Information about the system variables

The documentation contains selected system variables. It does not cover all system variables available for the System Software.

2 Safety

The safety information for the industrial robot can be found in the “Safety” chapter of the Operating and Programming Instructions for System Integrators or the Operating and Programming Instructions for End Users.

Comply with safety-relevant information
The safe use of this product requires knowledge of and compliance with fundamental safety measures. Death, severe injuries or damage to property may otherwise result.

- The “Safety” chapter in the operating and programming instructions of the system software must be observed.

3 System variables

3.1 Signal declarations

The input and output signals that are available on the robot controller are declared in the file …STEU\Mada\$machine.dat.

Examples:

- Signals of the Automatic External interface
- Signals for the brake test and mastering test

The signals for the brake test and mastering test are not described in this documentation. Information about these signals can be found in the documentation of the safety options, e.g. KUKA.SafeOperation.

WARNING

Danger to life and limb due to incorrectly used signals
These signals are not redundant in design and can supply incorrect information. If they are used for safety-relevant applications, this may result in death, severe injuries or considerable damage to property.

- Do not use these signals for safety-relevant applications.

Input signals

By default, some input signals are routed to $IN[1025] or $[1026]. To use these signals, they must be assigned to other input numbers.

Signals that are not required can be deactivated with FALSE.

Output signals

Some output signals are preset to FALSE by default. There is no compelling need to assign output numbers to these signals. This is only necessary in order to be able to read these signals, for example via the variable display or program run.

Some output signals are assigned to specific output numbers by default. These signals can be assigned to other output numbers.

Signals that are not required can be deactivated with FALSE.

3.2 $ABS_ACCUR

Description

Use of the positionally accurate robot model

The variable can be used to check whether a positionally accurate robot model is saved on the RDC and whether it is being used.

Writability

The system variable is write-protected.

Syntax

state = `$ABS_ACCUR`

Explanation of the syntax

Element	Description
state	Type: ENUM ABS_ACCUR_STATE • #ACTIVE: positionally accurate robot model is loaded and is being used. A positionally accurate robot model is saved on the RDC. The variable $DEACTIVATE_ABS_ACCUR is FALSE. • #INACTIVE: positionally accurate robot model is loaded, but is not used. A positionally accurate robot model is saved on the RDC, but it is currently deactivated with $DEACTIVATE_ABS_ACCUR == TRUE (until the next cold start of the robot controller). • #NONE: standard robot model. No positionally accurate robot model is used. No positionally accurate robot model is saved on the RDC. The value of the variable $DEACTIVATE_ABS_ACCUR is irrelevant.

3.3 $ABS_CONVERT

Description

Conversion to positionally accurate Cartesian robot poses

The variable can be used to convert non-positionally accurate Cartesian robot poses to positionally accurate ones. For example, Cartesian robot poses in motion programs that were taught without a positionally accurate robot model can be nominally converted to positionally accurate coordinates and stored in the associated data list. For this purpose, the variable $ABS_CONVERT is set to TRUE.

In physical terms, there is no change in the way the converted robot poses are addressed. Bit 5 of the state specification indicates whether positionally accurate coordinates are stored for a point. If the bit is set (bit 5 = 1), the coordinates are positionally accurate and the point is not converted again.

It is advisable to set the variable $ABS_CONVERT to FALSE again as soon as the Cartesian robot poses have been converted.

Syntax

```
$ABS_CONVERT = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: conversion to positionally accurate Cartesian robot poses • FALSE: no conversion to positionally accurate Cartesian robot poses Default: FALSE

3.4 $ACC

Description

Acceleration of the TCP in the advance run

This system variable is used to define the Cartesian acceleration. The following limit values apply:

- **0.0 … $ACC_MA**
  The maximum Cartesian acceleration $ACC_MA is defined in the machine data.

If $ACC violates the limit values, the message *Value assignment inadmissible* is displayed. Program execution is stopped or the associated motion instruction is not executed during jogging.

Syntax

```
$ACC = {CP continuous path, ORI1 swivel, ORI2 rotation}
```

Explanation of the syntax

Element	Description
CP	Type: REAL; unit: m/s^2 Path acceleration • **0.0 … $ACC_MA.CP**
ORI1	Type: REAL; unit: $°/s^2$ Swivel acceleration • **0.0 … $ACC_MA.ORI1**
ORI2	Type: REAL; unit: $°/s^2$ Rotational acceleration • **0.0 … $ACC_MA.ORI2**

Example

```
$ACC = {CP 5.0, ORI1 500.0, ORI2 500.0}
```

3.5 $ACC_C

Description

Acceleration of the TCP in the main run

The system variable contains the programmed Cartesian acceleration for the current motion block and is the main run copy of the motion parameter .

Writability

The system variable is write-protected.

Syntax

```
{CP continuous path, ORI1 swivel, ORI2 rotation} = $ACC_C
```

Explanation of the syntax

Element	Description
CP	Type: REAL; unit: m/s^2 Path acceleration • **0.0 … $ACC_MA.CP**
ORI1	Type: REAL; unit: $°/s^2$ Swivel acceleration • **0.0 … $ACC_MA.ORI1**
ORI2	Type: REAL; unit: $°/s^2$ Rotational acceleration • **0.0 … $ACC_MA.ORI2**

3.6 $ACC_AXIS

Description

Acceleration of the robot axes in the advance run

The variable contains the planned axis acceleration as a percentage. In the case of motions planned using the dynamic model, the percentage value refers to the axis torque available for acceleration.

If no dynamic model is present, the percentage value refers to the axis ramp-up time configured in WorkVisual.

Syntax

`$ACC_AXIS[`*axis number*`] = `*acceleration*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6
acceleration	Type: INT; unit: % • **1 … 100**

3.7 $ACC_AXIS_C

Description

Acceleration of the robot axes in the main run

The variable contains the axis acceleration of the motion currently being executed as a percentage value. In the case of motions planned using the dynamic model, the percentage value refers to the axis torque available for acceleration.

If no dynamic model is present, the percentage value refers to the axis ramp-up time configured in WorkVisual.

Writability

The system variable is write-protected.

Syntax

acceleration = $ACC_AXIS_C[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6
acceleration	Type: INT; unit: % • **1 … 100**

3.8 $ACC_CAR_ACT

Description

Current Cartesian acceleration

The variable of structure type ACC_CAR contains the current Cartesian acceleration for the following components:

- X, Y, Z: Cartesian acceleration for X, Y, Z in [m/s^2]
- A, B, C: Cartesian acceleration for A, B, C in [°/s^2]. This acceleration is not evaluated.
- ABS: overall Cartesian acceleration in the XYZ space in [m/s^2] (absolute value of the acceleration in X, Y, Z)

The current Cartesian acceleration $ACC_CAR_ACT must not exceed the maximum Cartesian acceleration $ACC_CAR_LIMIT defined in the machine data.

(>>> *3.9 "$ACC_CAR_LIMIT" Page 19*)

To ensure this, the monitoring of the Cartesian acceleration must be activated in the machine data: $ACC_CAR_STOP = TRUE

(>>> *3.11 "$ACC_CAR_STOP" Page 20*)

If the monitoring is active, the manipulator stops with a STOP 2 if the maximum permissible acceleration is exceeded. Additionally, the acknowledgement message *Maximum Cartesian acceleration exceeded* is displayed.

3.9 $ACC_CAR_LIMIT

Description

Maximum Cartesian acceleration

The variable of structure type ACC_CAR defines the maximum permissible Cartesian acceleration for the following components:

- X, Y, Z: Maximum Cartesian acceleration for X, Y, Z in [m/s^2]
- A, B, C: The maximum Cartesian acceleration for A, B, C is not evaluated.
- ABS: Maximum overall Cartesian acceleration in the XYZ space in [m/s^2] (absolute value of the acceleration in X, Y, Z)

The current Cartesian acceleration $ACC_CAR_ACT may not exceed the maximum Cartesian acceleration $ACC_CAR_LIMIT. If the maximum permissible acceleration is exceeded, the robot is brought to a halt by ramp-

down braking (STOP 2) and the acknowledgement message *Maximum Cartesian acceleration exceeded* is generated.

The monitoring of the Cartesian acceleration is deactivated by default and must be activated using the variable $ACC_CAR_STOP. Otherwise, no ramp-down braking (STOP 2) occurs if the maximum Cartesian acceleration is exceeded.

Default

```
$ACC_CAR_LIMIT={X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0
ABS 0.0}
```

3.10 $ACC_CAR_MAX

Description

Maximum Cartesian acceleration

The variable of structure type ACC_CAR saves the value of the highest magnitude that the Cartesian acceleration $ACC_CAR_ACT reaches:

- X, Y, Z: Cartesian acceleration for X, Y, Z in [m/s^2]
- A, B, C: Cartesian acceleration for A, B, C in [$°/s^2$]. This acceleration is not evaluated.
- ABS: Overall Cartesian acceleration in the XYZ space in [m/s^2] (absolute value of the acceleration in X, Y, Z)

Example

The variable can be set to zero in the KRL program in order to determine the maximum values.

```
$ACC_CAR_MAX={X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0,
ABS 0.0}
```

3.11 $ACC_CAR_STOP

Description

Cartesian acceleration monitoring

If the monitoring is active and the maximum Cartesian acceleration $ACC_CAR_LIMIT is exceeded, the robot is brought to a halt by ramp-down braking (STOP 2) and the acknowledgement message *Maximum Cartesian acceleration exceeded* is generated.

Syntax

`$ACC_CAR_STOP=`*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Monitoring is active. • FALSE: Monitoring is not active. Default: FALSE

3.12 $ACC_CAR_TOOL

Description

Monitoring of the Cartesian acceleration in relation to the TCP or flange

The variable of structure type FRAME defines the point at which the Cartesian accelerations that affect a tool mounted on the flange are cyclically calculated:

- X, Y, Z: Position on the X, Y, Z axes in [mm]
- A, B, C: The orientation of the angles is not evaluated.

The accelerations determined at this point must not exceed the maximum value defined by the variable $ACC_CAR_LIMIT.

Default

```
$ACC_CAR_TOOL={X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0}
```

3.13 $ACC_EXTAX

Description

Acceleration of the external axes in the advance run

The variable contains the planned axis acceleration as a percentage. In the case of motions planned using the dynamic model, the percentage value refers to the axis torque available for acceleration.

If no dynamic model is present, the percentage value refers to the axis ramp-up time configured in WorkVisual.

Syntax

`$ACC_EXTAX[`*axis number*`] = `*acceleration*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: external axis E1 ... E6
acceleration	Type: INT; unit: % • **1 … 100**

3.14 $ACC_EXTAX_C

Description

Acceleration of the external axes in the main run

The variable contains the axis acceleration of the motion currently being executed as a percentage value. In the case of motions planned using the dynamic model, the percentage value refers to the axis torque available for acceleration.

If no dynamic model is present, the percentage value refers to the axis ramp-up time configured in WorkVisual.

Writability

The system variable is write-protected.

Syntax

acceleration = $ACC_EXTAX_C[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: external axis E1 ... E6
acceleration	Type: INT; unit: % • **1 … 100**

3.15 $ACC_OV

Description

Acceleration of the TCP in the event of a change to the program override (POV)

The variable of structure type CP defines the Cartesian acceleration in the event of a change to the program override (POV).

The aggregate consists of the following components:

- CP: Path acceleration in [m/s^2]
- ORI1: Swivel acceleration in [°/s^2]
- ORI2: Rotational acceleration in [°/s^2]

Example

```
$ACC_OV={CP 4.6,ORI1 200.0,ORI2 200.0}
```

3.16 $ACCU_STATE

Description

Result of the battery test

The variable can be used to display the result of the battery test or the result of monitoring of the charging current.

Writability

The system variable is write-protected.

Syntax

result = $ACCU_STATE

Explanation of the syntax

Element	Description
result	Type: ENUM ACCU_STATE • #CHARGE_OK: The battery test was positive. • #CHARGE_OK_LOW: The battery test was positive but the battery was still not fully charged after the maximum charging time. • #CHARGE_UNKNOWN: The battery is being charged but the charging current has not yet dropped sufficiently. The battery test has not yet been carried out. • #CHARGE_TEST_NOK: The battery test was negative. • #CHARGE_NOK: A battery test is not possible. The battery was still not fully charged after the maximum charging time. • #CHARGE_OFF: No charging current. Either there is no battery present or the battery is defective.

3.17 $ACT_ADVANCE

Description

Number of motion blocks currently planned in the main run

The maximum possible number of planned motion blocks depends on $ADVANCE.

(>>> *3.22 "$ADVANCE" Page 25*)

Writability

The system variable is write-protected.

Syntax

number = `$ACT_ADVANCE`

Explanation of the syntax

Element	Description
Number	Type: INT • **0 … $ADVANCE+1**

3.18 $ACT_BASE

Description

Number of the current BASE coordinate system in the advance run

Syntax

number = `$ACT_BASE`

Explanation of the syntax

Element	Description
Number	Type: INT **Note:** When selecting or resetting a robot program, the variable is set to the value -1.

3.19 $ACT_BASE_C

Description

Number of the current BASE coordinate system in the main run

When selecting or resetting a robot program, the variable is set to the value -1.

Writability

The system variable is write-protected.

Syntax

number = `$ACT_BASE_C`

Explanation of the syntax

Element	Description
Number	Type: INT

3.20 $ACT_TOOL

Description

Number of the current TOOL coordinate system in the advance run

Syntax

number = `$ACT_TOOL`

Explanation of the syntax

Element	Description
Number	Type: INT **Note:** When selecting or resetting a robot program, the variable is set to the value -1.

3.21 $ACT_TOOL_C

Description

Number of the current TOOL coordinate system in the main run

When selecting or resetting a robot program, the variable is set to the value -1.

Writability

The system variable is write-protected.

Syntax

number = $ACT_TOOL_C

Explanation of the syntax

Element	Description
Number	Type: INT

3.22 $ADVANCE

Description

Maximum number of motion instructions in the advance run

The variable is used to define the maximum number of motion instructions that the robot controller can calculate and plan in advance. The actual number of motion instructions calculated in advance is dependent on the capacity of the computer.

The advance run refers to the current position of the block pointer. The advance run is required, for example, in order to be able to calculate approximate positioning motions.

Syntax

$ADVANCE = *number*

Explanation of the syntax

Element	Description
Number	Type: INT • **1 … 5** Default: **3**

3.23 $ALARM_STOP

Description

Signal declaration for a digital output

- **FALSE**: The safety controller has triggered a safety STOP 0 or safety STOP 1, e.g. due to an EMERGENCY STOP.
- **TRUE**: The safety controller has possibly triggered a safety stop.
  No EMERGENCY STOP is active.

The output is set to FALSE in the following EMERGENCY STOP situations:

- EMERGENCY STOP via the EMERGENCY STOP device on the smartPAD
- EMERGENCY STOP via an external EMERGENCY STOP device
- Only for KR C5: EMERGENCY STOP via the EMERGENCY STOP device on the cabinet door (if present)

$ALARM_STOP_INTERN indicates whether an EMERGENCY STOP has been triggered internally or externally.
(>>> *3.24 "$ALARM_STOP_INTERN" Page 26*)

Syntax

```
SIGNAL $ALARM_STOP $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1013**

3.24 $ALARM_STOP_INTERN

Description

Signal declaration for the internal EMERGENCY STOP

- **FALSE**: An internal EMERGENCY STOP is active.
- **TRUE**: No EMERGENCY STOP or external EMERGENCY STOP is active.

Internal EMERGENCY STOP means:

- EMERGENCY STOP via the EMERGENCY STOP device on the smartPAD
- Only for KR C5: EMERGENCY STOP via the EMERGENCY STOP device on the cabinet door (if present)

Syntax

```
SIGNAL $ALARM_STOP_INTERN $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1002**

3.25 $ANIN

Description

Voltage at the analog inputs

The variable indicates the input voltage, standardized to a range between -1.0 and +1.0. The actual voltage depends on the device settings of the relevant analog module (scaling factor).

Writability

The system variable is write-protected.

Syntax

```
voltage = $ANIN[number]
```

Explanation of the syntax

Element	Description
Number	Type: INT Input number • **1 … 32**
voltage	Type: REAL • **-1.0 … +1.0**

3.26 $ANOUT

Description

Voltage at the analog outputs

The variable can be used to set an analog voltage limited to values between -1.0 and +1.0. The actual voltage generated depends on the analog module used (scaling factor).

If an attempt is made to set voltages outside the valid range of values, the message *Limit {Signal name}* is displayed.

Syntax

`$ANOUT[`*number*`] = `*voltage*

Explanation of the syntax

Element	Description
Number	Type: INT Output number • **1 … 32**
voltage	Type: REAL • **-1.0 … +1.0**

3.27 $APO

Description

Approximation parameters in the advance run

This variable is used to define the approximation distance.

Syntax

`$APO = {CVEL` *velocity*`, CPTP` *aproxDistPTP*`, CDIS` *aproxDistCP*`, CORI` *orientation*`}`

Explanation of the syntax

Element	Description
CVEL	Type: INT; unit: % Velocity parameter • **1 … 100** The approximation parameter specifies the percentage of the programmed velocity at which the approximate positioning process is started, at the earliest, in the deceleration phase towards the end point.
CPTP	Type: INT; unit: % Approximation distance for PTP and PTP spline motions (= furthest distance before the end point at which approximate positioning can begin) • **1 … 100** Explanation of the approximation parameter: (>>> *"CPTP" Page 28*) **Note**: PTP spline motions (SPTP) can be programmed in KUKA System Software 8.3 or higher.
CDIS	Type: REAL; unit: mm distance parameter Approximation starts, at the earliest, when the distance to the end point falls below the value specified here.
CORI	Type: REAL; unit: ° Orientation parameter Approximation starts, at the earliest, when the dominant orientation angle (rotation or swiveling of the longitudinal axis of the tool) falls below the angle distance to the end point specified here.

CPTP

The approximation parameter CPTP has a different effect depending on whether a PTP or a PTP spline motion (SPTP) is programmed.

- In the case of a PTP motion, the percentage value specified for CPTP refers to an axis angle defined by $APO_DIS_PTP in the machine data. As soon as the axis angle of all axes has fallen below the approximation distance thus defined, approximate positioning is carried out.

  The approximate positioning, however, is not started until 50% of the block length has been reached. This means that approximation starts, at the earliest, when half the distance between the start point and the end point relative to the contour of the PTP motion without approximation has been covered.
- This 50% limitation also applies to approximate positioning between 2 individual SPTP motions. In the case of approximate positioning between PTP splines that are programmed as one of several segments in spline blocks, the earliest point at which approximate positioning may be started is not defined. The approximate positioning starts as defined by CPTP.
- In the case of approximate positioning between PTP splines, the percentage value specified by CPTP refers to the sum of the following distances:

– Distance of the last spline segment in the first spline block
– Distance of the first spline segment in the subsequent spline block covered by all robot axes and mathematically coupled external axes in the axis space.

3.28 $APO_C

Description

Approximation parameters in the main run
The variable contains the current approximation distance.

Writability

The system variable is write-protected.

Syntax

`$APO_C = {CVEL` *velocity*`, CPTP` *aproxDistPTP*`, CDIS` *aproxDistCP*`, CORI` *orientation*`}`

Explanation of the syntax

Element	Description
CVEL	Type: INT; unit: % Velocity parameter • **1 … 100** The approximation parameter specifies the percentage of the programmed velocity at which the approximate positioning process is started, at the earliest, in the deceleration phase towards the end point.
CPTP	Type: INT; unit: % Approximation distance for PTP and PTP spline motions (= furthest distance before the end point at which approximate positioning can begin) • **1 … 100** Explanation of the approximation parameter: (>>> *"CPTP" Page 30*) **Note**: PTP spline motions (SPTP) can be programmed in KUKA System Software 8.3 or higher.
CDIS	Type: REAL; unit: mm distance parameter Approximation starts, at the earliest, when the distance to the end point falls below the value specified here.
CORI	Type: REAL; unit: ° Orientation parameter Approximation starts, at the earliest, when the dominant orientation angle (rotation or swiveling of the longitudinal axis of the tool) falls below the angle distance to the end point specified here.

CPTP

The approximation parameter CPTP has a different effect depending on whether a PTP or a PTP spline motion (SPTP) is programmed.

- In the case of a PTP motion, the percentage value specified for CPTP refers to an axis angle defined by $APO_DIS_PTP in the machine data. As soon as the axis angle of all axes has fallen below the approximation distance thus defined, approximate positioning is carried out.

  The approximate positioning, however, is not started until 50% of the block length has been reached. This means that approximation starts, at the earliest, when half the distance between the start point and the end point relative to the contour of the PTP motion without approximation has been covered.
- This 50% limitation also applies to approximate positioning between 2 individual SPTP motions. In the case of approximate positioning between PTP splines that are programmed as one of several segments in spline blocks, the earliest point at which approximate positioning may be started is not defined. The approximate positioning starts as defined by CPTP.
- In the case of approximate positioning between PTP splines, the percentage value specified by CPTP refers to the sum of the following distances:
  - Distance of the last spline segment in the first spline block
  - Distance of the first spline segment in the subsequent spline block covered by all robot axes and mathematically coupled external axes in the axis space.

3.29 $AUT

Description

Signal declaration for Automatic mode

This output is set when Automatic mode is selected.

Syntax

```
SIGNAL $AUT $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **995**

3.30 $AUX_POWER

Description

Signal declaration for external power supply

This input is set when the external power supply is active.

Syntax

```
SIGNAL $AUX_POWER $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

3.31 $AXIS_ACT

Description

Current axis-specific setpoint position of the robot

The variable of structure type E6AXIS contains the current axis angles or axis positions.

- **A1 … A6**: Setpoint position of the robot axes in [°] or [mm]
- **E1 … E6**: Setpoint position of the external axes in [°] or [mm]

In the robot program, the variable triggers an advance run stop.

Example

```
$AXIS_ACT={A1 0.0,A2 -90.0,A3 90.0,A4 0.0,A5 0.0,A6 0.0,E1
250.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}
```

3.32 $AXIS_ACT_MEAS

Description

Current axis-specific actual position of the robot

The variable of structure type E6AXIS contains the current axis angles or axis positions.

- **A1 … A6**: Actual position of the robot axes in [°] or [mm]
- **E1 … E6**: Actual position of the external axes in [°] or [mm]

Unlike $AXIS_ACT, which contains the setpoint positions, this variable always delivers the current actual axis angles of the drive.

3.33 $AXIS_BACK

Description

Axis-specific start position of the current motion block

The variable of structure type E6AXIS contains the axis angles or axis positions at the start position.

- **A1 … A6**: Axis position of the robot axes in [°] or [mm]
- **E1 … E6**: Axis position of the external axes in [°] or [mm]

$AXIS_BACK can be used to execute a PTP motion to return to the start position of an interrupted motion instruction. $AXIS_BACK corresponds to the beginning of the window for an interruption within the approximation window and to the end of the window for an interruption after the approximation window.

The variable is write-protected. In the robot program, the variable triggers an advance run stop.

Example

Approximated PTP motion

```
PTP P1
PTP P2 C_PTP
PTP P3
```

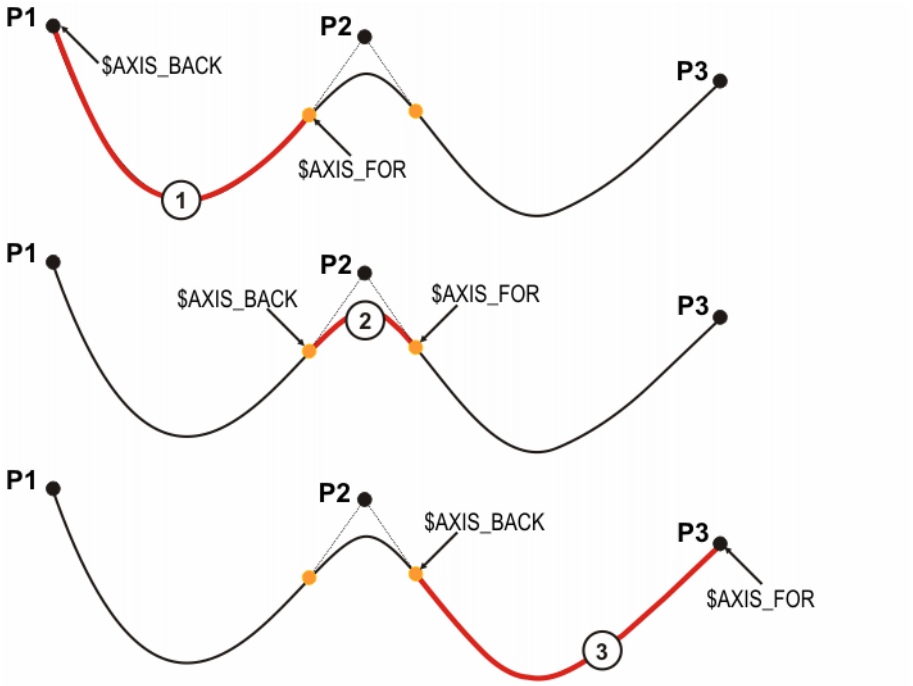


Fig. 3-1: $AXIS_BACK, $AXIS_FOR – P2 is approximated

1 Single block
2 Intermediate block
3 Following block

3.34 $AXIS_FOR

Description

Axis-specific target position of the current motion block

The variable of structure type E6AXIS contains the axis angles or axis positions at the target position.

- **A1 … A6**: Axis position of the robot axes in [°] or [mm]
- **E1 … E6**: Axis position of the external axes in [°] or [mm]

$AXIS_FOR can be used to execute a PTP motion to the target position of an interrupted motion instruction. $AXIS_FOR corresponds to the end of the window for an interruption within the approximation window and to the beginning of the window for an interruption before the approximation window.

The variable is write-protected. In the robot program, the variable triggers an advance run stop.

Example

(>>> *3.33 "$AXIS_BACK" Page 31*)

3.35 $AXIS_HOME

Description

HOME position $AXIS_HOME[1] … $AXIS_HOME[5]

With this variable, it is possible to define 5 other HOME positions in addition to the HOME position, which is defined with $H_POS.

Data type: E6AXIS

- A1 ... A6: angular values in [°] or translation values in [mm]
- E1 ... E6: angular values in [°] or translation values in [mm]

Example

```
$AXIS_HOME[1]={A1 0.0,A2 -90.0,A3 90.0,A4 0.0,A5 0.0,A6
0.0,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}
```

3.36 $AXIS_INT

Description

Axis-specific robot position in the case of an interrupt

The variable of structure type E6AXIS contains the axis angles or axis positions at the time of the interrupt.

- **A1 … A6**: Axis position of the robot axes in [°] or [mm]
- **E1 … E6**: Axis position of the external axes in [°] or [mm]

$AXIS_INT can be used to return to the axis-specific position at which an interrupt was triggered by means of a PTP motion.

The variable is write-protected and is only admissible in an interrupt program. In the interrupt program, the variable triggers an advance run stop.

3.37 $AXIS_MOT

Description

Current motor-specific robot position

The variable of structure type E6AXIS contains the current motor axis positions.

- **A1 … A6**: Motor angle of the robot axes in [°]
- **E1 … E6**: Motor angle of the external axes in [°]

3.38 $AXIS_RET

Description

Axis-specific robot position when leaving the path

The variable of structure type E6AXIS contains the axis angles or axis positions at the time that the programmed path was left.

- **A1 … A6**: Axis position of the robot axes in [°] or [mm]
- **E1 … E6**: Axis position of the external axes in [°] or [mm]

When the robot is stationary, $AXIS_RET can be used to return to the axis-specific position at which the path was left by means of a PTP motion.

Writability

The system variable is write-protected.

3.39 $AXWORKSTATE

Description

Signal declaration for monitoring of axis-specific workspaces

Each configured workspace must be assigned to a signal output. The output is set if an axis-specific workspace is violated.

	Further information about configuring workspaces is contained in the Operating and Programming Instructions for System Integrators.

Syntax

`SIGNAL $AXWORKSTATE`*Index* `$OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Index	Type: INT Number of workspace • **1 … 8**
Output number	Type: INT By default, the output is deactivated with **FALSE**.

3.40 $B_IN

Description

Value of a binary input

Syntax

`$B_IN[`*Input number*`]=`*Value*

Explanation of the syntax

Element	Description
Input number	Type: INT • **1 … 64**
Value	Type: INT The range of values depends on the configuration of the binary input $BIN_IN[*x*]. (>>> *3.45 "$BIN_IN" Page 36*)

3.41 $B_OUT

Description

Value of a binary output

Syntax

`$B_OUT[`*Output number*`]=`*Value*

Explanation of the syntax

Element	Description
Output number	Type: INT • **1 … 64**
Value	Type: INT The range of values depends on the configuration of the binary output $BIN_OUT[*x*]. (>>> *3.46 "$BIN_OUT" Page 37*)

Example

Configuration of a binary output in $CUSTOM.DAT:

```
$BIN_OUT[3] = {F_BIT 3, LEN 5, PARITY #EVEN}
```

This example configuration can be used to write values with a bit width of 5, starting from bit 3, with even parity.

Element	Description
F_BIT	Type: INT First bit: number of the first bit for which values can be set
LEN	Type: INT Bit width: number of bits for the values to be set
PARITY	Type: ENUM BIN_PARITY_T Parity bit • #NONE: no parity • #EVEN: even parity • #ODD: odd parity

3.42 $BASE

Description

BASE coordinate system in the advance run

The variable of structure type FRAME defines the setpoint position of the workpiece in relation to the WORLD coordinate system.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

3.43 $BASE_C

Description

BASE coordinate system in the main run

The variable of structure type FRAME defines the current actual position of the workpiece in relation to the WORLD coordinate system.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

Writability

The system variable is write-protected.

3.44 $BASE_KIN

Description

Information about the external BASE kinematic system

The variable contains the name of the external kinematic system and a list of the external axes contained in the transformation. The name and the external axes contained in the transformation are defined in the machine data, e.g. $ET1_NAME and $ET1_AX.

Writability

The system variable is write-protected.

Syntax

Information = $BASE_KIN[]

Explanation of the syntax

Element	Description
Information	Type: CHAR Name and external axes of the transformation: max. 29 characters

3.45 $BIN_IN

Description

Configuration of binary inputs

Syntax

$BIN_IN[*Input number*]={F_BIT *Start bit*, LEN *Bit width*, PARITY *Type*}

Explanation of the syntax

Element	Description
Input number	Type: INT • **1 … 64**
Start bit	Type: INT Maximum possible input ranges: • **1 … 4,096** • **1 … 8,192** **Note:** The number of inputs can be requested using the variable $NUM_IN. Default: **1**

Element	Description
Bit width	Type: INT • **0 … 31** Default: **0**
Type	Type: ENUM BIN_PARITY_T • #NONE: parity bit is not activated. • #EVEN: parity bit is activated. – If the parity sum is even, the parity bit has the value 0. – If the parity sum is odd, the parity bit has the value 1. • #ODD: parity bit is activated. – If the parity sum is odd, the parity bit has the value 0. – If the parity sum is even, the parity bit has the value 1. Default: #NONE

3.46 $BIN_OUT

Description

Configuration of binary outputs

The variable $B_OUT is used for assigning values to the binary outputs .

Syntax

`$BIN_OUT[`*Output number*`]={F_BIT` *Start bit*`, LEN` *Bit width*`, PARITY` *Type*`}`

Explanation of the syntax

Element	Description
Output number	Type: INT • **1 … 64**
Start bit	Type: INT Maximum possible output ranges: • **1 … 4,096** • **1 … 8,192** **Note:** The number of outputs can be requested using the variable $NUM_OUT. Default: **1**
Bit width	Type: INT • **0 … 31** Default: **0**

Element	Description
Type	Type: ENUM BIN_PARITY_T • #NONE: parity bit is not activated. • #EVEN: parity bit is activated. – If the parity sum is even, the parity bit has the value 0. – If the parity sum is odd, the parity bit has the value 1. • #ODD: parity bit is activated. – If the parity sum is odd, the parity bit has the value 0. – If the parity sum is even, the parity bit has the value 1. Default: #NONE

3.47 $BOOTTYPE

Description

Most recent system boot type

The system variable is overwritten by the subsequent state. Deletion or resetting is not possible.

Writability

The system variable is write-protected.

Syntax

type = `$BOOTTYPE`

Explanation of the syntax

Element	Description
type	Type: ENUM BOOTTYPE_T • #INVALID: an error occurred during the boot process. • #HIBERNATE: controller was booted from Hibernate mode. • #INITCOLDSTART: controller was booted with **Cold start** and **Reload files**. • #COLDSTART: controller was booted with **Cold start** without **Reload files** (quick start). • #RECONFIG: controller was reconfigured.

3.48 $BRAKE_SIG

Description

Bit array for reading the brake signals

The variable can be used to display the state of the axis brakes (open or closed).

Syntax

`$BRAKE_SIG=`*Bit array*

Explanation of the syntax

Element	Description
Bit array	• **Bit n = 0**: Brake is closed. • **Bit n = 1**: Brake is open.

Bit n	11 …	5	4	3	2	1	0
Axis	E6 …	A6	A5	A4	A3	A2	A1

Example

```
$BRAKE_SIG='B1000000'
```

The brakes of robot axes A1 to A6 are closed. The brake of external axis E1 is open.

3.49 $CHCK_MOVENA

Description

Check the input number of $MOVE_ENABLE in Automatic External mode

The variable $MOVE_ENABLE is used by the higher-level controller to check the robot drives. A precondition is that a suitable input number (<> $IN[1025]) has been assigned to the signal $MOVE_ENABLE. This is checked with $CHCK_MOVENA.

Background: by default, $IN[1025] is a system input which is always **TRUE**. Consequently, this input must not be allocated to the system variable $MOVE_ENABLE as this would mean that the motion enable would be issued on a permanent basis.

Syntax

`$CHCK_MOVENA=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: $MOVE_ENABLE is checked. • FALSE: $MOVE_ENABLE is not checked. Default: TRUE

3.50 $CIRC_MODE

Description

Behavior of the orientation control and external axis guidance at the auxiliary point and end point of a SCIRC circle

During SCIRC motions, the robot controller can take the programmed orientation of the auxiliary point into consideration. $CIRC_MODE can be used to define whether and to what extent it is taken into consideration.

In the case of SCIRC statements with circular angles, $CIRC_MODE can also be used to define whether the end point is to have the programmed orientation or whether the orientation is to be scaled according to the circular angle.

$CIRC_MODE can only be written to by means of an SCIRC statement. $CIRC_MODE cannot be read.

Syntax

Orientation of the auxiliary point:

`$CIRC_MODE.AUX_PT.ORI =` *auxPointValue*

Orientation of the end point:

`$CIRC_MODE.TARGET_PT.ORI =` *targetPointValue*

Explanation of the syntax

Element	Description
auxPointValue	Type: ENUM AUX_PT_VALUES Orientation of the auxiliary point • #INTERPOLATE: the programmed orientation is accepted at the auxiliary point. • #IGNORE: the transition from the start orientation to the end orientation is carried out over the shortest possible distance. The programmed orientation of the auxiliary point is disregarded. • #CONSIDER: the transition from the start orientation to the end orientation passes through the programmed orientation of the auxiliary point. This means the orientation of the auxiliary point is given at some point during the transition, but not necessarily at the auxiliary point. • #INVALID: invalid orientation Default: #CONSIDER
targetPointValue	Type: ENUM TARGET_PT_VALUES Orientation of the end point • #INTERPOLATE: the programmed orientation of the end point is accepted at the actual end point. (Only possibility for SCIRC without specification of circular angle. If #EXTRAPOLATE is set, #INTERPOLATE is nonetheless executed.) • #EXTRAPOLATE: the programmed orientation is accepted at the programmed end point. The orientation at the actual end point is scaled according to the circular angle. • #INVALID: invalid orientation Default for SCIRC with specification of circular angle: #EXTRAPOLATE

Constraints

- If $ORI_TYPE = #IGNORE for a SCIRC segment, $CIRC_MODE is not evaluated.

- If a SCIRC segment is preceded by a SCIRC or SLIN segment with $ORI_TYPE = #IGNORE, #CONSIDER cannot be used in this SCIRC segment.

For SCIRC with circular angle:

- #INTERPOLATE must not be set for the auxiliary point.
- If $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.
- If it is preceded by a spline segment with $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.

Example: auxiliary point

The TCP executes an arc with a Cartesian angle of 192°:

- The orientation at the start point is 0°.
- The orientation at the auxiliary point is 98°.
- The orientation at the end point is 197°.

The re-orientation is thus 197° if the auxiliary point is taken into consideration.

If the orientation at the auxiliary point is ignored, the end orientation can also be achieved by means of a re-orientation of 360° - 197° = 163°.

- #INTERPOLATE:

  The programmed orientation of 98° is accepted at the auxiliary point. The re-orientation is thus 197°.
- #IGNORE:

  The programmed orientation of the auxiliary point is disregarded. The shorter re-orientation through 163° is used.
- #CONSIDER:

  The route traveled includes the orientation of the auxiliary point, in this case the re-orientation through 197°. This means the orientation of 98° is accepted at some point during the transition, but not necessarily at the auxiliary point.

Example: end point

The example schematically illustrates the behavior of #INTERPOLATE and #EXTRAPOLATE.

- The pale, dotted arrows show the programmed orientation.
- The dark arrows show the actual orientation where this differs from the programmed orientation.

#INTERPOLATE:

At **TP**, which is situated before **TP_CA**, the programmed orientation has not yet been reached. The programmed orientation is accepted at **TP_CA**.

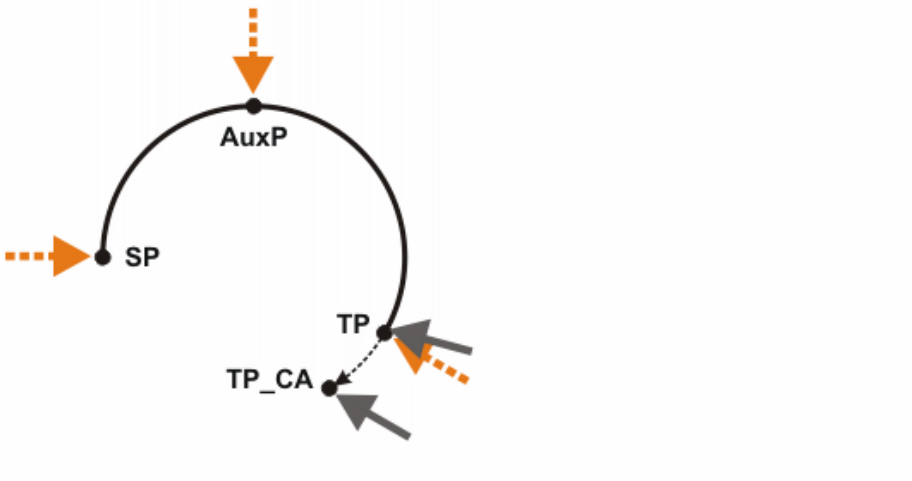


Fig. 3-2: #INTERPOLATE

SP	Start point
AuxP	Auxiliary point
TP	Programmed end point
TP_CA	Actual end point. Determined by the circular angle.

#EXTRAPOLATE:

The programmed orientation is accepted at **TP**. For **TP_CA**, this orientation is scaled in accordance with the circular angle.

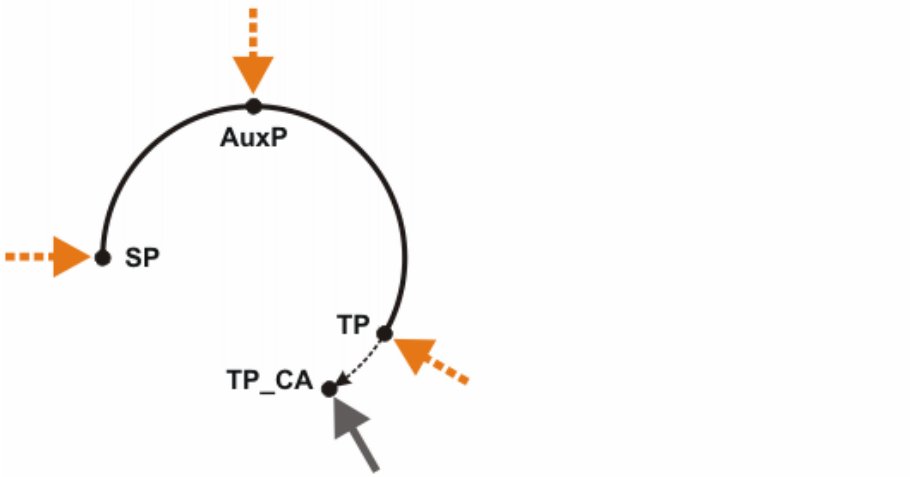


Fig. 3-3: #EXTRAPOLATE

3.51 $CIRC_TYPE

Description

Orientation control of CIRC in the advance run

The variable contains the programmed orientation control of a circular motion. This can be base-related or path-related.

Syntax

`$CIRC_TYPE = type`

Explanation of the syntax

Element	Description
type	Type: ENUM CIRC_TYPE • #BASE: base-related orientation control • #PATH: path-related orientation control

3.52 $CIRC_TYPE_C

Description

Orientation control of CIRC in the main run

The variable contains the orientation control of the circular motion currently being executed. This can be base-related or path-related.

Writability

The system variable is write-protected.

Syntax

type = `$CIRC_TYPE_C`

Explanation of the syntax

Element	Description
type	Type: ENUM CIRC_TYPE • #BASE: base-related orientation control • #PATH: path-related orientation control

3.53 $CMD

Description

Management number (handle) for command channel $CMD

The CWRITE() function can be used to write statements to the $CMD command channel.

Detailed information on the CWRITE() command can be found in the CREAD/CWRITE documentation .

Writability

The system variable is write-protected.

3.54 $COLLMON_STARTUP_MAX: adapting the inactive collision detection phase after start of motion

Description

When starting to move, the axes must apply torques that cannot be safely predicted due to static friction effects. To prevent the collision detection function from evaluating these torques as a collision, it is automatically deactivated for the motion start. At the end of the starting phase, collision detection is automatically reactivated.

The robot controller considers the starting phase to be completed when a specific velocity, namely the standstill monitoring limit, is exceeded. If the robot moves very slowly, the axes might not reach this limit. The motion is thus still regarded as the starting phase. Collision detection remains inactive.

$COLLMON_STARTUP_MAX limits this inactive collision detection phase at the start of motion.

Precondition

$COLLMON_STARTUP_MAX is only relevant if the following preconditions are met:

- $IMPROVED_COLLMON = TRUE
- Collision detection is activated.

Writability

The variable can be modified in $custom.dat in KRC:\STEU\Mada.

It cannot be modified via the variable display function or program.

Syntax

`$COLLMON_STARTUP_MAX` = *time*

Explanation of the syntax

Element	Description
time	Type: INT Maximum duration (ms) of the inactive collision detection phase at the start of motion. Collision detection is then reactivated, even if the axes are still in the starting phase. • **≥ 50**: Value is used as entered. • **1 … 49**: Value is implicitly increased to 50. • **≤ 0**: No limitation of the inactive phase. This means that collision detection remains inactive as long as the starting phase is in progress, continuously if necessary. Default: 200

At very low axis velocities, collision detection may generally react very sensitively. If collision detection is now reactivated after a certain time due to $COLLMON_STARTUP_MAX (i.e. with any value > 0), it may be triggered undesirably.
Possible remedy:

- Set $COLLMON_STARTUP_MAX = 0.
- Or: Significantly increase the collision detection limit values for the axes concerned.

3.55 $COM_VAL_MI

Description

Limitation of the command rotational speed

This variable allows the axial command velocity to be limited to the percentage set here.

Syntax

```
$COM_VAL_MI[Axis number]=Limit value
```

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
Limit value	Type: REAL; unit: % Default: **150.0** **Note:** This value must not be changed.

3.56 $CONF_MESS

Description

Signal declaration for the external acknowledgement of messages

If the Automatic External interface is active ($I_O_ACT is TRUE), this input can be set by the higher-level controller to acknowledge an error message as soon as the cause of the error has been eliminated.

Only the rising edge of the signal is evaluated.

Syntax

```
SIGNAL $CONF_MESS $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

3.57 $CONSIDER_DLIN_ENERGY: Consideration of the crash energy in DLIN planning

Description

Activates consideration of the crash energy in DLIN planning.

A DLIN motion is a dynamic LIN motion. In other words, an attempt is made to achieve the specified velocity, acceleration and approximation distance. If these values cannot be reached, they are gradually reduced in the path planning. This may be the case on account of physical limitations (such as gear torque).

Consideration of the crash energy is very time-consuming and memory-intensive and should only be activated if crash energy monitoring is triggered.

Declaration

```
BOOL $CONSIDER_DLIN_ENERGY = FALSE
```

Explanation

Element	Description
$CONSIDER_DLIN_ENERGY	Data type: BOOL Activates consideration of the crash energy in DLIN planning. Default: FALSE

3.58 $COULD_START_MOTION

Description

Displays whether robot can be moved

$COULD_START_MOTION specifies whether a program start or jogging is possible. Possible means that no messages that disable active commands are still active.

Besides this, the safety-oriented signals are checked (e.g. motion enable, safety stop, operator safety, enable for jogging). If there are no inhibiting safety-oriented signals or messages still active, $COULD_START_MOTION becomes TRUE and the robot can be moved.

Writability

The system variable is write-protected.

Syntax

state = $COULD_START_MOTION

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: robot can be moved. • FALSE: robot cannot be moved.

3.59 $COUNT_I

Description

Freely usable variable for a counter

Syntax

$COUNT_I[*Index*]=*Number*

Explanation of the syntax

Element	Description
Index	Type: INT Counter number • **1 … 64**
Number	Type: INT

3.60 $CP_STATMON

Description

The user can activate a Status/Turn check using $CP_STATMON. This applies for the following motions:

- LIN, CIRC
- Individual spline blocks
- Spline blocks

Either only the Status can be checked or both the Status and Turn.

A check is made to see if the TCP at the end point has reached the taught or programmed Status/Turn. If not, the robot stops at the end point and the robot controller generates the message *$CP_STATMON: incorrect axis angle*. The program can be resumed once the message has been acknowledged.

If the end point is approximated during a spline motion, the robot controller checks during planning whether the Status/Turn would be correctly approached if the end point were an exact positioning point. If this would not be the case, then a message is displayed: *$CP_STATMON: approximate positioning not possible*. The robot moves exactly to the point, stops there and the message *$CP_STATMON: incorrect axis angle* is generated.

Following a block selection into a spline block, a check is carried out after the BCO run to see whether the end point of the block (not the end point of the BCO run) is reached with the correct Status/Turn. If not, the robot stops and the message *$CP_STATMON: incorrect axis angle* is generated. If the message is acknowledged and the program is resumed, the robot stops again at the end point of the block and the message is generated again.

No check is carried out in the following cases, even if a check has been activated:

- For CIRC motions with a circular angle specification
- If $ORI_TYPE == #JOINT, the Status of the wrist axes is not checked.
- If no Status and/or Turn has been programmed for a programmed point, the relevant component is not checked.
- If the BASE coordinate system is unknown at the time of planning (e.g. if the BASE is a conveyor belt), the Turn is not checked.
- For LIN and CIRC, the Turn is not checked if the axis angles at the end point are between -2° and +2°.

Writability

The system variable can be modified via the variable display function or in $custom.dat in KRC:\STEU\Mada.

It cannot be modified via the program.

Syntax

`$CP_STATMON =` *state*

Explanation of the syntax

Element	Description
state	Type: ENUM CP_STATMON • #CHECK_S: Status is checked. • #CHECK_TS: Status and Turn are checked. • #NONE: the check is deactivated. Default: #NONE

3.61 $CP_VEL_TYPE: Reduction of the axis velocity for CP motions

Description

The variable causes an automatic reduction of the axis velocity in certain situations. The reduction is particularly effective in the vicinity of singularities and generally allows the robot to pass through the singularity position. $CP_VEL_TYPE can be written from the robot interpreter.

If $CP_VEL_TYPE is modified via the robot interpreter, the value in the file $CUSTOM.DAT is also modified. After a cold start, the robot controller boots with the new setting.

Syntax

`$CP_VEL_TYPE` = *type*

Explanation of the syntax

Element	Description
type	Type: ENUM CP_VEL_TYPE • #CONSTANT: no reduction of axis velocity • #VAR_ALL: reduction of axis velocity in all modes • #VAR_ALL_MODEL: model-based reduction of axis velocity in program mode • #VAR_T1: reduction of axis velocity in T1 **Note:** Reduction of axis velocity is always active in Cartesian jogging.

3.62 $CP_VEL_TYPE_AX_RED[]: Reduction of the maximum axis velocity

Description

In program mode, the axis velocity can only be adjusted if $CP_VEL_TYPE is active (>>> *3.61 "$CP_VEL_TYPE: Reduction of the axis velocity for CP motions" Page 48*). Here, the velocity is not limited generally, but rather to the maximum axis velocity.
$CP_VEL_TYPE_AX_RED[] can be used to reduce this maximum axis velocity in all operating modes. If the program is reset, the system variable is reset to its default value.

Declaration

```
INT $CP_VEL_TYPE_AX_RED[6]
```

Explanation

Element	Description
$CP_VEL_TYPE_AX_RED[i]	Data type: INT Reduces the velocity of the desired robot axis [%] • **1 ... 100** • **i**: Robot axis (1-6) Default: 100

3.63 $CUR_MEMORY

Description

Current memory allocation of the real-time system

Writability

The system variable is write-protected.

Syntax

memoryinfo = $CUR_MEMORY

Explanation of the syntax

Element	Description
memoryinfo	Type: STRUC MEMORY_INFO • SYSTEM_USED: memory occupied by the system in bytes • SYSTEM_RESERVED: memory reserved by the system in bytes • USER_USED: memory occupied by the user in bytes • FREE: memory available to the user in bytes

3.64 $CURR_ACT

Description

Actual current of axes

The variable contains the actual current as a percentage of the maximum amplifier or motor current. The actual current always refers to the lower of the two maximum values.

Writability

The system variable is write-protected.

Syntax

current = $CURR_ACT[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
current	Type: REAL; unit: % • **-100.0 … +100.0**

3.65 $CYCFLAG

Description

Activation of cyclical flags

There are a total of 256 cyclical flags, 64 of which can be activated at the same time.

Cyclical evaluation of cyclical flags can be activated by assigning a Boolean expression in a robot program. Assignment of a Boolean expression in a submit program does not result in cyclical evaluation.

Syntax

$CYCFLAG[*Number*] =*Boolean expression*

Explanation of the syntax

Element	Description
Number	Type: INT • **1 … 256**
Boolean expression	Type: BOOL • Boolean expression: Cyclical flag is activated. • FALSE: Cyclical flag is deactivated. Default: FALSE

Example

Assignment of a Boolean expression in the robot program:

```
$CYCFLAG[15] = $IN[3] OR NOT $FLAG[7] AND (UserVar > 15)
```

The assignment causes the expression on the right-hand side to be evaluated cyclically in the background. This means as soon as the value of a sub-expression on the right-hand side changes, the value of $CYCFLAG also changes.

The Boolean expression assigned to the cyclical flag can be overwritten at any time by the robot program or by a trigger assignment. Cyclical processing is stopped as soon as the cyclical flag is assigned the value FALSE.

3.66 $DATA_EXT_OBJ1, $DATA_EXT_OBJ2

Description

Counter for data packets received via an external module of type LD_EXT_OBJ

The variable can be used to monitor whether data are available for reading.

Further information on use of the counter is contained in the CREAD/CWRITE documentation.

Writability

The system variable is write-protected.

Syntax

number = $DATA_LD_EXT_OBJ1

number = $DATA_LD_EXT_OBJ2

Explanation of the syntax

Element	Description
Number	Type: INT Number of data packets received via the channel

3.67 $DATA_INTEGRITY

Description

Check of data consistency for input and output signals

The variable is relevant when signals are transferred in groups; it has a different effect on inputs and ouputs:

- With inputs, it is ensured that the I/O map does not change when a signal is read.
- With outputs, it is checked whether a signal is mapped onto a single device I/O block.

Writability

The system variable is write-protected.

Syntax

state = $DATA_INTEGRITY

Explanation of the syntax

Element	Description
state	Type: BOOL Check of data consistency for input and output signals • TRUE: check is activated. • FALSE: check is not activated. Default: TRUE

3.68 $DATAPATH

Description

The variable display can be used to display and modify variables from KRL modules. By default, variables are only visible if they are declared as GLOBAL or declared in $CONFIG.DAT. To display non-global or local variables, $DATAPATH must be set to the module in which the variable is declared.

- $DATAPATH is automatically set to this module when a module is selected.
- $DATAPATH can be changed in the “Module” field of the variable display.

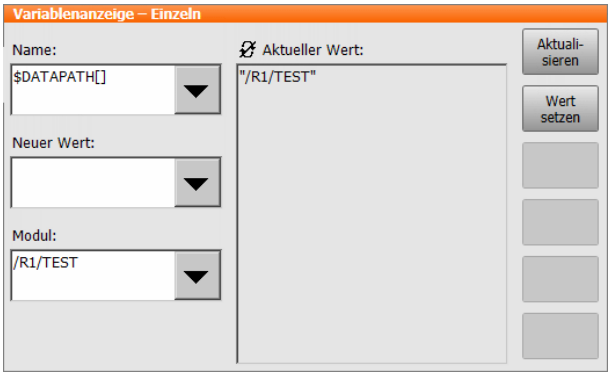


Fig. 3-4: Variable display

Syntax

`$DATAPATH[]="/R1/modulename"`

Explanation of the syntax

Element	Description
modulename	Type: CHAR[] Module name (max. 28 characters)

Example

TEST.DAT

```
; ...
INT DAT_VAR
GLOBAL INT GLO_VAR = 0
; ...
```

TEST.SRC

```
; ...
INT LOC_VAR
LOC_VAR = 0
; ...
```

$CONFIG.DAT

```
; ...
INT CONF_VAR = 0
; ...
```

If the data path is not set, it is only possible to display GLO_VAR and CONF_VAR in the variable display. If the data path is set to “/R1/TEST”, then DAT_VAR and LOC_VAR can also be displayed. The latter can only be displayed if the program pointer is positioned in the corresponding program.

3.69 $DATE

Description

Date and time of the real-time operating system (VxWorks)

Writability

The system variable is write-protected.

Syntax

currDate = $DATE

Explanation of the syntax

Element	Description
currDate	Type: STRUC DATE • CSEC: The value is always "0". • SEC • MIN • HOUR • DAY • MONTH • YEAR

Example

Generating a time stamp:

```
DECL STATE_T STAT
DECL INT offset
DECL CHAR timeStamp[100]

timeStamp[] = " "
offset = 0

SWrite(TimeStamp[], STAT, offset, "%i-%i-%i_%i-%i",
$DATE.YEAR, $DATE.MONTH, $DATE.DAY, $DATE.HOUR, $DATE.MIN)
```

3.70 $DEACTIVATE_ABS_ACCUR

Description

Deactivation of the positionally accurate robot model

The variable can be used to temporarily deactivate a positionally accurate robot model backed up on the RDC (only for test purposes).

Syntax

$DEACTIVATE_ABS_ACCUR = *state*

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: the positionally accurate robot model saved on the RDC is deactivated. After a cold start of the robot controller, the variable is automatically FALSE again. • FALSE: the positionally accurate robot model saved on the RDC is activated. Default: FALSE

3.71 $DELTA_WORKSPACE: Cartesian workspace for delta robots

Description

Defines a Cartesian workspace for delta robots within which the flange must remain. The configuration variable determines the dimensions of this space.

The workspace of the delta robot is defined relative to $ROBROOT and is described completely with the variables Z1 (Z coordinate of the first cylinder plane), Z2 (Z coordinate of the second cylinder plane and, at the same time, the first frustum plane), Z3 (Z coordinate of the second frustum plane), R1 (cylinder radius and, at the same time, the radius of the larger frustum plane) and R2 (radius of the smaller frustum plane). The cylinder and cone frustum are always rotationally symmetrical to the Z axis of the ROBROOT coordinate system.

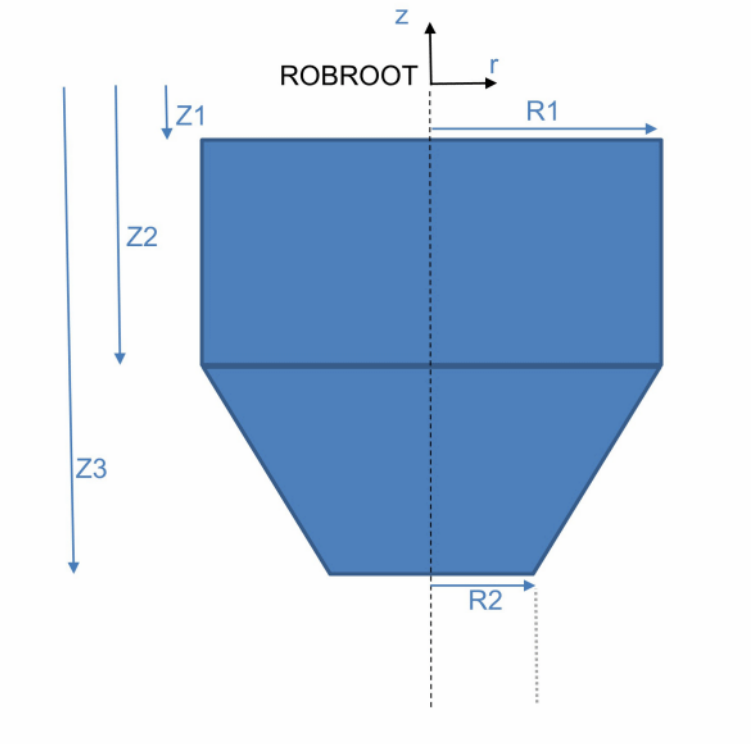


Fig. 3-5: Overview of Z and R

Declaration

```
DELTA_WORKSPACE $DELTA_WORKSPACE={Z1 0.0,Z2 0.0,Z3
0.0,R1 0.0,R2 0.0,MODE #OFF}
```

Explanation

Element	Description
$DELTA_WORKSPACE	Data type: DELTA_WORKSPACE Definition of the dimensions of a Cartesian workspace for delta robots

Data type DELTA_WORKSPACE

```
STRUC DELTA_WORKSPACE REAL Z1, Z2, Z3, R1, R2, SW_ONOFF
MODE
```

Element	Description
Z1	Data type: REAL Z coordinate of the first cylinder plane Default: 0.0
Z2	Data type: REAL Z coordinate of the second cylinder plane and, at the same time, the first frustum plane Default: 0.0
Z3	Data type: REAL Z coordinate of the second frustum plane Default: 0.0
R1	Data type: REAL Cylinder radius and, at the same time, radius of the larger frustum plane Default: 0.0
R2	Data type: REAL Radius of the smaller frustum plane Default: 0.0
MODE	Data type: BOOL Workspace monitoring active or not active Possible values: • #ON: Workspace monitoring active • #OFF: Workspace monitoring not active Default: #OFF

3.72 $DELTAWORKSTATE: Status of the workspace for delta robots

Description

Indicates whether the workspace (>>> *3.71 "$DELTA_WORKSPACE: Cartesian workspace for delta robots" Page 54*) is active and whether the flange position is outside the relevant area.

Declaration

```
SIGNAL $DELTAWORKSTATE $OUT[856]
```

Explanation

Element	Description
$DELTAWORKSTATE	Data type: BOOL Indicates whether a workspace is active and the flange position is outside the permissible area. • TRUE: Workspace monitoring ((>>> *3.71 "$DELTA_WORKSPACE: Cartesian workspace for delta robots" Page 54*)) is active and the flange position relative to $ROBROOT is outside the workspace. • FALSE: Monitoring is inactive or the flange position is inside the workspace.

3.73 $DEVICE

Description

Operating state of the connected teach pendant

Syntax

state = $DEVICE

Explanation of the syntax

Element	Description
state	Type: ENUM DEVICE • #ACTIVE: The teach pendant is active. • #BLOCK: The teach pendant is blocked, e.g. by error messages. • #PASSIVE: The teach pendant is passive, e.g. if the robot controller is operated by an external PLC in Automatic External mode. **Note:** In operating modes T1 and T2, the teach pendant is always #ACTIVE.

Example

```
DECL DEVICE state

state = $DEVICE

IF state == #ACTIVE THEN
 HALT
ENDIF
```

3.74 $DIR_CAL

Description

Mastering direction

During mastering, the reference point of an axis is addressed in the positive or negative direction.

Syntax

`$DIR_CAL=`*Bit array*

Explanation of the syntax

Element	Description
Bit array	Bit array that specifies the mastering direction for each axis • **Bit n = 0**: The reference point is approached in the positive direction. • **Bit n = 1**: The reference point is approached in the negative direction.

Bit n	11 …	5	4	3	2	1	0
Axis	E6 …	A6	A5	A4	A3	A2	A1

Example

```
$DIR_CAL='B010011'
```

The reference points of axes A3, A4 and A6 are approached in the positive direction. The reference points of axes A1, A2 and A5 are approached in the negative direction.

3.75 $DISTANCE

Description

Current path length relative to the most recent exact positioning point of the motion.

The variable can be used to evaluate a function relative to the path. At the start of a CP motion and a PTP-CP approximate positioning motion, the variable is set to the value zero.

Writability

The system variable is write-protected.

Syntax

path length = `$Distance`

Explanation of the syntax

Element	Description
path length	Type: REAL; unit: mm

3.76 $DIST_LAST

Description

$DIST_LAST specifies the length of the path from the current TCP position to the previous taught point. The value is generally positive.

Type: REAL Unit:

- For CP motions (spline and conventional): mm

- For SPTP motions: no unit

$DIST_LAST cannot be used for PTP motions. The value is always zero in this case.

Writability

The system variable is write-protected.

Syntax

Distance = $DIST_LAST

Explanation of the syntax

Element	Description
Distance	Type: REAL Length of the path from the current TCP position to the previous taught point Unit: • For CP motions (spline and conventional): mm • For SPTP motions: no unit The value is generally positive. $DIST_LAST cannot be used for PTP motions. The value is always zero in this case.

Programming a PATH trigger with the aid of $DIST_LAST

$DIST_LAST can be used as an aid for programming PATH triggers with ONSTART. It can be used to determine the value that must be assigned to the PATH parameter.

1. Move to the position on the path where the switching point is to be located.
2. Read the system variable.
3. Program the trigger after the previous point.
   - Program the trigger with ONSTART.
   - Assign the value of the system variable to the PATH parameter.

3.77 $DIST_NEXT

Description

$DIST_NEXT specifies the length of the path from the current TCP position to the next taught point.

Writability

The system variable is write-protected.

Syntax

Distance = $DIST_NEXT

Explanation of the syntax

Element	Description
Distance	Type: REAL Length of the path from the current TCP position to the next taught point Unit: • For CP motions (spline and conventional): mm • For SPTP motions: no unit The value is generally negative. $DIST_NEXT cannot be used for PTP motions. The value is always zero in this case.

Programming a PATH trigger with the aid of $DIST_NEXT

$DIST_NEXT can be used as an aid for programming PATH triggers without ONSTART. It can be used to determine the value that must be assigned to the PATH parameter.

1. Move to the position on the path where the switching point is to be located.
2. Read the system variable.
3. Program the trigger before the next point.
   - Program the trigger without ONSTART.
   - Assign the value of the system variable to the PATH parameter.

3.78 $DRIVES_ENABLE

Description

Switching drives on/off

Syntax

`$DRIVES_ENABLE = state`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: Switches the drives on. • FALSE: Switches the drives off.

3.79 $DRIVES_FANSPEED[]: Current fan speed of the inverters

Description

Enables reading of the current fan speed of the inverters. For a KSP300, two fans are always used, while three fans are used for a KSP600.

In the array, the fans are output one after the other as in the bus structure. The meaning of the index depends on the configuration of the drives.

Example with two external axes:

- $DRIVES_FAN_SPEED[1] = 1st fan of KSP600
- $DRIVES_FAN_SPEED[2] = 2nd fan of KSP600
- $DRIVES_FAN_SPEED[3] = 3rd fan of KSP600
- $DRIVES_FAN_SPEED[4] = 1st fan of first KSP600-3L
- $DRIVES_FAN_SPEED[5] = 2nd fan of first KSP600-3L
- $DRIVES_FAN_SPEED[6] = 3rd fan of first KSP600-3L
- $DRIVES_FAN_SPEED[7] = 1st fan of second KSP600-3L
- $DRIVES_FAN_SPEED[8] = 2nd fan of second KSP600-3L
- $DRIVES_FAN_SPEED[9] = 3rd fan of second KSP600-3L
- $DRIVES_FAN_SPEED[10-20] = EMPTY

Declaration

```
INT $DRIVES_FANSPEED[20]
```

Explanation

Element	Description
$DRIVES_FANSPEED[i]	Data type: INT Current fan speed of the inverters [rpm] • **i**: Fans, depending on the configuration of the drives

3.80 $DRIVES_OFF

Description

Signal declaration for switching off the drives

If there is a LOW-level pulse of at least 20 ms duration at this input, the higher-level controller switches off the robot drives.

Syntax

```
SIGNAL $DRIVES_OFF $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.81 $DRIVES_ON

Description

Signal declaration for switching on the drives

If there is a HIGH-level pulse of at least 20 ms duration at this input, the higher-level controller switches on the robot drives.

Syntax

```
SIGNAL $DRIVES_ON $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **140**

3.82 $ECO_LEVEL

Description

Energy saving mode

The system variable $ECO_LEVEL can be used to operate the robot in energy saving mode. The degree of energy saving can be set to “Low”, “Middle” or “High”. Energy saving mode causes the robot axes and external axes to move more slowly. The higher the saving, the lower the velocity. How much energy is saved relative to full power depends primarily on the axis positions and cannot be predicted.

$ECO_LEVEL does not affect all motions. The following table indicates which motions it affects and which it does not:

Motion	Effect?
PTP	Yes
LIN	No
CIRC	No
CP spline motions (block and individual motion) With higher motion profile	Yes
CP spline motions (block and individual motion) Without higher motion profile	No
PTP spline motions (block and individual motion)	Yes

If a program is reset or deselected, energy saving mode is automatically deactivated.

Energy saving mode is inactive in the following cases, even if it has been activated:

- In the case of a BCO run
- In a constant velocity range with spline
- In a time block with spline

If low values have already been programmed for acceleration and velocity, $ECO_LEVEL has little or no effect.

Depending on the robot type, the savings may be the same, or virtually the same, for both “Middle” and “High” (e.g. with a payload below 30% of the default payload).

Precondition

- $ADAP_ACC not equal to #NONE
- $OPT_MOVE not equal to #NONE

The default setting for both system variables is “not equal to #NONE”.

Syntax

`$ECO_LEVEL =` *level*

Explanation of the syntax

Element	Description
level	Type: ENUM ECO_LEVEL Degree of energy saving • #OFF: energy saving mode is deactivated. • #LOW: low saving • #MIDDLE: medium saving • #HIGH: high saving

3.83 $EMSTOP_PATH

Description

Path-maintaining EMERGENCY STOP for operating modes T1, T2, Automatic and Automatic External

Syntax

```
$EMSTOP_PATH = {T1 state, T2 state, AUT state, EX state}
```

Explanation of the syntax

Element	Description
state	Type: ENUM EMSTOP • #ON: path-maintaining EMERGENCY STOP is activated. • #OFF: path-maintaining EMERGENCY STOP is deactivated. Default: #ON

3.84 $EMSTOP_TIME

Description

Timeout monitoring for path-maintaining EMERGENCY STOP (STOP 1)

This variable defines a period of time after completion of interpolation in the event of a path-maintaining EMERGENCY STOP. After this period the drives are switched off.

Syntax

```
$EMSTOP_TIME=Time
```

Explanation of the syntax

Element	Description
Time	Type: INT; unit: ms Default: **100**

3.85 $ENERGY_CONFIG_STATE: status of the energy model for each axis

Description

$ENERGY_CONFIG_STATE[*axis*] = status

Data type: Energy_Config_State_T

$ENERGY_CONFIG_STATE provides the status of the energy model for each axis; in other words, the extent to which the consumption for the axis can be determined.

Energy_Config_State_T

```
ENUM Energy_Config_State_T OK, IRRELEVANT,
APPROXIMATED, IGNORED
```

The component is write-protected.

Element	Description
axis	Axis number Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
status	Type: ENUM Energy_Config_State_T • #OK: The consumption of this axis can be determined. Or: this axis is not present. • #IRRELEVANT: The axis is irrelevant for the determination of consumption because it is simulated. • #APPROXIMATED: The consumption is approximated for the axis and determined less accurately. • #IGNORED: Only relevant in the simulation case. Since certain model parameters for the simulation are not available for the axis, the consumption cannot be determined.

3.86 $ENERGY_INTERIM: energy consumption between time stamps

Description

$ENERGY_INTERIM displays the energy consumption of the robot and robot controller in the time between the two most recent calls of the INTERIMENERGY() function.

(>>> *3.86.1 "INTERIMENERGY: setting a time stamp for the measurement of energy consumption" Page 64*)

The initial value of $ENERGY_INTERIM in the case of cold start is 0. The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy,
CHAR info[]
```

The components are write-protected. Exception: `info[]` can be set using the INTERIMENERGY() function.

Element	Description
time	Duration of the measurement (s) Corresponds to the time that has elapsed between the two most recent calls of the INTERIMENERGY() function.
energy	Energy consumption during measurement period (kWh)
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)
info[]	The string that was transferred during the most recent call of the INTERIMENERGY() function

3.86.1 INTERIMENERGY: setting a time stamp for the measurement of energy consumption

Description

The INTERIMENERGY function sets up a time stamp. Energy consumption values are written to $ENERGY_INTERIM using the stamp (= function call). When $ENERGY_INTERIM is read out, you obtain the values incurred between the second-to-last and final stamp.

(>>> *3.86 "$ENERGY_INTERIM: energy consumption between time stamps" Page 63*)

Values from previous stamp intervals are overwritten. If these values are to be saved, the user must ensure that they are written to special-purpose variables prior to every new function call.

Syntax

result = `INTERIMENERGY` (*string*)

Explanation of the syntax

Element	Description
result	Type: INT Variable for the return value • 1: The function was called successfully. • <>1: Bug
string	Type: CHAR Type of transfer: IN parameter Any string. Serves as additional info on the respective function call.

Example

KRL program:

The function calls have been marked as “nr_1”, “nr_2”, etc. using the parameter *string* in this case.

```
1 ...
2 my_result = interimenergy ("nr_1")
3 ...
4 my_result = interimenergy ("nr_2")
5 ...
6 my_result = interimenergy ("nr_3")
7 ...
```

Reading out $ENERGY_INTERIM:

When $ENERGY_INTERIM is read out, you obtain the values incurred between the second-to-last and final stamp.

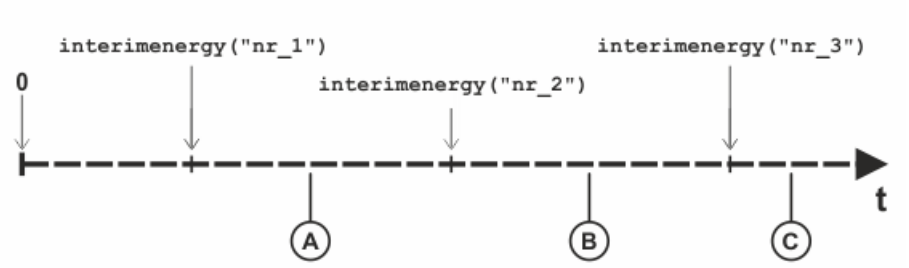


Fig. 3-6: Chronological sequence

Item	Description
0	Cold start
Reading out $ENERGY_INTERIM:	
A	Reading out between "nr_1" and "nr_2" displays the following consumption: • From cold start up to "nr_1"
B	Reading out between "nr_2" and "nr_3" displays the following consumption: • From "nr_1" to "nr_2"
C	Reading out after "nr_3" (and up to a potential "nr_4") displays the following consumption: • From "nr_2" to "nr_3"

$ENERGY_INTERIM also displays other values: Duration of measurement period, string from the most recently completed function call , unstored energy.

3.87 $ENERGY_MEASURING: starting and stopping consumption measurement

Description

The user can use $ENERGY_MEASURING to measure the energy consumption of the robot and robot controller. He can start and stop a measurement and read out the results.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Measuring_Struc

Energy_Measuring_Struc

```
STRUC Energy_Measuring_Struc BOOL active, locked, REAL
time, energy, lostenergy
```

The REAL components are write-protected.

Element	Description
active	• TRUE: Start measurement • FALSE (default): Stop measurement **Note:** A measurement that has been started via the smartHMI in the **Energy consumption** window cannot be stopped via KRL as long as the **Energy consumption** window remains open. If an attempt is made to stop the measurement via KRL, the robot controller displays the following message: *Energy measurement cannot currently be stopped*. Once the **Energy consumption** window has been closed again, the measurement can also be stopped via KRL. This prevents a measurement started in the **Energy consumption** window from permanently blocking measurements via KRL.
locked	• TRUE: Starting the measurement via the **Energy consumption** window is inhibited. Pressing the **Start measuring**button triggers the following message: *Energy measurement cannot currently be started.* Pressing **Stop measuring** or leaving the **Energy consumption** window automatically sets the variable to FALSE again. • FALSE (default): Starting the measurement via the **Energy consumption** window is possible.
time	Duration of the measurement (s) Also displayed on the smartHMI in the **Energy consumption** window, under **Energy consumption of last 3 measurements** > **Time [s]** (top line)
energy	Energy consumption during measurement period (kWh) Also displayed on the smartHMI in the **Energy consumption** window, under **Energy consumption of last 3 measurements** > **Energy consumption [kWh]** (top line)
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)

3.88 $ENERGY_PERIOD: energy consumption of the last 60 min

Description

$ENERGY_PERIOD indicates the energy consumption of the robot and robot controller during the last 60 min since the most recent cold start.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy
```

The components are write-protected.

Element	Description
time	Duration of the measurement (s) • Maximum: 3,600 s The measurement begins at the last cold start. If the cold start took place more than 60 min ago, the measurement always covers the last 60 min. Also displayed on the smartHMI in the **Energy consumption** window, under **Continuous measurement (max. 1 h)** > **Time [s]**
energy	Energy consumption during measurement period (kWh) Also displayed on the smartHMI in the **Energy consumption** window, under **Continuous measurement (max. 1 h)** > **Energy consumption [kWh]**
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)

3.89 $ENERGY_TOTAL: energy consumption since last cold start

Description

$ENERGY_TOTAL indicates the energy consumption of the robot and robot controller since the most recent cold start.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy,
CHAR info[]
```

The components are write-protected.

Element	Description
time	Duration of the measurement (s) Corresponds to the time that has elapsed since the most recent cold start.
energy	Energy consumption during measurement period (kWh)
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat via the brake resistor (kWh)
info[]	For $ENERGY_TOTAL always without a value

3.90 $ENERGYMODULE: energy model available for robot?

Description

$ENERGYMODULE indicates whether and to what extent the energy model is available for this robot.

The state of $ENERGYMODULE is relevant for inquiries sent to KUKA Support, for example.

Data type: Energy_Module_Struc

Energy_Module_Struc

```
STRUC Energy_Module_Struc BOOL loaded INT estimated_axes
```

The components are write-protected.

Element	Description
loaded	• TRUE: the energy model is available for this robot. This means that for this robot, consumption can be determined – including external axes. • FALSE: the energy model is not available for this robot.
estimated_axes	Type: INT Bit array denotes how precisely the corresponding energy consumption can be determined for each axis. **Bit 1 … 6**: robot axis A1 ... A6 **Bit 7 … 12**: external axis E1 ... E6 • Bit n = 0: the consumption can be determined precisely for this axis. • Bit n = 1: the consumption can be determined approximately for this axis or is not considered.

Further information about the precision of the relevant axis can be read from $ENERGY_CONFIG_STATE.

Bit n = 1 means that same as $ENERGY_CONFIG_STATE[n] == #APPROXIMATED or #IGNORED.

3.91 $ERR

Description

Structure with information about the current program

The variable can be used to evaluate the currently executed program relative to the advance run. For example, the variable can be used to evaluate errors in the program in order to be able to respond to them with a suitable error strategy.

$ERR exists separately for the robot and submit interpreters. Each interpreter can only access its own variable. $ERR does not exist for the command interpreter.

Each subprogram level has its own representation of $ERR. In this way, the information from one level does not overwrite the information from different levels and information can be read from different levels simultaneously.

ON_ERROR_PROCEED implicitly deletes the information from $ERR in the current interpreter and at the current level.

Writability

The system variable is write-protected.

Syntax

Information=$ERR

Explanation of the syntax

Element	Description
Information	Type: Error_T List with information about the program currently being executed

Error_T

```
STRUC Error_T INT number, PROG_INT_E interpreter,
INT_TYP_E int_type, INT int_prio, line_nr, CHAR mod-
ule[24], up_name[24], TRIGGER_UP_TYPE trigger_type
```

Element	Description
Number	Only in the event of an interpreter error: Message number If no error has occurred, the value zero is displayed.
Interpreter	Current interpreter • #R_INT: Robot interpreter • #S_INT: System submit interpreter • #EXT_S_INT1: extended submit interpreter 1 • #EXT_S_INT2: extended submit interpreter 2 • …
int_type	Current program type and interrupt state • #I_NORMAL: The program is not an interrupt program. • #I_INTERRUPT: The program is an interrupt program. • #I_STOP_INTERRUPT: Interrupt by means of $STOPMESS (error stop)
int_prio	Priority of the interrupt
line_nr	Only in the event of an interpreter error: Number of the line that triggered the error **Note**: The number does not generally correspond to the line number in the smartHMI program editor! In order to understand the numbering, open the program with a simple editor and do not count lines that start with “&”. If no error has occurred, the value zero is displayed.
module[]	Name of the current program
up_name[]	Name of the current subprogram

Element	Description
trigger_type	Context in which the trigger belonging to a subprogram was triggered • #TRG_NONE: The subprogram is not a trigger subprogram. • #TRG_REGULAR: The trigger subprogram was switched during forward motion. • #TRG_BACKWARD: The trigger subprogram was switched during backward motion. • #TRG_RESTART: The trigger subprogram was switched on switching back to forward motion. • #TRG_REPLAY: The trigger subprogram was switched repeatedly after backward motion.

Example

$ERR can be used not only for error treatment, but also to determine the current surroundings.

In this example, a parameter is transferred to a subprogram from both a robot program and a submit program. In the subprogram, the system determines which interpreter the parameter came from. The action that is carried out depends on the result.

Robot program:

```
DEF Main ()
...
my_prog (55)
...
END
```

Submit program:

```
DEF my_sub ()
...
LOOP
     my_prog (33)
     ...
ENDLOOP
...
END
```

Subprogram:

```
GLOBAL DEF my_prog (par:IN)
INT par
INI
SWITCH $ERR.interpreter
     CASE #R_INT
       $OUT[par] = TRUE
     CASE #S_INT
       sub_prog_s()
     CASE #EXT_S_INT1
       sub_prog_1()
     CASE #EXT_S_INT2
       sub_prog_2()
     CASE #EXT_S_INT3
       sub_prog_3()
     ...
```

```
   ENDSWITCH
   ...
   END
```

Example 2

This example illustrates that each program level has its own representation of $ERR.

```
 1   DEF myMainProg ()
 2   INT myVar, myVar2
 3   INI
 4   ON_ERROR_PROCEED
 5   mySubProg (myVar)
 6   HALT
 7   myVar2 = 7
 8   mySubProg (myVar2)
 9   END
-------------------------------------
10   DEF mySubProg (myTest:IN)
11   INT myTest
12   HALT
13   END
```

Line	Description
4, 5	Line 5 actually triggers the message 1422 *{$variable} value invalid* because myVar is not initialized and can thus not be transferred to a subprogram. ON_ERROR_PROCEED in the preceding line suppresses the message.
6	If $ERR is read here via the variable display, the following components have the following values: $ERR.number == 1422 $ERR.line_nr == 15 $ERR.module[] == "MYMAINPROG" $ERR.up_name[] == "MYMAINPROG"
12	If $ERR is read here in the subprogram via the variable display, the following components have the following values: $ERR.number == 0 $ERR.line_nr == 0 $ERR.module[] == "MYMAINPROG" $ERR.up_name[] == "MYSUBPROG" This makes it clear that: $ERR always has the information from the current level (in this case, from the subprogram MySubProg). The information from MyMainProg, on the other hand, is unknown.

3.92 $ET1_NAME[]: Naming the kinematic system

Description

Name of the 1st external transformation

The variable defines the name of external transformation ET1. The name specified here is displayed in the **Robot** tab accessed via the menu sequence **Help** > **Info**. This gives the kinematic system a name that has no functional meaning in the system.

The names of external transformations ET2 to ET6 are defined analogously with the variables $ET2_NAME to $ET6_NAME.

Declaration

```
CHAR $ET1_NAME[40]
```

Explanation

Element	Description
$ET1_NAME[]	Data type: CHAR Name of transformation: Maximum 40 characters

3.93 $EX_AX_IGNORE

Description

$EX_AX_IGNORE can only be used in the WITH line of spline segments.

Each bit of $EX_AX_IGNORE corresponds to an external axis number. If a specific bit is set to the value “1”, the robot controller ignores the taught or programmed position of this external axis at the end point of the segment. Instead, the robot controller calculates the optimal position for this point on the basis of the surrounding external axis positions.

Recommendation: Whenever no specific position of the external axis is required for a point, use $EX_AX_IGNORE and set the bit for that external axis to the value “1”. This reduces the cycle time.

In the program run modes MSTEP and ISTEP, the robot stops at the positions calculated by the robot controller.

In the case of a block selection to a point with “$EX_AX_IGNORE = Bit n = 1”, the robot adopts the position calculated by the robot controller.

“$EX_AX_IGNORE = Bit n = 1” is not allowed for the following segments:

- For the first segment in a spline block (only up to KUKA System Software 8.2)
- For the last segment in a spline block
- In the case of successive segments with identical Cartesian end points, “$EX_AX_IGNORE = Bit n = 1” is not allowed for the first and last segments. (only up to and including KUKA System Software 8.2)

From KUKA System Software 8.3 onwards: If $EX_AX_IGNORE is programmed for an SPTP segment and the external axis concerned is mathematically coupled, the robot controller rejects $EX_AX_IGNORE. The taught or programmed position of that axis is taken into consideration. In T1/T2, the robot controller generates the following message: *Reject $EX_AX_IGNORE in line {Block number} because {External axis number} is mathematically coupled.*

Syntax

```
$EX_AX_IGNORE=Bit array
```

Explanation of the syntax

Element	Description
Bit array	• **Bit n = 1**: Taught/programmed position of the external axis is ignored. • **Bit n = 0**: Taught/programmed position of the external axis is taken into consideration.

Bit n	5	4	3	2	1	0
Axis	E6	E5	E4	E3	E2	E1

Example

```
SPLINE
   SPL P1
   SPL P2
   SLIN P3 WITH $EX_AX_IGNORE = 'B000001'
   SPL P4
ENDSPLINE
```

For P3, the robot controller ignores the taught position of external axis E1.

3.94 $EXT

Description

Signal declaration for Automatic External mode

This output is set when Automatic External mode is selected.

Syntax

`SIGNAL $EXT $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **996**

3.95 $EXT_SINT_LIST1 … $EXT_SINT_LIST7

Description

Information about an interrupt declared in a submit program that is assigned to one of the 1 … 7 extended submit interpreters

The information can be displayed via the variable display or by means of the diagnosis function in the main menu.

Procedure

- In the main menu, select **Diagnosis** > **Interrupts**.

Further information about interrupt programming is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected.

Syntax

intinfo = $EXT_SINT_LIST1[*index*]
…
intinfo = $EXT_SINT_LIST7[*index*]

Explanation of the syntax

Element	Description
index	Type: INT Index of the interrupt • **1 … 64**
intinfo	Type: STRUC INT_INFO Structure with information about the interrupt

INT_INFO

```
STRUC INT_INFO INT INT_PRIO, INT_STATE, INT_TYPE,
PROG_LINE, CHAR PROG_NAME[32]
```

Element	Description
INT_PRIO	Type: INT Priority of the interrupt • **1, 2, 4 … 39** • **81 …128**
INT_STATE	Bit array for interrupt states • **Bit 0 = 1**: Interrupt is declared and activated. • **Bit 1 = 1**: Interrupt is activated and enabled. • **Bit 2 = 1**: Interrupt is globally declared.
INT_TYPE	Type: INT Type of interrupt • **0**: Standard interrupt • **1**: Interrupt due to an EMERGENCY STOP ($EMSTOP) • **2**: Interrupt due to activation of the Fast Measurement inputs ($MEAS_PULSE) • **3**: Interrupt due to an error stop ($STOPMESS) • **4**: Interrupt due to a trigger (subprogram call)
PROG_LINE	Type: INT Line number of the submit program in which the interrupt is declared
PROG_NAME[]	Type: CHAR[] Directory and name of the submit program in which the interrupt is declared

3.96 $EXT_START

Description

Signal declaration for the external program start

If the Automatic External interface is active ($I_O_ACTCONF is **TRUE**), this input can be set by the higher-level controller to start or continue a program.

	Only the rising edge of the signal is evaluated.

Syntax

```
SIGNAL $EXT_START $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

3.97 $FAST_MEAS_COUNT

Description

Counter for fast measurement

The variable can be used to poll the number of edges detected at the inputs for fast measurement since the last reset of the counter.

Writability

The system variable is write-protected.

Syntax

number = `$FAST_MEAS_COUNT[index]`

Explanation of the syntax

Element	Description
state	Type: INT Number of the input for fast measurement • **1 … 8**
Number	Type: INT Number of measurements

3.98 $FAST_MEAS_COUNT_RESET

Description

Reset of the counter for fast measurement

The variable can be used to reset the counter $FAST_MEAS_COUNT and the corresponding time stamp $FAST_MEAS_COUNT_TIME to zero.

Syntax

```
$FAST_MEAS_COUNT_RESET = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: resets counter and time stamp. • FALSE: reset is not active. Default: FALSE

3.99 $FAST_MEAS_COUNT_TIME

Description

Time stamp of the counter for fast measurement

The variable contains the time stamp of the most recently requested values of the counter $FAST_MEAS_COUNT.

Writability

The system variable is write-protected.

Syntax

time stamp = `$FAST_MEAS_COUNT_TIME[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Number of the input for fast measurement • **1 ... 8**
time stamp	Type: REAL

3.100 $FILTER

Description

Programmed length of the mean value filter

The mean value filter is used to smooth out motions in order to stabilize increases in acceleration and eliminate disruptive frequencies (restrictions on jerking).

Effects of the mean value filter:

- In the case of PTP, LIN, CIRC: at axis level / somewhat path-distorting
- In the case of SPLINE, SLIN, SPTP: path-related / path-maintaining

The optimum value for the mean value filter is machine-specific and is stored in the machine datum **$DEF_FLT_PTP**. This value is assigned to the variable **$FILTER** of the standard packages. The value is reduced internally to a multiple of the interpolation cycle and, if necessary, offset against other filter elements. The effective filter time can be requested in **$FILTER_C**.

By using the mean value filter, the traversing time of an approximated motion sequence is extended to include the effective filter time.

NOTICE
The value of $FILTER should never be set by the user.

Writability

The system variable is write-protected.

Syntax

value = $FILTER

Explanation of the syntax

Element	Description
value	Type: INT; unit: ms

3.101 $FILTER_C

Description

Filter time currently used in the main run

The value of $FILTER_C corresponds to the effective filter time in the current motion command. The value is usually consistent with the filter time programmed in $FILTER. In the case of robot systems with internal filter adaptation (specific to machine data), the filter time may be different to the value programmed in $FILTER, depending on the programmed motion command.

Further information:
- $FILTER (>>> *3.100 "$FILTER" Page 76*)

Writability

The system variable is write-protected.

Syntax

value = $FILTER_C

Explanation of the syntax

Element	Description
value	Type: INT; unit: ms

3.102 $FLAG

Description

Management of flags

A Boolean expression can be freely assigned to the variable in a robot or submit program. Unlike $CYCFLAG (>>> *3.65 "$CYCFLAG" Page 50*), this Boolean expression is only evaluated at the time of assignment.

Syntax

$FLAG[*Number*]=*Boolean expression*

Explanation of the syntax

Element	Description
Number	Type: INT • **1 … 1,024**
Boolean expression	Type: BOOL • **TRUE**, **FALSE** or other Boolean expression Default: **FALSE**

3.103 $FOL_ERROR

Description

Following error relative to the position
The current following error for each axis is saved in the variable.

Writability

The system variable is write-protected.

Syntax

following error = `$FOL_ERROR[`*axis number*`]`

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
following error	Type: REAL Unit: • Rotational axes: ° • Linear axes: mm

3.104 $FCT_CALL

Description

Management number (handle) for command channel $FCT_CALL
The CWRITE() function can be used to call functions via the $FCT_CALL command channel.

Detailed information on the CWRITE() command can be found in the CREAD/CWRITE documentation .

Writability

The system variable is write-protected.

3.105 $GEAR_JERK

Description

Gear jerk of the axes in the advance run

The variable is used to define the gear jerk of an axis. The gear jerk is specified as a percentage of machine data defined with the dynamic model.

If $GEAR_JERK[…] is not initialized at the start of a spline motion, e.g. because the INI line has not been executed, the acknowledgement message *Gear jerk not programmed {Axis number}* is displayed and program execution is stopped.

Syntax

`$GEAR_JERK[`*axis number*`] = `*jerk*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
jerk	Type: INT; unit: % • **1 … 100**

3.106 $GEAR_JERK_C

Description

Gear jerk of the axes in the main run

The variable contains the currently valid gear jerk of an axis. The gear jerk is specified as a percentage of machine data defined with the dynamic model.

Writability

The system variable is write-protected.

Syntax

jerk = `$GEAR_JERK_C[`*axis number*`]`

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
jerk	Type: INT; unit: % • **1 … 100**

3.107 $H_AXIS_TOL

Description

Tolerance window for reaching the HOME position $AXIS_HOME

This variable of structure type E6AXIS is used to define the maximum permissible position deviation for each of the 5 additional HOME positions.

- A1 ... A6: Angular values in [°] or translation values in [mm]
- E1 ... E6: Angular values in [°] or translation values in [mm]

Example

```
$H_AXIS_TOL[1]={A1 2.0,A2 2.0,A3 2.0,A4 2.0,A5 2.0,A6
2.0,E1 2.0,E2 2.0,E3 2.0,E4 2.0,E5 2.0,E6 2.0}
```

3.108 $H_POS

Description

HOME position

This variable of structure type E6AXIS is used to define the HOME position.

- A1 ... A6: Angular values in [°] or translation values in [mm]
- E1 ... E6: Angular values in [°] or translation values in [mm]

Default

```
$H_POS={A1 0.0,A2 -90.0,A3 90.0,A4 0.0,A5 0.0,A6 0.0,E1
0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}
```

3.109 $H_POS_TOL

Description

Tolerance window for reaching the HOME position $H_POS

This variable of structure type E6AXIS defines the maximum permissible position deviation.

- A1 ... A6: Angular values in [°] or translation values in [mm]
- E1 ... E6: Angular values in [°] or translation values in [mm]

Example

```
$H_POS_TOL={A1 2.0,A2 2.0,A3 2.0,A4 2.0,A5 2.0,A6 2.0,E1
2.0,E2 2.0,E3 2.0,E4 2.0,E5 2.0,E6 2.0}
```

3.110 $HOLDING_TORQUE

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Description

Holding torque of an axis (torque mode)

The variable contains the holding torque of an axis calculated from the dynamic model. The holding torque refers to the current actual position of the axis and the current load.

The variable is write-protected. Its value is not dependent on the interpreter. In the robot program, the variable triggers an advance run stop.

If the upper and lower torque limits are set to $HOLDING_TORQUE for all axes, the robot must remain stationary when the brakes are released.
If this is not the case, i.e. if the robot drifts, the load is not correctly configured.

Syntax

`$HOLDING_TORQUE[`*Axis number*`]=`*Holding torque*

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 - 6**: Robot axis A1 ... A6 • **7 - 12**: External axis E1 ... E6
Holding torque	Type: REAL; unit: Nm (for linear axes: N) **Note:** For external axes, the value 0 N is always returned.

3.111 $HOLDING_TORQUE_MAND

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Description

Monitoring of the holding torque of an axis (torque mode)

If torque mode is activated with the function SET_TORQUE_LIMITS(), monitoring is carried out by default to check whether the holding torque (axes with dynamic model) or zero torque (axes without dynamic model) is in the programmed interval. This permissibility of the limits is only checked at the time of activation.

The purpose of the variable is to prevent unintentionally hazardous programming by means of SET_TORQUE_LIMITS(), as for normal applications the potentially dangerous state of allowing less than the holding torque is neither required nor intentional. Programmers who are aware of the effects and have to use the feature can deactivate the monitoring by writing to the variable. For this, the variable $HOLDING_TORQUE_MAND must be set to FALSE immediately before SET_TORQUE_LIMITS() and then reset.

Syntax

`$HOLDING_TORQUE_MAND[`*Axis number*`]=`*State*

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
State	Type: BOOL • TRUE: Monitoring is activated. • FALSE: Monitoring is deactivated. Default: TRUE

3.112 $HOME

Description

HOME directory of the compiler

Syntax

```
$HOME[] = "directory"
```

Explanation of the syntax

Element	Description
directory	Type: CHAR (max. 3 characters) • **/R1**: Robot system 1 • **/R2**: Robot system 2 Default: **/R1**

3.113 $I_O_ACT

Description

Signal declaration for the Automatic External interface
If this input is set, the Automatic External interface is active.

Syntax

```
SIGNAL $I_O_ACT $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.114 $I_O_ACTCONF

Description

Signal declaration for the active Automatic External interface

This output is set if Automatic External mode is selected and the Automatic External interface is active (input $I_O_ACT is TRUE). The higher-level controller can start a program.

Syntax

```
SIGNAL $I_O_ACTCONF $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **140**

3.115 $IDENT_OPT

Description

Enabling of load data determination

Syntax

```
$IDENT_OPT=State
```

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: load data can be determined. • FALSE: no load data can be determined. Default: TRUE

3.116 $ILLEGAL_SPEED

Description

Velocity limit value before the filter

This variable is used to define a limit value for monitoring the velocity before the filter $MONITOR_ILLEGAL_SPEED.

i	The value of the variable is unaffected by a software update.

Syntax

```
$ILLEGAL_SPEED=Limit value
```

Explanation of the syntax

Element	Description
Limit value	Type: INT; unit: % • **150 … 500** Default: **200** **Note:** If the value zero is entered here, the monitoring is deactivated.

3.117 $IMPROVEDMIXEDBLENDING

Description

Improved mixed approximate positioning

If the improved approximation mechanism is deactivated or the variable is missing in the file $OPTION.DAT, it is possible that points will not be approximated.

Syntax

`$IMPROVEDMIXEDBLENDING=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: improved approximate positioning is activated. • FALSE: improved approximate positioning is deactivated. Default: TRUE

3.118 $IN

Description

Value of a digital input

The system variable can be used to request the current value of a digital input in a robot or submit program or via the variable display function.

Writability

The system variable is write-protected.

Syntax

value = `$IN[`*input number*`]`

Explanation of the syntax

Element	Description
input number	Type: INT • **1 … 8192** **Note:** The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be requested via $NUM_IN/ $NUM_OUT.
value	Type: BOOL • TRUE or FALSE

3.119 $IN_HOME

Description

Signal declaration for reaching the HOME position

The variable $H_POS defines the HOME position of the robot in the machine data. By setting the output, the robot controller communicates to the higher-level controller that the robot is located in its HOME position.

Syntax

```
SIGNAL $IN_HOME $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1000**

3.120 $IN_HOME1 … $IN_HOME5

Description

Signal declaration for reaching HOME position 1 ... 5

The variable $AXIS_HOME allows up to 5 HOME positions to be defined in $machine.dat under KRC:\R1\Mada (in addition to the HOME position defined using $H_POS).

By setting the output $IN_HOME*x*, the robot controller communicates to the higher-level controller that the robot is located in HOME position *x* (1 ... 5).

Syntax

Default in $machine.dat under KRC:\STEU\Mada:

```
SIGNAL $IN_HOMEindex FALSE
```

To link an output to an $IN_HOME, make the following adaptation:

```
SIGNAL $IN_HOMEIndex $OUT[Output number]
```

Explanation of the syntax

Element	Description
Index	Type: INT Index of the HOME position • **1 … 5**
Output number	Type: INT

3.121 $INDIVIDUAL_MAMES

Description

Offset from the mastering position

This variable is used to define whether the offset data for mastering are saved. This depends on the robot model.

- In the conventional robot calibration procedure, the mastering marks are set exactly to the mastering position saved in $MAMES[*x*], e.g. 0.0, -90.0, 90.0, 0.0, 0.0, 0.0.
- For manufacturing reasons, a new procedure has been introduced. The mastering equipment has been fixed, and during robot calibration the offset from the mastering position saved in $MAMES[*x*] is calculated.

  The offset data are saved on the RDC with the file name *Robot serial number*.MAM.

Syntax

```
$INDIVIDUAL_MAMES = state
```

Explanation of the syntax

Element	Description
state	Type: ENUM INDIVIDUAL_MAMES • #NONE: no offset is saved. • #RDC: an offset is saved. During mastering, the robot controller accesses the offset data on the hard drive and calculates the exact mastering position. **Note:** The offset is not taken into consideration during reference mastering. **Note:** If #RDC is used, the mastering position of the manipulator must be stored in $MAMES[*x*].

3.122 $INPOSITION

Description

Positioning window reached – axis in position

Syntax

```
$INPOSITION=Bit array
```

Explanation of the syntax

Element	Description
Bit array	Bit array indicating the axes that have reached the positioning window. • **Bit n = 0**: Axis still moving • **Bit n = 1**: Axis in the positioning window

Bit n	12 …	5	4	3	2	1	0
Axis	E6	A6	A5	A4	A3	A2	A1

3.123 $INSIM_TBL

Description

Simulation of inputs

- To simulate an input, it must be set to the desired simulated state (TRUE or FALSE).
- Simulation must also be activated.
  (>>> *3.129 "$IOSIM_OPT" Page 90*)

Some inputs are write-protected and cannot be simulated.

Simulated inputs are labeled with "SIM" in the KUKA System Software while write-protected ones are labeled with "SYS". The display is called via the main menu:

- **Display** > **Inputs/outputs** > **Digital I/O**

Precondition

- The I/O simulation can only be used with an Office PC (OPS).

Syntax

`$INSIM_TBL[`*input number*`] = `*state*

Explanation of the syntax

Element	Description
input number	Type: INT
state	Type: CHAR • **"-"**: the input is not simulated. • **"1"**: the input is TRUE (simulated). • **"0"**: the input is FALSE (simulated). Default: **"-"**

Properties

- If simulation is activated, the robot controller takes simulated inputs into account.
- When simulation is deactivated again:
  - All inputs resume their real state.
- When the robot controller is rebooted:
  - Simulation is automatically deactivated.
  - The simulated state of each input is reset.

Example

Input 8 is set to the simulated state TRUE.

```
$INSIM_TBL[8] = "1"
```

3.124 $INSTALLED_MOTION_MODES

Description

The variable lists the installed Motion Modes with TRUE and the non-installed Motion Modes with FALSE.

The variable can only be modified manually in simulation operation; this is because simulation operation of KUKA Motion Modes is free of charge in OfficeLite and OfficePC and is possible without using installed option packages.

Manual modification of the variable on a real controller leads to errors. In the case of a real controller, installation of the corresponding Motion Modes option packages is imperative.

Writability

The system variable is write-protected.

Syntax

```
{PATH state, DYNAMIC state, ECO state} = $INSTALLED_MOTION_MODES
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: the Motion Mode is installed. • FALSE: the Motion Mode is not installed. Default: FALSE

3.125 $INTERPRETER

Description

Selecting the interpreter

Numerous system states can be read, and in many cases also set, via variables. Strictly speaking, these variables exist multiple times – once per interpreter. They are identical in name for all interpreters.

Examples:

- Program run mode ($PRO_MODE)
- Program state ($PRO_STATE)
- Data of the process pointer ($PRO_IP)
- WAIT FOR statement at which the interpreter is currently waiting ($WAIT_FOR)

When such a variable is accessed, this is always referred automatically to the current interpreter. This is defined by $INTERPRETER.

Syntax

```
$INTERPRETER = interpreter
```

Explanation of the syntax

Element	Description
Interpreter	Type: INT Selection of the interpreter • 1: Robot interpreter • 2: System submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • Etc. Default: 1

3.126 $IOBLK_EXT

Description

Specifies whether it is possible to set outputs via the smartPAD in Automatic External mode.

Syntax

`$IOBLK_EXT =` *State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: the setting of outputs via the smartPAD in Automatic External mode is blocked. • FALSE: the setting of outputs via the smartPAD in Automatic External mode is not blocked. Default: TRUE

3.127 $IOBUS_INFO

Description

Structure with information about the bus driver

Writability

The system variable is write-protected.

Syntax

information `= $IOBUS_INFO[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Bus driver number • **1 … 32** The serial number is automatically assigned to the bus drivers.
information	Type: Iobus_Info_T List with information about the bus driver

Iobus_Info_T

```
STRUC Iobus_Info_T CHAR name[256], drv_name[256], BOOL
bus_ok, bus_installed
```

Element	Description
name[]	Name of the bus instance, e.g. SYS-X44
drv_name[]	Name of the bus driver, e.g. ECat.DRV

Element	Description
bus_ok	• TRUE: bus driver is OK. • FALSE: bus driver is faulty or incompatible.
bus_installed	• TRUE: bus driver is installed. • FALSE: bus driver is not installed.

3.128 $IOSIM_IN

Description

Value of a digital input with simulation calculated

The system variable can be used to request the current value of a digital input in the robot program or via the variable display function. It also specifies whether or not the input is simulated.

Writability

The system variable is write-protected.

Syntax

value = `$IOSIM_IN[`*input number*`]`

Explanation of the syntax

Element	Description
input number	Type: INT • **1 … 8192** **Note:** The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be requested via $NUM_IN/ $NUM_OUT.
value	Type: CHAR • **0**: FALSE and not simulated • **1**: TRUE and not simulated • **2**: FALSE and simulated • **3**: TRUE and simulated • **4**: FALSE and not simulated and system input • **5**: TRUE and not simulated and system input • **6**: FALSE and simulated and system input • **7**: TRUE and simulated and system input

3.129 $IOSIM_OPT

Description

Activation or deactivation of simulation

Precondition

- The I/O simulation can only be used with an Office PC (OPS).

Syntax

`$IOSIM_OPT =` *state*

Explanation of the syntax

Element	Description
state	Type: BOOL • FALSE: simulation is deactivated. • TRUE: simulation is activated. Default: FALSE

Properties

- If simulation is activated, the robot controller takes simulated inputs and outputs into account.
  Inputs and outputs are simulated by means of the system variables $INSIM_TBL and $OUTSIM_TBL.
- Outputs can only be set if the enabling switch is pressed.
- If simulation is not activated, the robot controller takes account of the real state of all inputs and outputs, and the simulated state is not relevant.

Robot controller response:

- If an output[x] is simulated, its real state (i.e. $OUT[x]) can no longer be modified. To allow this, the simulated state of the output must first be reset.
- When simulation is deactivated again:
  - All outputs resume the state they had prior to simulation.
  - All inputs resume their real state.
- When the robot controller is rebooted:
  - Simulation is automatically deactivated.
  - The simulated state of each input and output is reset.

Example 1

State before simulation: $OUT[8] == FALSE

1. The simulated state of the output is set to TRUE. ($OUTSIM_TBL[8] = "1")
2. Simulation is activated. ($IOSIM_OPT = TRUE)
   The real state now reflects the simulated state, i.e. $OUT[8]==TRUE.
   $OUT[8] can no longer be modified.
3. Simulation is deactivated. ($IOSIM_OPT = FALSE)
   Now $OUT[8] == FALSE.

Deactivation of the simulation has reset $OUT[8] to the state it had before the simulation, i.e. FALSE. $OUT[8] can now be modified again.

Example 2

State before simulation: $OUT[9] == FALSE
Furthermore, $OUTSIM_TBL[9] = "**-**", i.e. the output is not simulated.

1. Simulation is activated. ($IOSIM_OPT = TRUE)
2. The real state of the output is changed to TRUE. ($OUT[9] = TRUE)
3. Simulation is deactivated again. ($IOSIM_OPT = FALSE)

Now $OUT[9] == FALSE.

Deactivation of the simulation has reset $OUT[9] to the state it had before the simulation, i.e. FALSE.

3.130 $IOSIM_OUT

Description

Value of a digital output with simulation calculated

The system variable can be used to request the current value of a digital output in the robot program or via the variable display function. It also specifies whether or not the output is simulated.

Writability

The system variable is write-protected.

Syntax

value = `$IOSIM_OUT[`*output number*`]`

Explanation of the syntax

Element	Description
output number	Type: INT • **1 … 8192** **Note:** The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be requested via $NUM_IN/ $NUM_OUT.
value	Type: CHAR • **0**: FALSE and not simulated • **1**: TRUE and not simulated • **2**: FALSE and simulated • **3**: TRUE and simulated • **4**: FALSE and not simulated and system input • **5**: TRUE and not simulated and system input • **6**: FALSE and simulated and system input • **7**: TRUE and simulated and system input

3.131 $IOSYS_IN_FALSE

Description

Number of the system input that is always FALSE

By default, input $IN[1026] is configured as the system input that is always FALSE.

In the VW System Software, a different system input can be configured. The number of this system input can be requested using the variable.

Writability

The system variable is write-protected.

Syntax

Input number = `$IOSYS_IN_FALSE`

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

3.132 $IOSYS_IN_TRUE

Description

Number of the system input that is always TRUE

By default, input $IN[1025] is configured as the system input that is always TRUE.

In the VW System Software, a different system input can be configured. The number of this system input can be requested using the variable.

Writability

The system variable is write-protected.

Syntax

Input number = `$IOSYS_IN_TRUE`

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.133 $IPO_MODE

Description

Interpolation mode in the advance run

The variable defines the interpolation mode:

- Base-related interpolation of the motion path (default setting after a controller boot)

  This mode is used if the tool is mounted on the robot flange. The robot controller then calculates the current position $POS_ACT in relation to the BASE coordinate system.
- Gripper-related interpolation of the motion path

  This mode is used if the tool is anchored in the space and the workpiece itself is guided along the external/fixed tool using a gripper. The robot controller then calculates the current position $POS_ACT in relation to the TOOL coordinate system.

Writability

Interpolation mode can only be changed if gripper-related interpolation is switched on in $option.dat:

- $TCP_IPO=TRUE

Syntax

```
$IPO_MODE = mode
```

Explanation of the syntax

Element	Description
mode	Type: ENUM IPO_MODE • #TCP: gripper-related interpolation (external/fixed tool) • #BASE: base-related interpolation (tool on the robot flange) Default: #BASE

3.134 $IPO_MODE_C

Description

Interpolation mode in the main run
The variable displays the current interpolation mode.

Writability

The system variable is write-protected.

Syntax

```
mode = $IPO_MODE_C
```

Explanation of the syntax

Element	Description
mode	Type: ENUM IPO_MODE • #TCP: gripper-related interpolation (external/fixed tool) • #BASE: base-related interpolation (tool on the robot flange) Default: #BASE

3.135 $IS_OFFICE_LITE

Description

The variable indicates whether the installation in question is an OfficeLite system.

Writability

The system variable is write-protected.

Syntax

```
Identifier = $IS_OFFICE_LITE
```

Explanation of the syntax

Element	Description
Identifier	Type: BOOL • TRUE: Software is KUKA.OfficeLite. • FALSE: Software is KUKA System Software.

Example

```
IF ($IS_OFFICE_LITE == TRUE)
THEN
DoSomething() ; This call should only occur if the
installation is an OfficeLite system
ENDIF
```

3.136 $I2T_OL

Description

Deactivation of I^2t monitoring (only permissible in KUKA.OfficeLite)
The variable can only be used with the KR C2.

Syntax

`$I2T_OL =` *state*

Explanation of the syntax

Element	Description
state	Type: ENUM SW_ONOFF • #ON: I^2t monitoring is activated. • #OFF: I^2t monitoring is not activated. Default: #ON

3.137 $KCP_IP

Description

IP address of the currently connected KUKA smartPAD

Writability

The system variable is write-protected.

Syntax

IP address `= $KCP_IP`

Explanation of the syntax

Element	Description
IP address	Type: INT If the value zero is displayed, no smartPAD is connected.

3.138 $KCP_POS

Description

Position of the KUKA smartPAD relative to the position of the robot (compass dial)

Syntax

`$KCP_POS=`*Position*

Explanation of the syntax

Element	Description
Position	Type: REAL; unit: ° Default: **0.0**

3.139 $KCP_TYPE

Description

Currently connected teach pendant

The variable can be used to monitor which teach pendant is currently connected to the robot controller.

Writability

The system variable is write-protected.

Syntax

type = `$KCP_TYPE`

Explanation of the syntax

Element	Description
type	Type: ENUM KCP_TYPE • #NO_KCP: no teach pendant is connected. • #KCP: KUKA smartPAD is connected. • #VRP: a virtual KUKA smartPAD is connected (KUKA.VirtualRemotePendant).

3.140 $KDO_ACT

Description

Indication of whether a command motion is currently active

Examples of command motions are jog motions and motions of asynchronous axes.

Writability

The system variable is write-protected.

Syntax

Status = `$KDO_ACT`

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Command motion active • FALSE: No command motion active

3.141 $KR_SERIALNO

Description

Robot serial number saved on the RDC

Syntax

```
$KR_SERIALNO = number
```

Explanation of the syntax

Element	Description
Number	Type: INT

3.142 $KR_SERIALNO_CAL

Description

Serial number of the robot according to CAL file
The CAL file contains data from the EMD mastering.

Writability

The system variable is write-protected.

Syntax

```
number = $KR_SERIALNO_CAL
```

Explanation of the syntax

Element	Description
Number	Type: INT

3.143 $LANGUAGE

Description

Language selection
The language currently set on the KUKA smartHMI can be displayed and changed with the aid of the variable.

Writability

- Via the variable display
- Via the KRL program (robot interpreter)

Syntax

```
$LANGUAGE[]="shortcut"
```

Explanation of the syntax

Element	Description
shortcut	Type: CHAR[6] Language abbreviation

Language abbreviation

Language	Language abbreviation
Chinese	zh
Danish	da
German	de
English	en
Finnish	fi
French	fr
Greek	el
Italian	it
Japanese	ja
Korean	ko
Dutch	nl
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Swedish	sv
Slovak	sk
Slovenian	sl
Spanish	es
Czech	cs
Turkish	tr
Hungarian	hu
Vietnamese	vi

Example

```
$LANGUAGE[]="zh"
```

3.144 $LDC_ACTIVE

Description

Activation/deactivation of online load data verification

Precondition

- Online load data verification is loaded: $LDC_LOADED=TRUE

Syntax

```
$LDC_ACTIVE=State
```

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: online load data verification is activated. • FALSE: online load data verification is deactivated. Default: TRUE

3.145 $LDC_LOADED

Description

Indication of whether online load data verification is loaded

The variable can be used to check whether online load data verification for the current robot type is available. Online load data verification is available for those robot types for which KUKA.LoadDataDetermination can be used.

Syntax

```
$LDC_LOADED=State
```

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: online load data verification is loaded. • FALSE: online load data verification is not loaded.

3.146 $LDC_RESULT

Description

Current result of the online load data verification

Syntax

```
$LDC_RESULT = Result
```

Explanation of the syntax

Element	Description
Result	Type: CHAR • #OK: The payload is OK. (Neither overload nor underload.) • #OVERLOAD: There is an overload. • #UNDERLOAD: There is an underload. • #CHECKING: The load data verification is still active. • #NONE: There is currently no result, e.g. because the tool has been changed.

3.147 $LINE_SEL_OK

Description

Execute a block selection

Syntax

`$LINE_SEL_OK=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: block selection was executed. • FALSE: no block selection Default: FALSE

3.148 $LOAD

Description

Currently valid load data in the advance run

The structure contains the payload data entered in the robot controller and assigned to the current tool. The reference coordinate system is the FLANGE coordinate system.

(>>> *"Loads on the robot" Page 100*)

Syntax

`$LOAD={M` *Mass*`, CM` *Center of gravity*`, J` *Inertia*`}`

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the flange • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the flange by A, B and C

Loads on the robot

Various loads can be mounted on the robot:

- Payload on the flange
- Supplementary load on axis 3
- Supplementary load on axis 2
- Supplementary load on axis 1

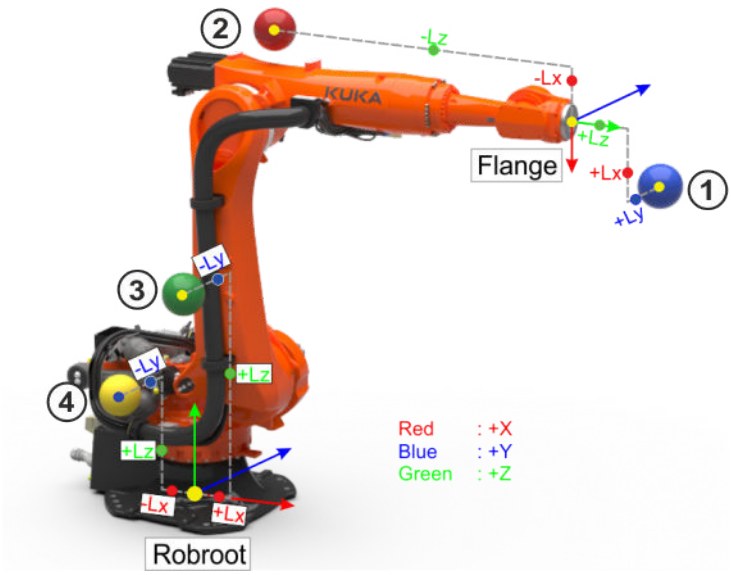


Fig. 3-7: Loads on the robot

1	Payload	3	Supplementary load on axis 2
2	Supplementary load on axis 3	4	Supplementary load on axis 1

Parameters

The load data are defined using the following parameters:

Parameter		Unit
Mass	m	kg
Distance to the center of gravity	L_x, L_y, L_z	mm
Mass moments of inertia at the center of gravity	I_x, I_y, I_z	kg m^2

Reference systems of the X, Y and Z values for each load:

Load	Reference system
Payload	FLANGE coordinate system
Supplementary load A3	FLANGE coordinate system A4 = 0°, A5 = 0°, A6 = 0°
Supplementary load A2	ROBROOT coordinate system A2 = -90°
Supplementary load A1	ROBROOT coordinate system A1 = 0°

3.149 $LOAD_C

Description

Currently valid load data in the main run

The structure contains the payload data entered in the robot controller and assigned to the current tool. The reference coordinate system is the FLANGE coordinate system.

(>>> *"Loads on the robot" Page 100*)

Writability

The system variable is write-protected.

Syntax

```
{M Mass, CM Center of mass, J Inertia} = $LOAD_C
```

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the flange • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the flange by A, B and C

3.150 $LOAD_A1

Description

Currently valid supplementary load data for axis A1 in the advance run

The structure contains the supplementary load data of the load mounted on axis A1 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the ROBROOT coordinate system with A1= 0°.

Syntax

```
$LOAD_A1={M Mass, CM Center of gravity, J Inertia}
```

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the robot base by A, B and C.

3.151 $LOAD_A1_C

Description

Currently valid supplementary load data for axis A1 in the advance run

The structure contains the supplementary load data of the load mounted on axis A1 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the ROBROOT coordinate system with A1= 0°.

Writability

The system variable is write-protected.

Syntax

`{M` *Mass*`, CM` *Center of mass*`, J` *Inertia*`} = $LOAD_A1_C`

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the robot base by A, B and C.

3.152 $LOAD_A2

Description

Currently valid supplementary load data for axis A2 in the advance run

The structure contains the supplementary load data of the load mounted on axis A2 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the ROBROOT coordinate system with A2= -90°.

	In the case of a SCARA robot with 4 axes, the reference coordinate system is the ROBROOT coordinate system with A2= 0°.

Syntax

`$LOAD_A2={M` *Mass*`, CM` *Center of gravity*`, J` *Inertia*`}`

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the robot base by A, B and C.

3.153 $LOAD_A2_C

Description

Currently valid supplementary load data for axis A2 in the advance run

The structure contains the supplementary load data of the load mounted on axis A2 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the ROBROOT coordinate system with A2= -90°.

	In the case of a SCARA robot with 4 axes, the reference coordinate system is the ROBROOT coordinate system with A2= 0°.

Writability

The system variable is write-protected.

Syntax

`{M` *Mass*`, CM` *Center of mass*`, J` *Inertia*`} = $LOAD_A2_C`

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the robot base by A, B and C.

3.154 $LOAD_A3

Description

Currently valid supplementary load data for axis A3 in the advance run

The structure contains the supplementary load data of the load mounted on axis A3 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the FLANGE coordinate system with A4= 0°, A5= 0° and A6= 0°.

Syntax

```
$LOAD_A3={M Mass, CM Center of gravity, J Inertia}
```

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the flange • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the flange by A, B and C

3.155 $LOAD_A3_C

Description

Currently valid supplementary load data for axis A3 in the main run

The structure contains the supplementary load data of the load mounted on axis A3 and entered on the robot controller.

(>>> *"Loads on the robot" Page 100*)

The reference coordinate system is the FLANGE coordinate system with A4= 0°, A5= 0° and A6= 0°.

Writability

The system variable is write-protected.

Syntax

```
{M Mass, CM Center of mass, J Inertia} = $LOAD_A3_C
```

Explanation of the syntax

Element	Description
Mass	Type: REAL; unit: kg
Center of gravity	Type: FRAME • **X**, **Y**, **Z**: Position of the center of gravity relative to the flange • **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
Inertia	Type: INERTIA • **X**, **Y**, **Z**: Mass moments of inertia about the axes of the coordinate system that is rotated relative to the flange by A, B and C

3.156 $LOOP_CONT

Description

Simulation of a wait condition

If a wait message activated with $LOOP_MSG is displayed, the variable $LOOP_CONT is automatically set to TRUE.

The **Simulate** button in the message window is associated with a wait message. This button can be used to simulate the wait condition and continue with the program execution (precondition: operating mode T1 or T2.)

If the **Simulate** button is pressed and the wait condition is simulated, then the wait message will be hidden and the variable $LOOP_CONT reset.

Syntax

`$LOOP_CONT=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: wait message is displayed. • FALSE: wait condition was simulated with the **Simulate** button and the wait message hidden. Default: FALSE

3.157 $LOOP_MSG

Description

Activation of a wait message

The variable can be used to activate a wait message relating to a wait statement in the KRL program. The message text defined with the variable and the **Simulate** button are displayed. In order to end the wait message, the message text must be deleted from the variable again. For this purpose, an empty string can be programmed with $LOOP_MSG or the StrClear() function can be used.

The signal combination at the inputs/outputs defined in a wait statement can be simulated and program execution resumed via the **Simulate** button. Precondition: Operating mode T1 or T2.

Syntax

```
$LOOP_MSG[]="Message"
```

Explanation of the syntax

Element	Description
Message	Type: CHAR Message text: max. 128 characters

3.158 $MAMES

Description

Mastering position

The variable contains the specific mastering position for a robot type (= offset between the mechanical zero position (mastering mark) and the electronic zero position).

The robot-specific mastering position may deviate slightly. If this is the case, the offset relative to the mastering position stored in $MAMES[*x*] for each axis is determined and saved in a MAM file.

During mastering, the axes are moved to the mechanical zero position. In the mechanical zero position, the axis counter takes the degree or millimeter value saved under $MAMES[*x*] as the current axis position. If offset data are saved, the axis counter takes the MAMES value plus the saved offset as the current axis position.

Syntax

```
$MAMES[axis number] = axis value
```

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
axis value	Type: REAL; unit: ° (for linear axes: mm) • **-180°… +180°**

Example

```
$MAMES[1]=0.0
$MAMES[2]=-90.0
$MAMES[3]=90.0
$MAMES[4]=0.0
$MAMES[5]=0.0
$MAMES[6]=0.0
```

3.159 $MAMES_ACT

Description

Robot-specific mastering position

The mastering position for each axis of a specific robot type is defined by means of $MAMES[*x*] in the machine data.

(>>> *3.158 "$MAMES" Page 107*)

The robot-specific mastering position $MAMES_ACT[*x*] may deviate slightly. The offset relative to the mastering position stored in $MAMES[*x*] is then saved as a MAM file on the RDC.

If offset values for the mastering are saved and are to be used, this must be specified in the machine data with $INDIVIDUAL_MAMES.

(>>> *3.121 "$INDIVIDUAL_MAMES" Page 85*)

The robot controller then reads the offset values during booting, adds them to the $MAMES values and writes the result to the variable $MAMES_ACT[*x*].

- If a MAM file with offset data is to be used, $INDIVIDUAL_MAMES ≠ #NONE, $MAMES_ACT[*x*] = $MAMES[*x*] + MAM offset.
- If a MAM file with offset data is to be used, $INDIVIDUAL_MAMES ≠ #NONE, but a MAM file is not saved, $MAMES_ACT[*x*] is invalid.
- If a MAM file with offset data is not to be used, $INDIVIDUAL_MAMES = #NONE, $MAMES_ACT[*x*] = $MAMES[*x*].

Writability

The system variable is write-protected.

Syntax

axis value = `$MAMES_ACT[`*axis number*`]`

Explanation of the syntax

Element	Description
axis value	Type: REAL; unit: ° (for linear axes: mm) • **-180°… +180°**
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6

Example

The $MAMES value in the machine data for A3 is 90°. In the MAM file, an offset of 1° is saved for this axis:

- $MAMES_ACT[3] = 90.0 + 1.0 = 91.0

3.160 $MEAS_PULSE

Description

Input for activating fast measurement.

The variable can be used to activate fast measurement via an interrupt. With **$MEAS_PULSE**, the interrupt condition is not evaluated at the usual intervals of 4 ms, but rather at the drive system bus clock rate of 125 mi-

croseconds. In the event of an edge change of the input, the robot position is recorded at the bus clock rate of the drive system and is available in the program via the two variables **$AXIS_INT** (>>> *3.36 "$AXIS_INT" Page 33*) and **$POS_INT** (>>> *3.200 "$POS_INT" Page 130*). One edge change can be detected every 4 ms.

The Fast Measurement inputs must be activated before use.

Syntax

`$MEAS_PULSE[`*number*`] = `*state*

Explanation of the syntax

Element	Description
Number	Type: INT Number of the measurement input • **1 ... 8** Measurement inputs 6 ... 8 are assigned internally and cannot be used freely.
state	Type: BOOL • TRUE: Measurement input is active. • FALSE: Measurement input is not active. Default: FALSE

Example program

In this program example, Fast Measurement input 4 is TRUE by default and the interrupt is triggered on switching to FALSE. This corresponds, for example, to the behavior of a photo-electric barrier that is interrupted.

```
DEF Mainprog()
...
IF VARSTATE("$MEAS_PULSE[4]")==#INITIALIZED THEN ;Check
whether the Fast Measurement inputs have been activated in
WorkVisual
INTERRUPT DECL 31 WHEN NOT $MEAS_PULSE[4] DO Fastmeasure()
;Declare interrupt 31 to Fast Measurement input 4
INTERRUPT ON 31  ;Activate interrupt 31
ENDIF
...
END

DEF Fastmeasure()
E6POS Meas_Pos
E6AXIS Meas_Axis
INTERRUPT OFF 31  ;Deactivate the interrupt to ignore any
potential sensor bounce
  Meas_Axis = $AXIS_INT        ;Copy the determined values
to variables
  Meas_Pos = $POS_INT

 INTERRUPT ON 31  ;Activate interrupt 31 for measuring
probe
END
```

3.161 $MODE_OP

Description

Current operating mode

The variable displays the operating mode currently set.

Syntax

mode = $MODE_OP

Explanation of the syntax

Element	Description
mode	Type: ENUM MODE_OP • #AUT: Automatic mode • #EX: Automatic External mode • #T1: T1 mode • #T2: T2 mode • #INVALID: Invalid operating mode

Example

```
DECL MODE_OP opmode

opmode = $MODE_OP

IF opmode == #AUT THEN
 HALT
ENDIF
```

3.162 $MONITOR_ILLEGAL_SPEED

Description

Velocity monitoring before the filter

This variable allows the monitoring of the velocity before the filter to be deactivated, without changing the variable $ILLEGAL_SPEED.

Syntax

$MONITOR_ILLEGAL_SPEED=*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • **TRUE**: Monitoring is activated. • **FALSE**: Monitoring is not activated. Default: **TRUE**

3.163 $MOT_STOP

Description

Disabling of the external start

The variable is set if the robot is not on the programmed path and the external start is disabled. The robot controller resets the variable if the user answers **Yes** to the prompt for confirmation of whether the robot should nevertheless be started. The external start from the higher-level controller is issued subsequently in this case.

Precondition

- “Block external start” option is active: $MOT_STOP_OPT=TRUE
  (>>> *3.164 "$MOT_STOP_OPT" Page 111*)

Syntax

`$MOT_STOP=`*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: External start is blocked. • FALSE: External start is not blocked. Default: FALSE

3.164 $MOT_STOP_OPT

Description

Activation of the “Block external start” option

The variable can be used to activate the function defined in the variable $MOT_STOP.

(>>> *3.163 "$MOT_STOP" Page 111*)

Syntax

`$MOT_STOP_OPT=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: option is activated. • FALSE: option is not activated. Default: FALSE

3.165 $MOT_TEMP

Description

Current motor temperature of an axis

In the case of master/slave axes, only the motor temperature of the master drive can be read.

Writability

The system variable is write-protected.

Syntax

Temperature = `$MOT_TEMP[`*Axis number*`]`

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
Temperature	Type: INT; unit: Kelvin; tolerance: ±12 K The value zero is displayed for axes that are not configured.

3.166 $MOUSE_ACT

Description

Operating state of the 6D mouse

Syntax

`$MOUSE_ACT =` *state*

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: 6D mouse is active. • FALSE: 6D is not active. Default: FALSE

3.167 $MOUSE_DOM

Description

Jog mode of the 6D mouse

Syntax

`$MOUSE_DOM =` *mode*

Explanation of the syntax

Element	Description
mode	Type: BOOL • TRUE: dominant mode is active. Only the axis with the greatest deflection of the 6D mouse is moved. • FALSE: dominant mode is not active. Depending on the axis selection, either 3 or 6 axes can be moved simultaneously. Default: TRUE

3.168 $MOUSE_ON

Description

Activation/deactivation of the 6D mouse on the KUKA smartPAD

Syntax

`$MOUSE_ON = state`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: 6D mouse is activated. • FALSE: 6D mouse is deactivated. Default: TRUE

3.169 $MOUSE_ROT

Description

Rotational motions with the 6D mouse on/off

Syntax

`$MOUSE_ROT = state`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: rotational motions are possible. • FALSE: rotational motions are not possible. Default: TRUE

3.170 $MOUSE_TRA

Description

Translational motions with the 6D mouse on/off

Syntax

```
$MOUSE_TRA = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: translational motions are possible. • FALSE: translational motions are not possible. Default: TRUE

3.171 $MOVE_BCO

Description

Indication of whether a BCO run is currently being executed

Writability

The system variable is write-protected.

Syntax

Status = `$MOVE_BCO`

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: BCO run is being carried out. • FALSE: BCO run is not being carried out.

3.172 $MOVE_ENA_ACK

Description

Signal declaration for signaling the motion enable

By setting this output, the robot controller communicates to the higher-level controller that it has received the motion enable signal **$MOVE_ENABLE**.

Syntax

`SIGNAL $MOVE_ENA_ACK $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **150**

3.173 $MOVE_ENABLE

Description

Signal declaration for motion enable

This input is used by the higher-level controller to check the robot drives.

- If the higher-level controller sets the input to **TRUE**, the robot can be moved manually and in program mode.
- If the input is set to **FALSE**, the drives are switched off and the active commands are inhibited.

If the drives have been switched off by the higher-level controller, the message *General motion enable* is displayed. It is only possible to move the robot again once this message has been acknowledged and an external start signal ($EXT_START) has been given.

Syntax

```
SIGNAL $MOVE_ENABLE $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.174 $MOVE_STATE

Description

Current motion state

The variable indicates the current motion path section of the motion.

Writability

The system variable is write-protected.

Syntax

```
state = $MOVE_STATE
```

Explanation of the syntax

Element	Description
state	Type: ENUM MOVE_STATE Old range of motion: • #PTP_AP01, #LIN_AP01, #CIRC_AP01 PTP, LIN or CIRC motion in the first approximate positioning range (start to middle) • #PTP_AP02, #LIN_AP02, #CIRC_AP02 PTP, LIN or CIRC motion in the second approximate positioning range (middle to end) • #PTP_SINGLE, #LIN_SINGLE, #CIRC_SINGLE PTP, LIN or CIRC motion outside an approximate positioning range New range of motion (Spline): • #SPLINE_SPL, #SPLINE_LIN, #SPLINE_CIRC SPL, SLIN or SCIRC segment in a CP-SPLINE block • #SPLINE_PTP SPTP segment in a PTP-SPLINE block • #SPLINE_UES Spline motion in a Cartesian approximate positioning range, i.e. approximate positioning in a CP-SPLINE block or in a single SPL, SLIN or SCIRC motion • #SPLINE_UES_PTP Spline motion in an axial approximate positioning range, i.e. approximate positioning in a PTP-SPLINE block or in a single SPTP motion Other states: • #MOVE_EMI_SINGLE EMI motion • #NONE Currently no motion

3.175 $NEAR_POSRET

Description

Signal declaration for the tolerance window about $POS_RET

By setting the output, the robot controller communicates to the higher-level controller that the robot is located within a sphere about the position saved in $POS_RET. The higher-level controller can use this information to decide whether or not the program may be restarted.

The user can define the radius of the sphere in the file $custom.dat using the variable $NEARPATHTOL.

Syntax

```
SIGNAL $NEAR_POSRET $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **147**

3.176 $NEARPATHTOL

Description

Tolerance for deviation from $POS_RET (= radius of the sphere about $POS_RET)

Syntax

`$NEARPATHTOL=`*Radius*

Explanation of the syntax

Element	Description
Radius	Type: REAL; unit: mm

3.177 $NULLFRAME

Description

NULLFRAME

The variable of structure type FRAME can be used to set all components of a coordinate system to zero:

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

Example

Position of the BASE coordinate system is NULLFRAME

```
$BASE=$NULLFRAME
```

3.178 $NUM_AX

Description

Number of robot axes

Syntax

`$NUM_AX=`*Number*

Explanation of the syntax

Element	Description
Number	Type: INT • **1 … 6**

Example

Robot with 6 axes

```
$NUM_AX=6
```

3.179 $NUM_IN

Description

Number of available digital inputs $IN

Writability

The system variable is write-protected.

Syntax

Number = `$NUM_IN`

Explanation of the syntax

Element	Description
Number	Type: INT

3.180 $NUM_OUT

Description

Number of available digital outputs $OUT

Writability

The system variable is write-protected.

Syntax

Number = `$NUM_OUT`

Explanation of the syntax

Element	Description
Number	Type: INT

3.181 $ON_PATH

Description

Signal declaration for monitoring of the programmed path

This output is set after the BCO run. The robot controller thus communicates to the higher-level controller that the robot is located on the programmed path. The output is reset again only if the robot leaves the path, the program is reset or block selection is carried out.

Syntax

`SIGNAL $ON_PATH $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **997**

3.182 $ORI_CHECK

Description

Orientation check at the end point of CP movements

For 5-axis robots there are 2 possibilities of approaching a Cartesian position at the end point of a CP movement. The difference between them is a 180° shift in the orientation angle C.

Syntax

`$ORI_CHECK=`*State*

Explanation of the syntax

Element	Description
State	Type: INT • **0**: The orientation at the end point is not checked (default setting for 6-axis robots). • **1**: A check is made to see if the taught end point has been reached (default setting for 5-axis robots).

3.183 $ORI_TYPE

Description

Orientation control of a CP motion in the advance run

Syntax

`$ORI_TYPE =` *type*

Explanation of the syntax

Element	Description
type	Type: ENUM ORI_TYPE • #CONSTANT: the orientation of the TCP remains constant during the motion. • #VAR: the orientation of the TCP changes continuously during the motion. • #JOINT: the orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles. **Note:** If $ORI_TYPE = #JOINT, the variable $CIRC_TYPE is ignored.

3.184 $ORI_TYPE_C

Description

Orientation control of a CP motion in the main run

Writability

The system variable is write-protected.

Syntax

type = `$ORI_TYPE_C`

Explanation of the syntax

Element	Description
type	Type: ENUM ORI_TYPE • #CONSTANT: the orientation of the TCP remains constant during the motion. • #VAR: the orientation of the TCP changes continuously during the motion. • #JOINT: the orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles. **Note:** If $ORI_TYPE_C = #JOINT, the variable $CIRC_TYPE_C is ignored.

3.185 $OUT

Description

Value of a digital output

The system variable can be used to set a digital output in the robot program and then reset it again. Furthermore, the variable can be used to request the current value of a digital output in the robot program or via the variable display function.

Syntax

`$OUT[`*output number*`]` = *value*

Explanation of the syntax

Element	Description
output number	Type: INT • **1 … 8192** **Note:** The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be requested via $NUM_IN/ $NUM_OUT.
value	Type: BOOL • TRUE or FALSE

3.186 $OUT_C

Description

Setting of digital outputs in the main run

The system variable can be used to set or reset a digital output relative to the main run. The user can use the variable, for example, in order to set a digital output at the target point of an exact positioning motion or at the vertex of an approximate positioning motion.

Syntax

`$OUT_C[`*output number*`] = `*value*

Explanation of the syntax

Element	Description
output number	Type: INT • **1 … 8192** **Note:** The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be requested via $NUM_IN/ $NUM_OUT.
value	Type: BOOL • TRUE or FALSE

3.187 $OUT_NODRIVE

Description

Defines whether actuation of an enabling switch is required for setting an output.

Syntax

`$OUT_NODRIVE = `*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: outputs can be set without pressing the enabling switch. • FALSE: the enabling switch has to be pressed in order to set outputs. Default: FALSE

3.188 $OUTSIM_TBL

Description

Simulation of outputs

- To simulate an output, it must be set to the desired simulated state (TRUE or FALSE).

- Simulation must also be activated.
  (>>> *3.129 "$IOSIM_OPT" Page 90*)

Some outputs are write-protected and cannot be simulated.

Simulated outputs are labeled with “SIM” in the KUKA System Software while write-protected ones are labeled with “SYS”. The display is called via the main menu:

- **Display** > **Inputs/outputs** > **Digital I/O**

Precondition

- The I/O simulation can only be used with an Office PC (OPS).

Syntax

`$OUTSIM_TBL[`*output number*`] = `*state*

Explanation of the syntax

Element	Description
output number	Type: INT
state	Type: CHAR • **"-"**: the output is not simulated. • **"1"**: the output is TRUE (simulated). • **"0"**: the output is FALSE (simulated). Default: **"-"**

Properties

- If simulation is activated, the robot controller takes simulated outputs into account.
- If an output[x] is simulated, its real state (i.e. $OUT[x]) can no longer be modified. To allow this, the simulated state of the output must first be reset.
- When simulation is deactivated again:
  - All outputs resume the state they had prior to simulation.
- When the robot controller is rebooted:
  - Simulation is automatically deactivated.
  - The simulated state of each output is reset.

Example 1

State before simulation: $OUT[8] == FALSE

1. The simulated state of the output is set to TRUE. ($OUTSIM_TBL[8] = "1")
2. Simulation is activated. ($IOSIM_OPT = TRUE)
   The real state now reflects the simulated state, i.e. $OUT[8] == TRUE. $OUT[8] can no longer be modified.
3. Simulation is deactivated. ($IOSIM_OPT = FALSE)
   Now $OUT[8]==FALSE!

Deactivation of the simulation has reset $OUT[8] to the state it had before the simulation, i.e. FALSE. $OUT[8] can now be modified again.

Example 2

State before simulation: $OUT[9] == FALSE.

Furthermore, $OUTSIM_TBL[9] == "-", i.e. the output is not simulated.

1. Simulation is activated. ($IOSIM_OPT = TRUE)
2. The real state of the output is changed to TRUE. ($OUT[9] = TRUE)
3. Simulation is deactivated again. ($IOSIM_OPT = FALSE)
   Now $OUT[9] == FALSE!

Deactivation of the simulation has reset $OUT[9] to the state it had before the simulation, i.e. FALSE.

3.189 $OV_ACT

Description

Current actual override

$OV_ACT can be used to read the current actual override for program mode.

- The controller readjusts the actual override $OV_ACT to the setpoint override $OV_ROB in order to prevent excessively abrupt changes in velocity when the setpoint override is modified.
- There are internal override reductions, e.g. T1 override reduction or singularity strategies, whose effects are not represented in $OV_ROB. These cause the actual override to remain below the setpoint override for a longer time.
- If no internal override reductions are active, $OV_ACT adjusts to $OV_ROB.
  (>>> *3.192 "$OV_ROB" Page 125*)

Writability

The system variable is write-protected.

Syntax

override = `$OV_ACT`

Explanation of the syntax

Element	Description
override	Type: INT; unit: % • **0 … 100** Default: **100**

3.190 $OV_APPL

$VEL_APPL can be used to modify the process velocity in the main run (i.e. during motion execution). The originally planned profile represents the upper limit and cannot be exceeded.
$VEL_APPL is particularly suitable for processes in which the required velocity only becomes known during program execution. Example: Sensor processes in which the velocity must be varied depending on the circumstances detected by the sensor.
Unlike $OV_APPL, the acceleration and braking ramps do not change with $VEL_APPL.
Available from KSS/VSS 8.6.8 and KSS/VSS 8.7.2.
(>>> *3.336 "$VEL_APPL: changing the velocity of a CP motion in the main run" Page 223*)

Description

Application override

$OV_APPL can be used to reduce the effective override applied when motions are executed in program mode. $OV_APPL is specified as a percentage and multiplied as a factor by $OV_PRO in order to determine this effective override.

The override reduction immediately affects the velocity of the current and all subsequent motions.

To determine the effective override, the percentage value of $RED_VEL is also multiplied as a factor by $OV_PRO. Internal override reductions are additionally incorporated into the calculation.
Further information: (>>> *3.192 "$OV_ROB" Page 125*)

Writability

The system variable can be written via any interpreter. In the robot interpreter, the system variable triggers an advance run stop. The advance run stop can be prevented by means of a preceding CONTINUE.

Syntax

`$OV_APPL =` *override*

Explanation of the syntax

Element	Description
override	Type: REAL; unit: % • **0 … 100** Default: **100**

Example

```
PTP XP1

$RED_VEL = 100

$OV_APPL = 50
TRIGGER WHEN DISTANCE = 0 DELAY = 200 DO $OV_APPL = 30

PTP XP2
```

The motion XP1 is executed at 100% of the displayed program override $OV_PRO. The override reduction programmed with $OV_APPL takes immediate effect. The motion XP2 is initially executed at 50% of the displayed program override $OV_PRO. The override is reduced to 30% while the motion is still being executed.

3.191 $OV_PRO

Description

Program override

$OV_PRO can be used to modify and read the program override displayed on the KUKA smartHMI user interface.

The program override is a reduction factor for the velocity of the robot during program execution which can also be modified during the program runtime. The program override is specified as a percentage. All motions in the program are executed more slowly than programmed at a velocity reduced by this factor.

Syntax

`$OV_PRO =` *override*

Explanation of the syntax

Element	Description
override	Type: INT; unit: % • **0 … 100** Default: **100**

3.192 $OV_ROB

Description

Current setpoint override

$OV_ROB can be used to read the current setpoint override for program mode. The setpoint override is composed of the following elements:

- The components $OV_APPL, $OV_PRO and $RED_VEL programmable by the user
- Internal override reductions which in certain situations are specified by the system automatically (e.g. warm-up or reduced velocity specified by the safety controller)

If no internal override reductions are in effect, the following applies:

- $OV_ROB = $OV_PRO · ($RED_VEL_C/100) · ($OV_APPL/100)

Writability

The system variable is write-protected.

Syntax

override `= $OV_ROB`

Explanation of the syntax

Element	Description
override	Type: INT; unit: % • **0 … 100**

3.193 $PAL_MODE

Description

Activation of palletizing mode (only relevant for palletizing robots)

Depending on the robot type, palletizing mode must either be explicitly activated for this robot or is already activated:

- In the case of palletizing robots with 6 axes, palletizing mode is deactivated by default and must be activated. If palletizing mode is active, axis A4 may be locked at 0° and the mounting flange is held parallel to the floor by keeping A5 at a suitable angle. For a 6-axis robot, active palletizing mode is deactivated again after a cold restart of the robot controller.
- In the case of palletizing robots with 4 or 5 axes, palletizing mode is active by default.
  - For 5-axis robots, palletizing mode can be deactivated via $PAL_MODE.
  - For 4-axis robots, palletizing mode cannot be deactivated via $PAL_MODE.

Further information about activating palletizing mode can be found in the Operating and Programming Instructions for System Integrators for the KUKA System Software (KSS or VSS).

Syntax

`$PAL_MODE=`*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Palletizing mode is active. • FALSE: Palletizing mode is not active. Default: Dependent on the robot model

3.194 $PATHTIME

Description

Structure with the data of a time-based spline motion (TIME_BLOCK)

The variable can be used to read the data of a time-based spline. $PATHTIME is filled with the data as soon as the robot controller has completed the planning of the spline block. The data are retained until the next spline block has been planned.

Syntax

Path times`=$PATHTIME`

Explanation of the syntax

Element	Description
Path times	Type: Pathtime_Struc Data of the time-based spline motion

Pathtime_Struc

```
STRUC Pathtime_Struc REAL total, scheduled, programmed,
INT n_sections, REAL max_dev, INT max_dev_section
```

Element	Description
total	Time actually required for the entire spline block (s)
scheduled	Overall time planned for the time block (s)
programmed	Overall time programmed for the time block (s)
n_sections	Number of time components (= TIME_BLOCK_PART lines)
max_dev	Maximum deviation of all time components between the programmed time and the planned time (%)
max_dev_ section	Number of the time component with the greatest deviation between the programmed time and the planned time

3.195 $PERI_RDY

Description

Signal declaration for Drives ON

By setting this output, the robot controller communicates to the higher-level controller the fact that the intermediate circuit is fully charged and that the robot drives are ready.

Syntax

```
SIGNAL $PERI_RDY $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1012**

3.196 $POS_ACT

Description

Current Cartesian robot position

The variable of structure type E6POS defines the setpoint position of the TCP in relation to the currently set BASE coordinate system.

The variable is write-protected. In the robot program, the variable triggers an advance run stop.

Syntax

```
$POS_ACT = Cartesian position
```

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

3.197 $POS_ACT_MES

Description

Measured Cartesian robot position

The variable of structure type E6POS defines the actual position of the TCP in relation to the currently set BASE coordinate system.

Syntax

`$POS_ACT_MES` = *Cartesian position*

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

3.198 $POS_BACK

Description

Cartesian start position of the current motion block

The variable defines the start position of the TCP in relation to the currently selected BASE coordinate system.

$POS_BACK can be used to return to the start position of an interrupted motion instruction. $POS_BACK corresponds to the beginning of the window for an interruption within the approximation window and to the end of the window for an interruption after the approximation window.
$POS_BACK triggers an advance run stop in the KRL program.

Syntax

`$POS_BACK` = *Cartesian position*

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

Example

Approximated PTP motion

```
PTP  P1
PTP  P2  C_PTP
PTP  P3
```

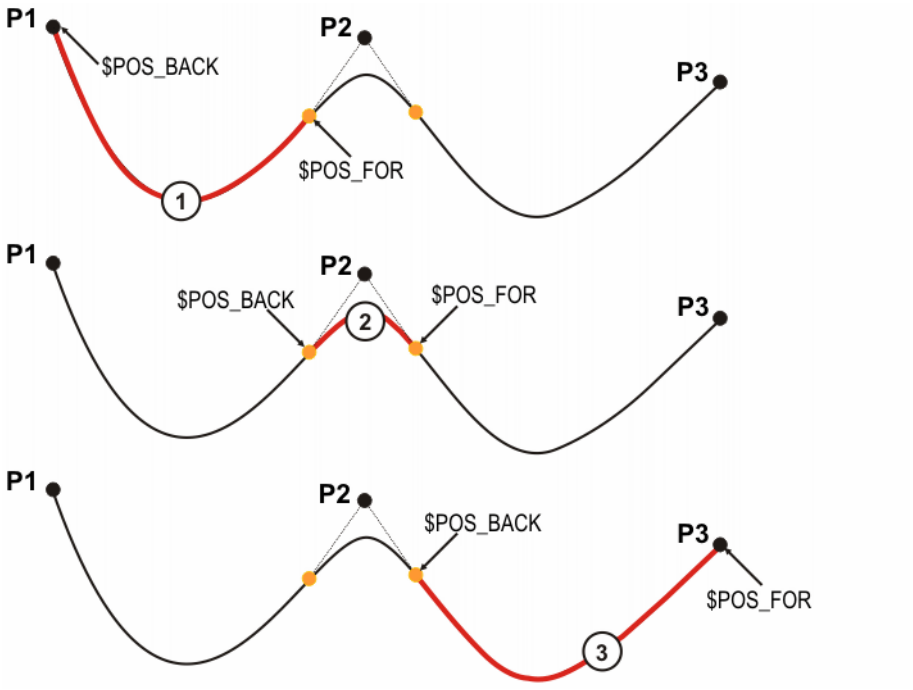


Fig. 3-8: $POS_BACK, $POS_FOR – P2 is approximated

1 Single block
2 Intermediate block
3 Following block

3.199 $POS_FOR

Description

Cartesian target position of the current motion block

The variable defines the end position of the TCP in relation to the currently selected BASE coordinate system.

$POS_FOR can be used to move to the target position of an interrupted motion instruction. $POS_FOR corresponds to the end of the window for an interruption within the approximation window and to the beginning of

the window for an interruption before the approximation window. $POS_FOR triggers an advance run stop in the KRL program.

Syntax

`$POS_FOR =` *Cartesian position*

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

Example

Approximated PTP motion

```
PTP P1
PTP P2 C_PTP
PTP P3
```

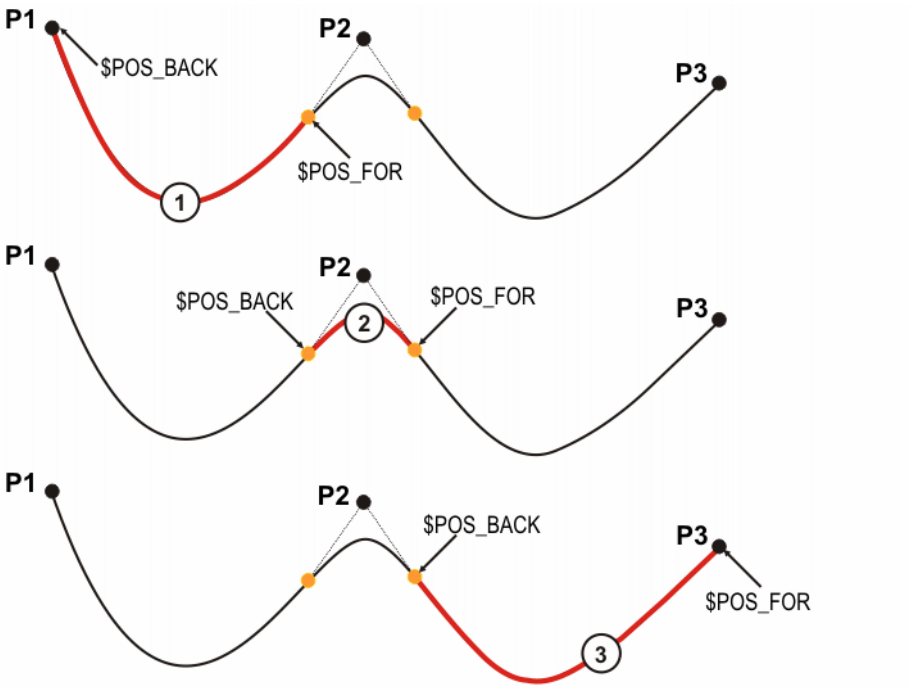


Fig. 3-9: $POS_BACK, $POS_FOR – P2 is approximated

1 Single block
2 Intermediate block
3 Following block

3.200 $POS_INT

Description

Cartesian robot position in the case of an interrupt

The variable of structure type E6POS defines the position of the TCP in relation to the BASE coordinate system at the time of the interrupt.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

$POS_INT can be used to return to the Cartesian position at which an interrupt was triggered. The variable is only admissible in an interrupt program and triggers an advance run stop.

Syntax

`$POS_INT =` *Cartesian position*

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

3.201 $POS_RET

Description

Cartesian robot position when leaving the path

The variable defines the position of the TCP in relation to the currently selected BASE coordinate system at the time that the programmed path was left.

When the robot is stationary, $POS_RET can be used to return to the Cartesian position at which the path was left.

Syntax

`$POS_RET =` *Cartesian position*

Explanation of the syntax

Element	Description
Cartesian position	Type: E6POS • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°] • **S**, **T**: status and turn for unambiguous definition of the axis positions • **E1 ... E6**: angle values or translation values of the external axes 7 to 12 (if present)

3.202 $POWER_FAIL

Description

Display of power failure

Writability

The system variable is write-protected.

Syntax

state = `$POWER_FAIL`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: power failure • FALSE: no power failure

Example

```
IF ($POWERFAIL == TRUE)
THEN
$OUT[APPLICATION_RUN] = FALSE ; Indicate that the
application has been stopped
ENDIF
```

3.203 $POWEROFF_DELAYTIME

Description

Wait time for shutdown of the robot controller
The robot controller is shut down after the time set here.

Syntax

`$POWEROFF_DELAYTIME=`*Wait time*

Explanation of the syntax

Element	Description
Wait time	Type: INT; unit: s • **1 … 30,000** • **0**: The robot controller is shut down despite external power supply. Default: **3**

3.204 $PR_MODE

Description

Signal declaration for programming mode
This output is set if operating mode T1 or T2 is selected and no program is running.

Syntax

`SIGNAL $PR_MODE $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **138**

3.205 $PRO_ACT

Description

Signal declaration for active process

This output is set whenever a process 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.

In the event of an error stop, a distinction must be made between the following possibilities:

- If interrupts have been activated but not processed at the time of the error stop, the process is regarded as inactive ($PRO_ACT=FALSE).
- If interrupts have been activated and processed at the time of the error stop, the process is regarded as active ($PRO_ACT=TRUE) until the interrupt program is completed or a STOP occurs in it ($PRO_ACT=FALSE).
- If interrupts have been activated and a STOP occurs in the program, the process is regarded as inactive ($PRO_ACT=FALSE). If, after this, an interrupt condition is met, the process is regarded as active ($PRO_ACT=TRUE) until the interrupt program is completed or a STOP occurs in it ($PRO_ACT=FALSE).

Syntax

```
SIGNAL $PRO_ACT $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1021**

3.206 $PRO_I_O

Description

Name of the system submit program

At a cold start of the robot controller, the submit interpreter automatically starts the program specified here (by default: "/R1/SPS.SUB"). It can be modified by modifying the value of $PRO_I_O_SYS.MODULE[] in STEU/MADA/$CUSTOM.DAT.

Writability

The system variable is write-protected.

Syntax

```
modulename = $PRO_I_O[]
```

Explanation of the syntax

Element	Description
modulename	Type: CHAR[] Directory and name of the system submit program (max. 64 characters)

3.207 $PRO_I_O_PROC_ID3…9

Description

The program to be started at a cold start of the robot controller for extended submit interpreters 1…7

A submit program can be assigned to every extended submit interpreter that is to be started at a cold start of the robot controller. A configuration window is provided for this on the smartHMI. The variable is written by the assignment.

The variable is of structure type PRO_IO_T. It can be read by both a robot program and a submit program. Data can also be written to it via the variable display.

Precondition

- Robot controller in Multi-Submit mode

Syntax

`$PRO_I_O_PROC_ID`*Index*`={MODULE[] "`*Name*`", COLD_BOOT_RUN` *Cold start*`}`

Explanation of the syntax

Element	Description
Index	Type: INT Number of the extended submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • … • 9: Extended submit interpreter 7
Name	Type: CHAR Directory and name of the submit program: max. 64 characters
Cold start	Type: ENUM SW_ONOFF Start of the submit program at a cold start • #ON: Submit program is started. • #OFF: Submit program is not started.

3.208 $PRO_I_O_SYS

Description

The program to be started at a cold start of the robot controller for the system submit interpreter

At a cold start of the robot controller, the system submit interpreter automatically starts the program specified here. By default, this is SPS.SUB.

By default, the program SPS.SUB is assigned to the system submit interpreter. It is strongly recommended not to change the assignment. It is necessary to enable the controller-internal submit tasks to be executed. If the assignment is changed, this can affect the functionality of technology packages and other options.

The variable is of structure type PRO_IO_T. It can be read by both a robot program and a submit program. Data can also be written to it via the variable display.

Precondition

- Robot controller in Multi-Submit mode

Syntax

```
$PRO_I_O_SYS={MODULE[] "Name", COLD_BOOT_RUN Cold start}
```

Explanation of the syntax

Element	Description
Name	Type: CHAR Directory and name of the submit program: max. 64 characters
Cold start	Type: ENUM SW_ONOFF Start of the submit program at a cold start • #ON: Submit program is started. • #OFF: Submit program is not started.

Example

Default entry for the system submit interpreter

```
DECL PRO_IO_T $PRO_I_O_SYS={MODULE[] "/R1/SPS()",
COLD_BOOT_RUN #ON}
```

3.209 $PRO_IP

Description

Process pointer information of an interpreter

The variable indicates the process pointer information of an interpreter subject to the context in accordance with the following rules:

- The process pointer information of the robot interpreter is indicated when the variable is read in a robot program.
- The process pointer information of the relevant submit interpreter is indicated when the variable is read in a submit program. In this case, only the components Name, Snr and I_Executed are set. The variable $PRO_IP1 can be used to read the process pointer information of the robot interpreter from a submit program.
- When the variable is read via the variable display, the process pointer information of the interpreter that is currently selected via $INTERPRETER is indicated. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

A more up-to-date variant is the system variable $PRO_IP_EXT[], which can also display hidden modules and up to 20 caller stacks:
(>>> *3.211 "$PRO_IP_EXT" Page 139*)

Writability

The system variable is write-protected.

Syntax

processdata = $PRO_IP

Explanation of the syntax

Element	Description
processdata	Type: STRUC PRO_IP Structure with the current data of the process pointer

PRO_IP

```
STRUC PRO_IP CHAR NAME[32], INT SNR, CHAR NAME_C[32],
INT SNR_C, BOOL I_EXECUTED, INT P_ARRIVED, CHAR
P_NAME[24], CALL_STACK SI01, SI02, ...SI10
```

Element	Description
NAME[]	Type: CHAR[] Name of the module in which the interpreter is in the advance run
SNR	Type: INT Number of the block in which the interpreter is in the advance run (usually not equal to the line number of the program)
NAME_C[]	Type: CHAR[] Name of the module in which the interpolator is in the main run
SNR_C	Type: INT Number of the block in which the interpolator is in the main run
I_EXECUTED	Type: BOOL Indicates whether the block has already been executed by the interpreter (= TRUE)

Element	Description
P_ARRIVED	Type: INT Indicates point on the path where the robot is located (only relevant for motion instructions) • **0**: Arrived at the target or auxiliary point of the motion • **1**: Target point not reached (robot is somewhere on the path) • **2**: Not relevant • **3**: Arrived at the auxiliary point of a CIRC or SCIRC motion • **4**: On the move in the section between the start and the auxiliary point
P_NAME[]	Type: CHAR[] Name or aggregate of the target or auxiliary point at which the robot is located
SI01 … SI10	Type: CALL_STACK Caller stack in which the interpreter is situated • SI01: 1st subprogram level • SI02: 2nd subprogram level

CALL_STACK

```
STRUC CALL_STACK CHAR NAME[32], INT SNR, T_FLAG
INT_FLAG
```

Element	Description
NAME[]	Type: CHAR[] Name of the module in which the interpreter is located
SNR	Type: INT Number of the block in which the interpreter is located (usually not equal to the line number of the program)
INT_FLAG	Type: T_FLAG If the caller stack in which the interpreter is situated is an interrupt program, flag of the interrupt No interrupt active: INT_FLAG 0

3.210 $PRO_IP1

Description

Process pointer information of the robot interpreter

The variable can be read by means of both a robot program and a submit program. Data can also be written to it via the variable display.

Syntax

processdata = `$PRO_IP1`

Explanation of the syntax

Element	Description
processdata	Type: STRUC PRO_IP Structure with the current data of the process pointer

PRO_IP

```
STRUC PRO_IP CHAR NAME[32], INT SNR, CHAR NAME_C[32],
INT SNR_C, BOOL I_EXECUTED, INT P_ARRIVED, CHAR
P_NAME[24], CALL_STACK SI01, SI02, ...SI10
```

Element	Description
NAME[]	Type: CHAR[] Name of the module in which the interpreter is in the advance run
SNR	Type: INT Number of the block in which the interpreter is in the advance run (usually not equal to the line number of the program)
NAME_C[]	Type: CHAR[] Name of the module in which the interpolator is in the main run
SNR_C	Type: INT Number of the block in which the interpolator is in the main run
I_EXECUTED	Type: BOOL Indicates whether the block has already been executed by the interpreter (= TRUE)
P_ARRIVED	Type: INT Indicates point on the path where the robot is located (only relevant for motion instructions) • **0**: Arrived at the target or auxiliary point of the motion • **1**: Target point not reached (robot is somewhere on the path) • **2**: Not relevant • **3**: Arrived at the auxiliary point of a CIRC or SCIRC motion • **4**: On the move in the section between the start and the auxiliary point
P_NAME[]	Type: CHAR[] Name or aggregate of the target or auxiliary point at which the robot is located
SI01 … SI10	Type: CALL_STACK Caller stack in which the interpreter is situated • SI01: 1st subprogram level • SI02: 2nd subprogram level

3.211 $PRO_IP_EXT

Description

Caller stack of the robot interpreter

The system variable **$PRO_IP_EXT** (“extended information”) is a further development of **$PRO_IP** (>>> *3.209 "$PRO_IP" Page 135*). Unlike **$PRO_IP**, it also displays hidden modules and caller stacks down to the 20th subprogram level.

Writability

The system variable is write-protected.

Syntax

module = `$PRO_IP_EXT[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Index of the element in the caller stack • **1 … 20 modules**
Missing in-line element 'cite'.	Type: STRUC T_CS

STRUC T_CS

```
STRUC T_CS CHAR MODULE_NAME[32], UP_NAME[32], INT SNR,
INTERRUPT_LEVEL, MODULE_HIDDEN
```

Element	Description
MODUL_NAME[]	Type: CHAR[] Name of the module in which the robot interpreter is in the advance run
UP_NAME[]	Type: CHAR[] Name of the subprogram in which the robot interpreter is in the advance run
SNR	Type: INT Number of the block in which the robot interpreter is in the advance run
INTERRUPT_ LEVEL	Type: INT Interrupt level
MODUL_ HIDDEN	Type: INT Flag indicating that the module is hidden

3.212 $PRO_IPS0 ... $PRO_IPS7: Displaying the process pointer information of a submit interpreter

Description

Process pointer information of a submit interpreter

The variables stand for the following submit interpreters:

- $PRO_IPS0: Main submit
- $PRO_IPS1: Extended submit interpreter 1
- $PRO_IPS2: Extended submit interpreter 2
- $PRO_IPS3: Extended submit interpreter 3
- $PRO_IPS4: Extended submit interpreter 4
- $PRO_IPS5: Extended submit interpreter 5
- $PRO_IPS6: Extended submit interpreter 6
- $PRO_IPS7: Extended submit interpreter 7

Declaration

```
PRO_IP $PRO_IPS0
PRO_IP $PRO_IPS1
PRO_IP $PRO_IPS2
PRO_IP $PRO_IPS3
PRO_IP $PRO_IPS4
PRO_IP $PRO_IPS5
PRO_IP $PRO_IPS6
PRO_IP $PRO_IPS7
```

Explanation

Element	Description
$PRO_IPS0	Type: PRO_IP Structure with the current data of the process pointer

Data type PRO_IP

```
STRUC PRO_IP CHAR NAME[32], INT SNR, BOOL I_EXECUTED,
CALL_STACK SI01, SI02, …SI10
```

Element	Description
NAME[]	Data type: CHAR[] Name of the module in which the interpreter is located
SNR	Data type: INT Number of the block in which the interpreter is located (usually not equal to the line number of the program)
I_EXECU-TED	Data type: BOOL Indicates whether the block has already been executed by the interpreter (= TRUE)
SI01 … SI10	Data type: CALL_STACK Caller stack in which an interpreter is situated • SI01: 1st subprogram level • SI02: 2nd subprogram level

Data type CALL_STACK

```
STRUC CALL_STACK CHAR NAME[32], INT SNR, T_FLAG
INT_FLAG
```

Element	Description
NAME[]	Data type: CHAR[] Name of the module in which the interpreter is in the advance run
SNR	Data type: INT Number of the block in which the interpreter is in the advance run (usually not equal to the line number of the program)
INT_FLAG	Data type: T_FLAG If the caller stack in which the interpreter is situated is an interrupt program, flag of the interrupt No interrupt active: INT_FLAG 0

3.213 $PRO_MODE

Description

Program run mode of an interpreter

The variable indicates the program run mode of an interpreter subject to the context in accordance with the following rules:

- The program run mode of the robot interpreter is indicated when the variable is read in a robot program.
- The program run mode of the relevant submit interpreter is indicated when the variable is read in a submit program.
- When the variable is read via the variable display, the program run mode of the interpreter that is currently selected via $INTERPRETER is indicated. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

Syntax

mode = `$PRO_MODE`

Explanation of the syntax

Element	Description
mode	Type: ENUM PRO_MODE • #GO: The program is executed through to the end without stopping. • #MSTEP: Motion Step The program is executed with a stop at each motion point, including auxiliary points and the points of a spline segment. The program is executed without advance processing. • #ISTEP: Incremental Step The program is executed with a stop after each program line. The motion is also stopped after program lines that cannot be seen and after blank lines. The program is executed without advance processing. • #BSTEP: Backward motion This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode. The response is like that for #MSTEP. • #PSTEP: Program Step The program is executed step by step without advance processing. Subprograms are executed completely. • #CSTEP: Continuous Step Approximate positioning points are executed with advance processing and approximated. Exact positioning points are executed without advance processing and with a stop after the motion instruction.

3.214 $PRO_MODE1

Description

Program run mode of the robot interpreter

Syntax

mode = `$PRO_MODE1`

Explanation of the syntax

Element	Description
mode	Type: ENUM PRO_MODE • #GO: The program is executed through to the end without stopping. • #MSTEP: Motion Step The program is executed with a stop at each motion point, including auxiliary points and the points of a spline segment. The program is executed without advance processing. • #ISTEP: Incremental Step The program is executed with a stop after each program line. The motion is also stopped after program lines that cannot be seen and after blank lines. The program is executed without advance processing. • #BSTEP: Backward motion This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode. The response is like that for #MSTEP. • #PSTEP: Program Step The program is executed step by step without advance processing. Subprograms are executed completely. • #CSTEP: Continuous Step Approximate positioning points are executed with advance processing and approximated. Exact positioning points are executed without advance processing and with a stop after the motion instruction.

3.215 $PRO_MOVE

Description

Signal declaration for active program motion

This output is set whenever a synchronous axis moves (also in jog mode). The signal is thus the inverse of $ROB_STOPPED.

Syntax

`SIGNAL $PRO_MOVE $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1022**

3.216 $PRO_NAME

Description

Name of the selected module

The variable can be used to query which module is selected in the interpreter that is currently selected via $INTERPRETER. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

Writability

The system variable is write-protected.

Syntax

modulename = `$PRO_NAME[]`

Explanation of the syntax

Element	Description
modulename	Type: CHAR[] Name of the selected module (max. 24 characters)

3.217 $PRO_NAME1

Description

Name of the module selected in the robot interpreter

The variable can be used to query which module is currently selected in the robot interpreter.

Writability

The system variable is write-protected.

Syntax

modulename = `$PRO_NAME1[]`

Explanation of the syntax

Element	Description
modulename	Type: CHAR[] Name of the module selected in the robot interpreter (max. 24 characters)

3.218 $PRO_STATE

Description

Program status of an interpreter

The variable indicates the status of an interpreter subject to the context in accordance with the following rules:

- The status of the robot interpreter is indicated when the variable is read in a robot program.
- The status of the relevant submit interpreter is indicated when the variable is read in a submit program.
- When the variable is read via the variable display, the status of the interpreter that is currently selected via $INTERPRETER is indicated. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

Writability

The system variable is write-protected.

Syntax

state = `$PRO_STATE`

Explanation of the syntax

Element	Description
state	Type: ENUM PRO_STATE An interpreter can have the following states: • #P_ACTIVE: program is selected and is running. • #P_FREE: program is deselected. • #P_END: program is selected and the end of the program has been reached. • #P_RESET: program is selected and has been stopped and reset. • #P_STOP: program is selected and has been stopped.

3.219 $PRO_STATE0: Multi-Submit

Description

Group indicator for the program status of the submit interpreters

The status bar of the KUKA smartHMI contains a group indicator for the status of the submit interpreters. $PRO_STATE0 reflects the group indicator.

Precondition

- Robot controller in Multi-Submit mode

	For a robot controller in Single Submit mode, the variable reflects the program status in the submit interpreter.

Writability

The system variable is write-protected.

Syntax

state = `$PRO_STATE0`

Explanation of the syntax

Element	Description
state	Type: ENUM PRO_STATE • #P_ACTIVE: at least 1 submit program is selected and is running. • #P_FREE: all submit programs are deselected. • #P_RESET: at least 1 submit program is selected and has been stopped and reset. No submit interpreter is in the #P_ACTIVE or #P_STOP state. • #P_STOP: at least 1 submit program is selected and has been stopped. No submit interpreter is in the #P_ACTIVE state.

3.220 $PRO_STATE1

Description

Program status of the robot interpreter

Writability

The system variable is write-protected.

Syntax

state = `$PRO_STATE1`

Explanation of the syntax

Element	Description
state	Type: ENUM PRO_STATE A robot interpreter can have the following states: • #P_ACTIVE: robot program is selected and is running. • #P_FREE: robot program is deselected. • #P_END: robot program is selected and the end of the program has been reached. • #P_RESET: robot program is selected and has been stopped and reset. • #P_STOP: robot program is selected and has been stopped.

3.221 $PROG_INFO

Description

The variable $PROG_INFO[] groups together the states of all interpreters. Each array element has the STRUCTURE type PROG_INFO and refers to an interpreter.

Writability

The system variable is write-protected.

Syntax

proginfo = `$PROG_INFO[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Index of the interpreter • 1: Robot interpreter • 2: System submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • Etc.
proginfo	Type: STRUC PROG_INFO State of the relevant interpreter

PROG_INFO

```
STRUC PROG_INFO CHAR SEL_NAME[32], PRO_STATE P_STATE,
PRO_MODE P_MODE, CHAR PRO_IP_NAME[32], INT PRO_IP_SNR
```

Element	Description
SEL_NAME[]	Type: CHAR[] Name of the selected program
P_STATE	Type: ENUM PRO_STATE Program status • #P_ACTIVE: Program is selected and is running. • #P_FREE: Program is deselected. • #P_END: Program is selected and the end of the program has been reached. • #P_RESET: Program is selected and has been stopped and reset. • #P_STOP: Program is selected and has been stopped.
P_MODE	Type: ENUM PRO_MODE Program run mode • #GO: The program is executed through to the end without stopping. • #MSTEP: Motion Step The program is executed with a stop at each motion point, including auxiliary points and the points of a spline segment. The program is executed without advance processing. • #ISTEP: Incremental Step The program is executed with a stop after each program line. The motion is also stopped after program lines that cannot be seen and after blank lines. The program is executed without advance processing. • #BSTEP: Backward motion This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode. The response is like that for #MSTEP. • #PSTEP: Program Step The program is executed step by step without advance processing. Subprograms are executed completely. • #CSTEP: Continuous Step Approximate positioning points are executed with advance processing and approximated. Exact positioning points are executed without advance processing and with a stop after the motion instruction.
PRO_IP_NAME[]	Type: CHAR[] Name of the current module
PRO_IP_SNR	Type: INT Block number in which the interpreter is currently located

Example

```
DEF myProgr()
...
WAIT FOR $PROG_INFO[4].P_STATE == #P_ACTIVE
```

Meaning: Wait until extended submit 2 has selected and started a program.

3.222 $RAMDISK_TOTAL_CAPACITY

Description

This variable can be used to read the size of the RAM disk of the real-time system.

Writability

The system variable is write-protected.

Syntax

capacity = $RAMDISK_TOTAL_CAPACITY

Explanation of the syntax

Element	Description
capacity	Type: INT Total size of the fast-boot RAM disk Unit: Byte

Example

```
capacity = $RAMDISK_TOTAL_CAPACITY
```

capacity = 75335680

Varies according to the software version.

3.223 $RAMDISK_FREE_CAPACITY

Description

This variable can be used to read the size of the free memory space on the RAM disk of the real-time system.

Writability

The system variable is write-protected.

Syntax

capacity = $RAMDISK_FREE_CAPACITY

Explanation of the syntax

Element	Description
capacity	Type: INT RAM disk space currently free Unit: Byte

3.224 $RCV_INFO

Description

Version identifier of the kernel system

Writability

The system variable is write-protected.

Syntax

`$RCV_INFO[]="`*Identifier*`"`

Explanation of the syntax

Element	Description
Identifier	Type: CHAR Version identifier: max. 128 characters

Example

```
$RCV_INFO[]="KS V8.2.111(krc1adm@deau1svr12pt-06) 1 Thu 29
Mar 2012 10:34:13 RELEASE"
```

The identifier consists of the following components:

- Kernel system version: KS V8.2.111
- Name of author: krc1adm
- Name of computer: deau1svr12pt-06
- Date and time of compilation: 29 March 2012 at 10.34 a.m.

3.225 $RED_ACC_ECO_LEVEL[]: Reduction factor for acceleration

Description

Reduction factors for accelerations in order to parameterize the robot more energy-efficiently, regardless of the currently selected $ECO_LEVEL (>>> *3.82 "$ECO_LEVEL" Page 61*). If a value less than 100% is configured for $RED_ACC_ECO_LEVEL[], this value is applied additionally by multiplication to the reduction factors resulting from $ECO_LEVEL. To avoid a reduction in acceleration, the value must be 100%.

Declaration

```
INT $RED_ACC_ECO_LEVEL[12]
```

Explanation

Element	Description
$RED_ACC_ECO_LEVEL[i]	Data type: INT Reduction factor for accelerations [%] • **1 ... 100** • **i**: Robot axis (1-6), external axis (7-12) Default: 100%

3.226 $RED_T1_OV_CP

Description

Reduction of the path velocity for CP motions in T1

The path velocity can be reduced in the following ways:

- Reduction by the percentage $RED_T1 specified in the file …R1\Mada \$machine.dat
- Reduction to the path velocity $VEL_CP_T1 specified in the file … R1\Mada\$machine.dat

Syntax

`$RED_T1_OV_CP=`*Reduction type*

Explanation of the syntax

Element	Description
Reduction type	Type: BOOL • TRUE: Reduction by $RED_T1 • FALSE: Reduction to $VEL_CP_T1 Default: TRUE

3.227 $RED_VEL

Description

Reduction factor for program override in the advance run

$RED_VEL can be used to reduce the effective override applied when motions are executed in program mode. $RED_VEL is specified as a percentage and multiplied as a factor by $OV_PRO in order to determine this effective override.

The override reduction does not immediately affect the velocity, but rather only comes into effect with the velocity of the subsequent motions in the main run.

To determine the effective override, the percentage value of $OV_APPL is also multiplied as a factor by $OV_PRO. Internal override reductions are additionally incorporated into the calculation.
Further information: (>>> *3.192 "$OV_ROB" Page 125*)

Writability

The system variable can only be written via the robot interpreter. No advance run stop is triggered.

Syntax

```
$RED_VEL = factor
```

Explanation of the syntax

Element	Description
factor	Type: INT; unit: % • **0 … 100** Default: **100**

Example

```
$OV_APPL = 100

PTP XP1 C_PTP

$RED_VEL = 50

PTP XP2
```

The motion XP1 is approximated and executed at 100% of the displayed program override $OV_PRO. The override reduction programmed via $RED_VEL is not applied until the motion XP2 is active in the main run. It is executed at 50% of the displayed program override $OV_PRO.

3.228 $RED_VEL_C

Description

Reduction factor for program override in the main run

$RED_VEL_C can be used to read the currently effective value for $RED_VEL.

Writability

The system variable is write-protected.

Syntax

```
factor = $RED_VEL_C
```

Explanation of the syntax

Element	Description
factor	Type: INT; unit: % • **0 … 100** Default: **100**

3.229 $RED_VEL_ECO_LEVEL[]: Reduction factor for velocity

Description

Reduction factors for velocities in order to parameterize the robot more energy-efficiently, regardless of the currently selected $ECO_LEVEL (>>> *3.82 "$ECO_LEVEL" Page 61*). If a value less than 100% is config-

ured for $RED_VEL_ECO_LEVEL[], this value is applied additionally by multiplication to the reduction factors resulting from $ECO_LEVEL. To avoid a reduction in velocity, the value must be 100%.

Declaration

```
INT $RED_VEL_ECO_LEVEL[12]
```

Explanation

Element	Description
$RED_ACC_ECO_LEVEL[i]	Data type: INT Reduction factor for velocities [%] • **1 ... 100** • **i**: Robot axis (1-6), external axis (7-12) Default: 100%

3.230 $REVO_NUM

Description

Counter for infinitely rotating axes

Writability

The system variable is write-protected.

Syntax

Number = `$REVO_NUM[`*axis number*`]`

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 … 6**: Robot axis A1 ... A6 • **7 … 12**: External axis E1 ... E6
Number	Type: INT Number of revolutions

3.231 $RINT_LIST

Description

Information about an interrupt that is declared in a robot program

The information can be displayed via the variable display or by means of the diagnosis function in the main menu.

Procedure

- In the main menu, select **Diagnosis** > **Interrupts**.

	In robot and submit programs, a maximum of 32 interrupts (64 from KSS 8.5 onwards) can be declared simultaneously and up to 16 interrupts (32 from KSS 8.5 onwards) can be active at the same time.

	Further information about interrupt programming is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected.

Syntax

intinfo = $RINT_LIST[*index*]

Explanation of the syntax

Element	Description
index	Type: INT Index of the interrupt • **1 … 32 (64 from KSS 8.5 onwards)**
intinfo	Type: STRUC INT_INFO Structure with information about the interrupt

INT_INFO

```
STRUC INT_INFO INT INT_PRIO, INT_STATE, INT_TYPE,
PROG_LINE, CHAR PROG_NAME[32]
```

Element	Description
INT_PRIO	Type: INT Priority of the interrupt • **1, 2, 4 … 39** • **81 …128**
INT_STATE	Bit array for interrupt states • **Bit 0 = 1**: Interrupt is declared and activated. • **Bit 1 = 1**: Interrupt is activated and enabled. • **Bit 2 = 1**: Interrupt is globally declared.
INT_TYPE	Type: INT Type of interrupt • **0**: Standard interrupt • **1**: Interrupt due to an EMERGENCY STOP ($EMSTOP) • **2**: Interrupt due to activation of the Fast Measurement inputs ($MEAS_PULSE) • **3**: Interrupt due to an error stop ($STOPMESS) • **4**: Interrupt due to a trigger (subprogram call)
PROG_LINE	Type: INT Line number of the robot program in which the interrupt is declared
PROG_NAME[]	Type: CHAR[] Directory and name of the robot program in which the interrupt is declared

3.232 $RC_RDY1

Description

Signal declaration for operational robot controller

If the robot controller sets this output, it is ready for operation and the program can be started by the higher-level controller.

Syntax

```
SIGNAL $RC_RDY1 $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **137**

3.233 $ROB_CAL

Description

Signal declaration for robot mastering

This output is set whenever all robot axes are mastered. The output is reset as soon as a robot axis has been unmastered.

Syntax

```
SIGNAL $ROB_CAL $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1001**

3.234 $ROB_STOPPED

Description

Signal declaration for robot standstill

This output is set when the robot is at a standstill. In the event of a WAIT statement, this output is set during the wait. The signal is thus the inverse of $PRO_MOVE.

Syntax

```
SIGNAL $ROB_STOPPED $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1023**

3.235 $ROB_TIMER

Description

Clock generator for measuring program runtimes
The variable is write-protected and counts in a cycle of 1 ms.

Syntax

Number = `$ROB_TIMER`

Explanation of the syntax

Element	Description
Number	Type: INT Number of cycles

3.236 $ROBNAME

Description

Name of the controller entered in the robot data by the user

Writability

The system variable can only be modified via the variable display.

Syntax

`$ROBNAME[] = "`*name*`"`

Explanation of the syntax

Element	Description
name	Type: CHAR Controller name: max. 50 characters

3.237 $ROBROOT_C

Description

ROBROOT coordinate system in the main run
The ROBROOT coordinate system is a Cartesian coordinate system, which is always located at the robot base. The variable of structure type FRAME defines the current position of the robot in relation to the WORLD coordinate system.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

Writability

The system variable is write-protected.

3.238 $ROBROOT_KIN[]

Description

Information about the external ROBROOT kinematic system

The variable contains the name of the external kinematic system and a list of the external axes contained in the transformation. The name and the external axes contained in the transformation are defined in the machine data, e.g. $ET1_NAME and $ET1_AX.

Writability

The system variable is write-protected.

Declaration

```
CHAR $ROBROOT_KIN[49]
```

Explanation

Element	Description
$ROBROOT_KIN[]	Type: CHAR Name of the external kinematic system with a list of the external axes contained in the transformation: Maximum 49 characters

3.239 $ROBRUNTIME

Description

Operating hours meter

The operating hours meter runs when the drives are switched on and counts this time in minutes.

Writability

The system variable is write-protected.

Syntax

`$ROBRUNTIME=`*operating hours*

Explanation of the syntax

Element	Description
operating hours	Type: INT; unit: min

3.240 $ROBTRAFO

Description

Robot name

The variable contains the robot name saved on the RDC. This name must match the name of the coordinate transformation specified in the machine data (variable $TRAFONAME[] in the file …R1\Mada\$machine.dat).

Syntax

```
$ROBTRAFO[]="Name"
```

Explanation of the syntax

Element	Description
Name	Type: CHAR (maximum 32 characters)

3.241 $ROTSYS

Description

Reference coordinate system for rotation in the advance run

The variable can be used to define the coordinate system in which the rotation (A, B, C) is executed in Cartesian relative motions or in Cartesian jogging.

By default, the rotation is executed in the coordinate system which is programmed for relative motion or set in the jogging options. $ROTSYS can be used to specify that, independently of the programmed/set reference coordinate system, the rotation is to be executed in the coordinate system of the current base or in the coordinate system of the current tool.

Syntax

```
$ROTSYS = coordinate system
```

Explanation of the syntax

Element	Description
coordinate system	Type: ENUM ROTSYS • #AS_TRA: – Program mode: rotation in the coordinate system currently programmed for relative motion. – Jog mode: rotation in the coordinate system currently set in the jogging options. • #BASE: rotation in the coordinate system of the current base • #TCP: rotation in the coordinate system of the current tool Default: #AS_TRA

3.242 $ROTSYS_C

Description

Reference coordinate system for rotation in the main run

The variable contains the coordinate system in which the rotation (A, B, C) is currently executed for Cartesian relative motions or Cartesian jogging.

Writability

The system variable is write-protected.

Syntax

coordinate system = $ROTSYS_C

Explanation of the syntax

Element	Description
coordinate system	Type: ENUM ROTSYS • #AS_TRA: – Program mode: rotation in the coordinate system currently programmed for relative motion. – Jog mode: rotation in the coordinate system currently set in the jogging options. • #BASE: rotation in the coordinate system of the current base • #TCP: rotation in the coordinate system of the current tool Default: #AS_TRA

3.243 $RVM

Description

Resonance avoidance for approximated motions

If a motion is approximated and the next motion is a PTP motion with exact positioning, this can result in resonance vibrations of axis A1 if the distance between the points is too small.

If the variable $RVM is set to TRUE in the robot program, the motion time of the next motion sequence consisting of approximated motion and PTP motion with exact positioning is lengthened so that axis A1 no longer vibrates. The variable is then automatically reset to FALSE.

This variable must not be used in the Submit interpreter or in an interrupt program.

Syntax

$RVM=*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Resonance avoidance is active. • FALSE: Resonance avoidance is not active. Default: FALSE

Example

```
...
$RVM=TRUE
...
PTP P5 C_PTP
LIN P6
...
```

```
LIN P10 C_DIS
PTP P11
...
```

Resonance avoidance does not take effect until the 2nd motion sequence in this program example. It has no effect on the 1st motion sequence because the exact positioning motion to which approximate positioning is carried out is a LIN motion.

3.244 $SAFE_FS_STATE

Description

Status of the safety controller

The variable indicates whether the safety controller is running without errors. If the variable switches to TRUE, safe monitoring of the failsafe state has been activated and the safety controller is no longer operational.

Syntax

`$SAFE_FS_STATE`=State

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: The safety controller is not operable. • FALSE: The safety controller is running without errors. Default: FALSE

3.245 $SAFE_IBN

Description

Activation of Start-up mode

The variable is written to if Start-up mode is activated or deactivated via the main menu on the smartHMI.

Precondition

- Switching to Start-up mode is allowed: $SAFE_IBN_ALLOWED=TRUE

Syntax

`$SAFE_IBN`=*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Start-up mode is active. • FALSE: Start-up mode is not active. Default: FALSE

3.246 $SAFE_IBN_ALLOWED

Description

Switching to Start-up mode allowed?

The variable indicates whether switching to Start-up mode is currently allowed. Only if this is the case can Start-up mode be activated via the main menu on the smartHMI.

Writability

The system variable is write-protected.

Syntax

state = $SAFE_IBN_ALLOWED

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: Start-up mode is allowed. • FALSE: Start-up mode is not allowed. Default: FALSE

3.247 $SAFEGATE_OP

Description

Signal declaration for operator safety

This input is set when the operator safety is active, e.g. in Automatic mode with the gate closed.

In the event of a loss of signal during Automatic operation (e.g. safety gate is opened), the drives are deactivated after 1 s. The robot and external axes (optional) stop with a STOP 1. When the signal is applied again at the input (e.g. safety gate closed), automatic operation can be resumed once the corresponding message has been acknowledged.

Syntax

SIGNAL $SAFEGATE_OP $IN[*Input number*]

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.248 $SAFETY_DRIVES_ENABLED

Description

Drives enable of the safety controller

The variable indicates whether the safety controller has enabled the drives.

Syntax

```
$SAFETY_DRIVES_ENABLED=State
```

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Drives are enabled. • FALSE: Drives are not enabled. Default: FALSE

3.249 $SAFETY_SW

Description

State of the enabling switches

Syntax

```
$SAFETY_SW = state
```

Explanation of the syntax

Element	Description
state	Type: ENUM SIG_STATE • #PRESSED: an enabling switch is pressed (center position). • #RELEASED: no enabling switch is pressed or an enabling switch is fully pressed (panic position). • #INVALID: invalid state

3.250 $SEQ_CAL

Description

Mastering sequence of the axes in steps
The mastering steps must be specified in ascending order.

Syntax

```
$SEQ_CAL[Mastering step]=Bit array
```

Explanation of the syntax

Element	Description
Mastering step	Type: INT • **1 … 12**
Bit array	Bit array that specifies the axis to be mastered • **Bit n = 0**: The axis is not mastered. • **Bit n = 1**: The axis is mastered.

Bit n	11 …	5	4	3	2	1	0
Axis	E6 …	A6	A5	A4	A3	A2	A1

Example

```
$SEQ_CAL[1]='B0001'
$SEQ_CAL[2]='B0010'
$SEQ_CAL[3]='B0100'
$SEQ_CAL[4]='B1000'
$SEQ_CAL[5]='B0001 0000'
$SEQ_CAL[6]='B0010 0000'
$SEQ_CAL[7]='B0100 0000'
$SEQ_CAL[8]='B1000 0000'
```

3.251 $SEN_PINT

Description

Exchange of integer values via a sensor interface

The sensor is used to transmit integer values to a sensor via an interface or to receive integer values from a sensor.

Syntax

`$SEN_PINT[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 128**
Value	Type: INT

3.252 $SEN_PINT_C

The variable $SEN_PINT_C has no relation to the main run. It is used in exactly the same way as the variable $SEN_PINT.

Description

Exchange of integer values via a sensor interface

The sensor is used to transmit integer values to a sensor via an interface or to receive integer values from a sensor.

Syntax

`$SEN_PINT_C[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 128**
Value	Type: INT

3.253 $SEN_PREA

Description

Exchange of real values via a sensor interface

The sensor is used to transmit real values to a sensor via an interface or to receive real values from a sensor.

Syntax

`$SEN_PREA[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 128**
Value	Type: REAL

3.254 $SEN_PREA_C

The variable $SEN_PREA_C has no relation to the main run. It is used in exactly the same way as the variable $SEN_PREA.

Description

Exchange of real values via a sensor interface

The sensor is used to transmit real values to a sensor via an interface or to receive real values from a sensor.

Syntax

`$SEN_PREA_C[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 128**
Value	Type: REAL

3.255 $SERVO_SIM

Description

Simulation of the axis motions

Syntax

`$SERVO_SIM=`*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Simulation is active. • FALSE: No simulation active. Default: FALSE

3.256 $SET_IO_SIZE

This system variable is only available for reasons of compatibility. The number of digital I/Os available can be monitored directly via the variables $NUM_IN/$NUM_OUT.

Writability

The system variable is write-protected.

Syntax

Number`=$SET_IO_SIZE`

Explanation of the syntax

Element	Description
Number	Type: INT • **1**: 1 … 1 024 • **2**: 1 … 2 048 • **4**: 1 … 4 096 • **8**: 1 … 8 192 Default: **4**

3.257 $SHUTDOWN

Description

Indicates whether the robot controller is currently being shut down.

Writability

The system variable is write-protected.

Syntax

state = `$SHUTDOWN`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: controller is shut down, transition to BUSPOWERDOWN (POWER_FAIL, POWER_OFF, SOFTPOWERDOWN). • FALSE: the controller is not shut down (BUSPOWERON).

3.258 $SINGUL_DIST

Description

Standardized distance from the singularity

The variable specifies the standardized distance of the current robot position from the singularity in question.

If the standardized distance at a robot position becomes ≤1 and this position is used as a Cartesian end point of a PTP motion, the robot controller calculates the ambiguous axis angles according to the strategy defined with $SINGUL_POS in the machine data.

Writability

The system variable is write-protected.

Syntax

distance = `$SINGUL_DIST[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Index for the type of singular position: • **1**: overhead singularity $SINGUL_POS[1] • **2**: extended position singularity $SINGUL_POS[2] • **3**: wrist axis singularity $SINGUL_POS[3]
distance	Type: REAL Standardized distance from the specified singularity

3.259 $SINGUL_STRATEGY

Description

Strategy for singularity-free motion

Syntax

`$SINGUL_STRATEGY` = *strategy*

Explanation of the syntax

Element	Description
strategy	Type: INT • **0**: no strategy • **1**: approximation strategy (= moving through a singularity by means of changes in orientation) Default: **0**

3.260 $SINT_LIST

Description

Information about an interrupt that is declared in a submit program

The information can be displayed via the variable display or by means of the diagnosis function in the main menu.

Procedure

- In the main menu, select **Diagnosis** > **Interrupts**.

In robot and submit programs, a maximum of 32 interrupts (64 from KSS 8.5 onwards) can be declared simultaneously and up to 16 interrupts (32 from KSS 8.5 onwards) can be active at the same time.

Further information about interrupt programming is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected.

Syntax

intinfo = `$SINT_LIST[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Index of the interrupt • **1 … 32 (64 from KSS 8.5 onwards)**
intinfo	Type: STRUC INT_INFO Structure with information about the interrupt

INT_INFO

```
STRUC INT_INFO INT INT_PRIO, INT_STATE, INT_TYPE,
PROG_LINE, CHAR PROG_NAME[32]
```

Element	Description
INT_PRIO	Type: INT Priority of the interrupt • **1, 2, 4 … 39** • **81 …128**
INT_STATE	Bit array for interrupt states • **Bit 0 = 1**: Interrupt is declared and activated. • **Bit 1 = 1**: Interrupt is activated and enabled. • **Bit 2 = 1**: Interrupt is globally declared.
INT_TYPE	Type: INT Type of interrupt • **0**: Standard interrupt • **1**: Interrupt due to an EMERGENCY STOP ($EMSTOP) • **2**: Interrupt due to activation of the Fast Measurement inputs ($MEAS_PULSE) • **3**: Interrupt due to an error stop ($STOPMESS) • **4**: Interrupt due to a trigger (subprogram call)
PROG_LINE	Type: INT Line number of the submit program in which the interrupt is declared
PROG_NAME[]	Type: CHAR[] Directory and name of the submit program in which the interrupt is declared

3.261 $SMARTPAD_SERIALNUMBER

Description

Indicates the serial number of the smartPAD that is connected to the robot controller.

Writability

The system variable is write-protected.

Syntax

"*serial*" = `$SMARTPAD_SERIALNUMBER[]`

Explanation of the syntax

Element	Description
serial	Type: CHAR Serial number of the smartPAD

3.262 $SOFTPLCBOOL

Description

Exchange of Boolean values between ProConOS and the robot controller

With the aid of function blocks of the Mulitprog library KrcExVarLib, individual or multiple values can be read from the array variable or written to the array variable.

Further information about the function blocks can be found in the **KUKA.PLC Multiprog** documentation.

Syntax

`$SOFTPLCBOOL[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 1024**
Value	Type: BOOL

3.263 $SOFTPLCINT

Description

Exchange of integer values between ProConOS and the robot controller

With the aid of function blocks of the Mulitprog library KrcExVarLib, individual or multiple values can be read from the array variable or written to the array variable.

Further information about the function blocks can be found in the **KUKA.PLC Multiprog** documentation.

Syntax

`$SOFTPLCINT[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 1024**
Value	Type: INT

3.264 $SOFTPLCREAL

Description

Exchange of real values between ProConOS and the robot controller

With the aid of function blocks of the Mulitprog library KrcExVarLib, individual or multiple values can be read from the array variable or written to the array variable.

Further information about the function blocks can be found in the **KUKA.PLC Multiprog** documentation.

Syntax

`$SOFTPLCREAL[`*Index*`]=`*Value*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 1024**
Value	Type: REAL

3.265 $SOFT_PLC_EVENT

Description

Event tasks of the Soft PLC (ProConOS)

The variable can be used to call the mapped ProConOS events 0 to 7. The event tasks are triggered by a positive edge of the variable.

Syntax

`$SOFT_PLC_EVENT[`*Index*`]=`*State*

Explanation of the syntax

Element	Description
Index	Type: INT Index of the variable • **1 … 8**: ProConOS events 0 … 7 (= KRC event bits 0 … 7)
State	Type: BOOL • TRUE: Event has been called. • FALSE: No event has been called.

3.266 $SPEED_LIMIT_TEACH_MODE

Description

Maximum TCP and flange velocity (HOV, T1)

This variable limits the TCP and flange velocity during jogging in T1 mode to the value set here.

Syntax

`$SPEED_LIMIT_TEACH_MODE=`*Velocity*

Explanation of the syntax

Element	Description
Velocity	Type: REAL; unit: m/s • **0.0 … 0.25** Default: **0.25**

3.267 $SPL_MIXEDBLENDING_OPT: Optimization of approximation (blending) between Cartesian and axis-specific spline motions

Description

Enables approximation (blending) between Cartesian and axis-specific spline motions to be optimized, e.g. approximation between SLIN and SPTP.

The improvements in behavior normally lead to a slightly different geometric path in the approximate positioning range.

Declaration

```
STEP_ENUM $SPL_MIXEDBLENDING_OPT = #NONE
```

Explanation

Element	Description
$SPL_MIXEDBLENDING_OPT	Data type: STEP_ENUM Mode for the optimization of approximation (blending) between Cartesian and axis-specific spline motions

Data type STEP_ENUM

```
ENUM STEP_ENUM NONE, STEP1, STEP2
```

Element	Description
#NONE	Default behavior In the case of system updates to systems in which the variable has not yet existed, the value is set to #NONE.
#STEP1	Improved symmetry with splines featuring mixed approximation
#STEP2	Improved symmetry with splines featuring mixed approximation. Leads to improved velocity profiles. #STEP2 is the default value for new installations.

3.268 $SPL_TECH

Description

Function parameters of the spline function generator in the advance run

The variable can be used to program up to 6 spline function generators. A spline function generator is active in the case of SPLINE, SLIN and SCIRC and approximation of these spline motions. Only the main run variables are evaluated.

The function parameters can be modified in the robot program relative to the advance run.

The validity of the variable is reset after planning of a spline motion. If $SPL_TECH is displayed using the variable display function, only the components of the next planned spline are visible.

Syntax

`$SPL_TECH[`*Index*`]=`*Parameters*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator: • **1 … 6**
Parameters	Type: Spl_Tech Definition of the function parameters and programming of the function evaluation

Spl_Tech

```
STRUC Spl_Tech BOOL hand_weaving, REAL
lim_full_hand_weaving, lim_no_hand_weaving, TECHMODE
mode, TECHCLASS class, SPL_TECHSYS ref_sys, SPL_FCTCTRL
fctctrl, SPL_TECHFCT fct, REAL fade_in, fade_out
```

Element	Description
hand_ weaving	Activation of wrist axis weaving. This can prevent the robot from vibrating during weaving. • TRUE: wrist axis weaving is activated. • FALSE: wrist axis weaving is deactivated. Default: FALSE
lim_full_ hand_ weaving	Limitation of the deviation from the programmed correction direction during wrist axis weaving. Only positive angles can be programmed. (unit: °)
lim_no_ hand_ weaving	• If the deviation is less than the angle `lim_full_hand_weaving`, only the wrist axis weaving correction is calculated. • If the deviation is greater than the angle `lim_full_hand_weaving`, but still less than `lim_no_hand_weaving`, both a wrist axis weaving correction and a standard correction are calculated. • If the deviation is greater than the angle `lim_no_hand_weaving`, only the standard correction is calculated.
mode	Technology mode: type of function evaluation • #OFF: no function evaluation • #SINGLE: the function is evaluated once. • #CYCLE: the function is evaluated cyclically.

Element	Description
class	Technology class: input variable for the spline function generator • #PATH: the input variable is the arc length of a spline motion $DISTANCE (unit: mm) • #SYSTIME: the input variable is the system time (unit: ms)
ref_sys	Structure for definition of the reference coordinate system for geometric weaving of the spline function generator `STRUC Spl_Techsys TECHSYS sys, TECHANGLE angles, TECHGEOREF georef` • Reference coordinate system selection (TECHSYS sys) – #WORLD – #BASE – #ROBROOT – #TCP – #TTS • Rotation of the reference coordinate system (TECHANGLE Angles; type: REAL; unit: °) – A, B, C: angles about which the Z, Y and X axes of the reference coordinate system are rotated • Reference coordinate system axis used for weaving (TECHGEOREF Georef) – #NONE: thermal weaving, not mechanical weaving, is carried out. This means, the weave pattern is not executed; instead, only the function value is written to the variable $TECHVAL. – #X, #Y, #Z: during weaving, the TCP is offset by the function value in the direction of the X, Y or Z axis of the reference coordinate system. – #A, #B, #C: these components are not permissible.
fctctrl	Control structure for the weave parameters of the spline function generator `STRUC Spl_Fctctrl BOOL adjust_wavelength, REAL scale_in, scale_out, offset_out` • BOOL adjust_wavelength: the wavelength `scale_in` can be adapted to an integer multiple of the remaining path for a single spline block (= TRUE) • REAL scale_in: wavelength of the weave pattern • REAL scale_out: amplitude of the weave pattern • REAL offset_out: position of the weave pattern, e.g.: – $SPL_TECH[1].FCTCTRL.OFFSET_OUT = 0.0: the zero point of the weave motion is on the spline path.

Element	Description
fct	Structure for defining the weave pattern for the spline function generator `STRUC Spl_TechFct SPL_TECHFCT_MODE mode, TECHFCT Polynomial, SPL_FCTDEF fct_def, REAL blend_in, blend_out, phase_shift` • ENUM for specification of the weave pattern type (SPL_TECHFCT_MODE mode) – #FCT_POLYNOMIAL: polynomial with control points – #FCT_SIN: asymmetrical sine (both half-waves can have different amplitudes and wavelengths) • Structure for defining the weave parameters for the weave pattern type #FCT_POLYNOMIAL (TECHFCT Polynomial) (>>> *"TechFct" Page 174*) • Structure for defining the weave parameters for the weave pattern type #FCT_SIN (SPL_FCTDEF fct_def) (>>> *"Spl_FctDef" Page 175*) • REAL blend_in: if the weave parameters are modified, it specifies the fraction of the wavelength at which weaving starts to deviate from the original pattern • REAL blend_out: If the weave parameters are modified, it specifies the fraction of the wavelength at which a transition to the new pattern is started • REAL phase_shift: phase offset for execution of the weave pattern, e.g.: – $SPL_TECH[1].FCT.PHASE_SHIFT = 0.0: weaving commences at the start of the spline path. **Note**: The definition range and range of values of the function `fct` are defined as follows: • Definition range: **0 … 1** • Range of values: **-1 … +1**
fade_in fade_out	Smoothing length for start of weaving and end of weaving • **0.0 … 1.0** For a defined interval after the start and before the end, the specified weave pattern is multiplied by an S-shaped function with values rising from 0 to 1. The length of these intervals is defined using `fade_in` and `fade_out`. The longer the interval, the smoother the motion.

TechFct

```
STRUC TechFct INT order, cpnum, TECHCPS cps1, cps2,
cps3, cps4, cps5, SPL_TECH_BOUND bound_cond
```

Element	Description
order	Degree of interpolation during spline evaluation • **1**: Weave pattern is defined by a polygon (sufficient in the case of thermal weaving alone) • **3**: Weave pattern is defined by a cubic spline (required in the case of mechanical weaving) **Note**: If a polygon is sufficiently smooth for mechanical weaving due to utilization of the maximum number of control points, degree of interpolation 1 is also allowed).
cpnum	Total number of valid control points with reference to the following 4 control point structures • **2 … 40** **Note**: No gaps are allowed between the valid control points.
cps1	List with control points 1 … 10 (type: REAL) • **X1, Y1, … X10, Y10**
cps2	List with control points 11 … 20 (type: REAL) • **X1, Y1, … X10, Y10**
cps3	List with control points 21 … 30 (type: REAL) • **X1, Y1, … X10, Y10**
cps4	List with control points 31 … 40 (type: REAL) • **X1, Y1, … X10, Y10**
cps5	This component cannot be written, as the number of control points is limited to 40.
bound_cond	Boundary conditions for a cubic spline (only relevant for degree of interpolation 3) • #CYCLIC: Cyclical boundary conditions (required in the case of mechanical weaving) • #NATURAL: Natural boundary conditions (sufficient in the case of thermal weaving alone)

Spl_FctDef

```
STRUC Spl_FctDef REAL amplitude1, amplitude2, wave-
length_ratio
```

Element	Description
amplitude1	Amplitude of the 1st half-wave of the asymmetrical sine (unit: dependent on the input size)
amplitude2	Amplitude of the 2nd half-wave of the asymmetrical sine (unit: dependent on the input size)
wave-length_ratio	Ratio of the wavelengths of the 2nd half-wave to the wavelength of the 1st half-wave

3.269 $SPL_TECH_C

Description

Function parameters of the spline function generator in the main run

The variable can be used to program up to 6 spline function generators. A spline function generator is active in the case of SPLINE, SLIN and SCIRC and approximation of these spline motions. Only the main run variables are evaluated.

The function parameters can be modified relative to the main run by means of triggers, interrupts and the variable display function. The modifications are retained after a block change if these parameters have not been reprogrammed in the advance run.

The variable contains the currently used data of a spline function generator. This has the following effect:

- If the variable is modified, e.g. by a trigger, the change cannot be read immediately, but only after it has been accepted by the function generator.
- In the case of a multidimensional function generator, this refers to the shared data of the function generator acting as the master. (>>> *3.270 "$SPL_TECH_LINK" Page 181*)

Syntax

`$SPL_TECH_C[`*Index*`]=`*Parameters*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator: • **1 … 6**
Parameters	Type: Spl_Tech Definition of the function parameters and programming of the function evaluation

Spl_Tech

```
STRUC Spl_Tech BOOL hand_weaving, REAL
lim_full_hand_weaving, lim_no_hand_weaving, TECHMODE
mode, TECHCLASS class, SPL_TECHSYS ref_sys, SPL_FCTCTRL
fctctrl, SPL_TECHFCT fct, REAL fade_in, fade_out
```

Element	Description
hand_ weaving	Activation of wrist axis weaving. This can prevent the robot from vibrating during weaving. • TRUE: wrist axis weaving is activated. • FALSE: wrist axis weaving is deactivated. Default: FALSE
lim_full_ hand_ weaving	Limitation of the deviation from the programmed correction direction during wrist axis weaving. Only positive angles can be programmed. (unit: °) • If the deviation is less than the angle `lim_full_hand_weaving`, only the wrist axis weaving correction is calculated. • If the deviation is greater than the angle `lim_full_hand_weaving`, but still less than `lim_no_hand_weaving`, both a wrist axis weaving correction and a standard correction are calculated. • If the deviation is greater than the angle `lim_no_hand_weaving`, only the standard correction is calculated.
lim_no_ hand_ weaving	
mode	Technology mode: type of function evaluation • #OFF: no function evaluation • #SINGLE: the function is evaluated once. • #CYCLE: the function is evaluated cyclically.
class	Technology class: input variable for the spline function generator • #PATH: the input variable is the arc length of a spline motion $DISTANCE (unit: mm) • #SYSTIME: the input variable is the system time (unit: ms)

Element	Description
ref_sys	Structure for definition of the reference coordinate system for geometric weaving of the spline function generator `STRUC Spl_Techsys TECHSYS sys, TECHANGLE angles, TECHGEOREF georef` • Reference coordinate system selection (TECHSYS sys) – #WORLD – #BASE – #ROBROOT – #TCP – #TTS • Rotation of the reference coordinate system (TECHANGLE Angles; type: REAL; unit: °) – A, B, C: angles about which the Z, Y and X axes of the reference coordinate system are rotated • Reference coordinate system axis used for weaving (TECHGEOREF Georef) – #NONE: thermal weaving, not mechanical weaving, is carried out. This means, the weave pattern is not executed; instead, only the function value is written to the variable $TECHVAL. – #X, #Y, #Z: during weaving, the TCP is offset by the function value in the direction of the X, Y or Z axis of the reference coordinate system. – #A, #B, #C: these components are not permissible.
fctctrl	Control structure for the weave parameters of the spline function generator `STRUC Spl_Fctctrl BOOL adjust_wavelength, REAL scale_in, scale_out, offset_out` • BOOL adjust_wavelength: the wavelength `scale_in` can be adapted to an integer multiple of the remaining path for a single spline block (= TRUE) • REAL scale_in: wavelength of the weave pattern • REAL scale_out: amplitude of the weave pattern • REAL offset_out: position of the weave pattern, e.g.: – $SPL_TECH[1].FCTCTRL.OFFSET_OUT = 0.0: the zero point of the weave motion is on the spline path.

Element	Description
fct	Structure for defining the weave pattern for the spline function generator `STRUC Spl_TechFct SPL_TECHFCT_MODE mode, TECHFCT Polynomial, SPL_FCTDEF fct_def, REAL blend_in, blend_out, phase_shift` • ENUM for specification of the weave pattern type (SPL_TECHFCT_MODE mode) – #FCT_POLYNOMIAL: polynomial with control points – #FCT_SIN: asymmetrical sine (both half-waves can have different amplitudes and wavelengths) • Structure for defining the weave parameters for the weave pattern type #FCT_POLYNOMIAL (TECHFCT Polynomial) (>>> *"TechFct" Page 179*) • Structure for defining the weave parameters for the weave pattern type #FCT_SIN (SPL_FCTDEF fct_def) (>>> *"Spl_FctDef" Page 180*) • REAL blend_in: if the weave parameters are modified, it specifies the fraction of the wavelength at which weaving starts to deviate from the original pattern • REAL blend_out: If the weave parameters are modified, it specifies the fraction of the wavelength at which a transition to the new pattern is started • REAL phase_shift: phase offset for execution of the weave pattern, e.g.: – $SPL_TECH[1].FCT.PHASE_SHIFT = 0.0: weaving commences at the start of the spline path. **Note**: The definition range and range of values of the function `fct` are defined as follows: • Definition range: **0 … 1** • Range of values: **-1 … +1**
fade_in fade_out	Smoothing length for start of weaving and end of weaving • **0.0 … 1.0** For a defined interval after the start and before the end, the specified weave pattern is multiplied by an S-shaped function with values rising from 0 to 1. The length of these intervals is defined using `fade_in` and `fade_out`. The longer the interval, the smoother the motion.

TechFct

```
STRUC TechFct INT order, cpnum, TECHCPS cps1, cps2,
cps3, cps4, cps5, SPL_TECH_BOUND bound_cond
```

Element	Description
order	Degree of interpolation during spline evaluation • **1**: Weave pattern is defined by a polygon (sufficient in the case of thermal weaving alone) • **3**: Weave pattern is defined by a cubic spline (required in the case of mechanical weaving) **Note**: If a polygon is sufficiently smooth for mechanical weaving due to utilization of the maximum number of control points, degree of interpolation 1 is also allowed).
cpnum	Total number of valid control points with reference to the following 4 control point structures • **2 … 40** **Note**: No gaps are allowed between the valid control points.
cps1	List with control points 1 … 10 (type: REAL) • **X1, Y1, … X10, Y10**
cps2	List with control points 11 … 20 (type: REAL) • **X1, Y1, … X10, Y10**
cps3	List with control points 21 … 30 (type: REAL) • **X1, Y1, … X10, Y10**
cps4	List with control points 31 … 40 (type: REAL) • **X1, Y1, … X10, Y10**
cps5	This component cannot be written, as the number of control points is limited to 40.
bound_cond	Boundary conditions for a cubic spline (only relevant for degree of interpolation 3) • #CYCLIC: Cyclical boundary conditions (required in the case of mechanical weaving) • #NATURAL: Natural boundary conditions (sufficient in the case of thermal weaving alone)

Spl_FctDef

```
STRUC Spl_FctDef REAL amplitude1, amplitude2, wave-
length_ratio
```

Element	Description
amplitude1	Amplitude of the 1st half-wave of the asymmetrical sine (unit: dependent on the input size)
amplitude2	Amplitude of the 2nd half-wave of the asymmetrical sine (unit: dependent on the input size)
wave-length_ratio	Ratio of the wavelengths of the 2nd half-wave to the wavelength of the 1st half-wave

Example

```
$SPL_TECH[1].FCTCTRL.SCALE_OUT=10.0
TRIGGER WHEN PATH=-100 DELAY=0 DO
$SPL_TECH_C[1].FCTCTRL.SCALE_OUT=20.0
```

```
SLIN XP1
SLIN XP2
```

The weave amplitude SCALE_OUT changes with SLIN XP1 from 10 to 20. For SLIN XP2, also, the amplitude of 20 remains valid.

If, for SLIN XP2, the original weave amplitude of 10 is to apply again, this must be explicitly programmed:

```
$SPL_TECH[1].FCTCTRL.SCALE_OUT=10.0
TRIGGER WHEN PATH=-100 DELAY=0 DO
$SPL_TECH_C[1].FCTCTRL.SCALE_OUT=20.0
SLIN XP1
$SPL_TECH[1].FCTCTRL.SCALE_OUT=10.0
SLIN XP2
```

3.270 $SPL_TECH_LINK

Description

Shared function parameters of the spline function generators in the advance run

The variable of type Spl_Tech_Map can be used to link and unlink spline function generators in the robot program, relative to the advance run.

Multidimensional function generators share the following function parameters:

Linked function generators ...	Parameter
are activated together and deactivated together.	MODE
have a shared wavelength.	FCTCTRL.SCALE_IN
	FCTCTRL.ADJUST_WAVELENGTH
require the same length of time to reach their full amplitude and to come back down from it.	FADE_IN
	FADE_OUT
use the same input parameters, arc length or time.	CLASS
use the same reference coordinate system for the geometric correction.	REF_SYS.SYS
use wrist axis weaving together.	HAND_WEAVING
	HAND_WEAVING_MAX_ADJUST

Wrist axis weaving is only possible for 2-dimensional function generators.

Spl_Tech_Map

```
STRUC Spl_Tech_Map INT FG1, FG2, FG3, FG4, FG5, FG6
```

In order to link function generator *x* with function generator *y*, component FG*x* is assigned the integer value *y*. This means that function generator *x* uses the data of function generator *y*. Function generator *y* acts as the master. If *x*=*y*, function generator *x* is not linked.

Example

```
$SPL_TECH_LINK={FG1 1, FG2 1, FG3 1, FG4 4, FG5 5, FG6 6}
```

Function generators 1 to 3 are linked. Shared data are taken from function generator 1. Function generators 4 to 6 work without coupling.

Links across multiple stations, such as in the following example, are not permissible:

```
$SPL_TECH_LINK={FG1 2, FG2 3, FG3 3, FG4 4, FG5 5, FG6 6}
```

If function generator 1 refers to function generator 2, function generator 2 cannot simultaneously refer to function generator 3.

3.271 $SPL_TECH_LINK_C

Description

Shared function parameters of the spline function generators in the main run

The variable of type Spl_Tech_Map can be used to link and unlink spline function generators by means of triggers or interrupts, relative to the main run.

A new component can only be added to a multidimensional function generator relative to the main run, or be removed again, if the multidimensional function generator is not currently active.

Spl_Tech_Map

```
STRUC Spl_Tech_Map INT FG1, FG2, FG3, FG4, FG5, FG6
```

In order to link function generator *x* with function generator *y*, component FG*x* is assigned the integer value *y*. This means that function generator *x* uses the data of function generator *y*. Function generator *y* acts as the master. If *x*=*y*, function generator *x* is not linked.

3.272 $SPL_TSYS

Description

Position of the reference coordinate system of a spline function generator relative to BASE

The variable contains the current position of the reference coordinate system of a function generator relative to the BASE coordinate system. The position is calculated for both active and inactive function generators in the spline interpolator.

In the case of a PTP, LIN or CIRC motion, the variable loses its validity. This also applies if a program is reset or deselected.

Writability

The system variable is write-protected.

Syntax

position = $SPL_TSYS[*index*]

Explanation of the syntax

Element	Description
index	Type: INT Number of the function generator: • **1 … 6**
position	Type: FRAME • **X**, **Y**, **Z**: offset of the origin along the axes in [mm] • **A**, **B**, **C**: rotational offset of the axis angles in [°]

3.273 $SPL_VEL_MODE

Description

Motion profile for spline motions

In the robot program, the variable triggers an advance run stop.

Syntax

`$SPL_VEL_MODE =` *mode*

Explanation of the syntax

Element	Description
mode	Type: ENUM SPL_VEL_MODE • #OPT: higher motion profile • #CART: conventional motion profile Default: #OPT

3.274 $SPL_VEL_RESTR

Description

Activation of Cartesian limits for spline motions with higher motion profile

The higher motion profile enables the robot controller already to take axis-specific limits into consideration during path planning, and not wait until they are detected by monitoring functions during execution. All applications can generally be executed faster with the higher motion profile.

The variable can be used to activate further Cartesian limits. In the robot program, the variable triggers an advance run stop.

Precondition

- The higher motion profile is active: $SPL_VEL_MODE=#OPT

Syntax

`$SPL_VEL_RESTR=`*Limits*

Explanation of the syntax

Element	Description
Limits	Type: Spl_Vel_Restr_Struc The restrictions are deactivated by default. #ON activates them; #OFF deactivates them.

Spl_Vel_Restr _Struc

```
STRUC Spl_Vel_Restr_Struc SW_ONOFF ori_vel, cart_acc,
ori_acc, cart_jerk, ori_jerk, rob_acc, rob_jerk
```

Element	Description
ori_vel	Maximum orientation velocity
cart_acc	Maximum Cartesian acceleration
ori_acc	Maximum orientation acceleration
cart_jerk	Maximum Cartesian jerk
ori_jerk	Maximum orientation jerk
rob_acc	Maximum acceleration of the robot axes
rob_jerk	Maximum jerk of the robot axes

Example

```
$SPL_VEL_RESTR={ORI_VEL=#OFF, CART_ACC=#ON, …, ROB_JERK=#OFF}
```

3.275 $SPO_GEARTORQ[]: Axis-specific maximum values for the gear torque for Safe Power Off

Description

Enables the definition of axis-specific maximum values for the gear torque if, during a Safe Power Off, a stop is carried out with an EMERGENCY STOP with applied brakes (NGB) or model-based maximum braking with applied brakes (MDGB).

Further information:

- Definition of the stop type with which the controller reacts to a Safe Power Off with $SPO_REACTION (>>> *3.276 "$SPO_REACTION: Stop type for Safe Power Off" Page 185*)

Declaration

```
REAL $SPO_GEARTORQ[6]
```

Explanation

Element	Description
$SPO_GEARTORQ[i]	Data type: REAL Axis-specific maximum values for the gear torque for Safe Power Off [Nm] • **i**: Robot axis (1-6)

3.276 $SPO_REACTION: Stop type for Safe Power Off

Description

Defines the stop type with which the controller reacts to a Safe Power Off.

Further information:
- Definition of axis-specific maximum values with $SPO_GEARTORQ[] (>>> *3.275 "$SPO_GEARTORQ[]: Axis-specific maximum values for the gear torque for Safe Power Off" Page 184*)

Declaration

```
SPO_REACTION $SPO_REACTION = #NEAR_PATH_STOP
```

Explanation

Element	Description
$SPO_REACTION	Data type: SPO_REACTION Defines the stop type with which the controller reacts to a Safe Power Off. Default: NEAR_PATH_STOP

Data type SPO_REACTION

```
ENUM SPO_REACTION OFF, NEAR_PATH_STOP, MODEL_RAMP_STOP
```

Element	Description
#OFF	Hardware does not support Safe Power Off
#NEAR_PATH_STOP	EMERGENCY STOP with applied brakes (NGB)
#MODEL_RAMP_STOP	Model-based maximum braking with applied brakes (MDGB)

3.277 $SS_MODE

Description

Signal declaration for Single Step mode

This output is set when operating mode T1 or T2 is selected.

Syntax

```
SIGNAL $SS_MODE $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **139**

3.278 $STOPMB_ID

Description

Identification of the mailbox for stop messages

The variable is write-protected and required for reading the mailbox contents with the MBX_REC function.

Syntax

```
$STOPMB_ID=Identifier
```

Explanation of the syntax

Element	Description
Identifier	Type: INT

3.279 $STOPMESS

Description

Signal declaration for stop messages

This output is set in order to communicate to the higher-level controller the occurrence of any message that required the robot to be stopped. For example, after an EMERGENCY STOP or operator safety violation.

Syntax

```
SIGNAL $STOPMESS $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1010**

3.280 $STOPNOAPROX

Description

Message type in the case of “approximation not possible”

In modes T1 and T2, $STOPNOAPROX determines the message type for messages 1123, 1442 and 2920:

- Either notification message that does not trigger a stop
- Or acknowledgement message that triggers a stop

These messages are always notification messages in operating modes AUT (KR C), AUT EXT (KR C) and EXT (VKR C). In Automatic operating modes, these notification messages are filtered out by the smartHMI by default and not displayed (this can be configured in C:\KRC\USER \SmartHMI.User.config).

Syntax

```
$STOPNOAPROX = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE – T1 and T2 mode: acknowledgement message – Operating mode AUT (KR C), AUT EXT (KR C) and EXT (VKR C): notification message • FALSE: notification message (in all operating modes) Default: FALSE

3.281 $SUPPLY_VOLTAGE: Labeling of machine data with information about voltage

Description

Enables the machine data to be labeled with the voltage for which they were created.

Declaration

```
SUPPLY_VOLTAGE $SUPPLY_VOLTAGE = #SV400
```

Explanation

Element	Description
$SUPPLY_VOLTAGE	Data type: SUPPLY_VOLTAGE Labeling of the machine data with the voltage for which they were created Default: 400 V

Data type SUPPLY_VOLTAGE

```
ENUM SUPPLY_VOLTAGE SV120, SV200, SV230, SV380, SV400,
SV440, SV480
```

Element	Description
SV120	Power supply system with 120 V
SV200	Power supply system with 200 V
SV230	Power supply system with 230 V
SV380	Power supply system with 380 V
SV400	Power supply system with 400 V
SV440	Power supply system with 440 V
SV480	Power supply system with 480 V

3.282 $TARGET_STATUS

Description

Status for the motion to the target point

The variable is used by the KRL INVERSE() function. The Status defined here is adopted if the Status value transferred for the target point is invalid.

Syntax

```
$TARGET_STATUS = state
```

Explanation of the syntax

Element	Description
state	Type: ENUM TARGET_STATUS • #SOURCE: the Status of the start point is adopted. • #BEST: the 8 possible Status combinations are calculated. The combination resulting in the shortest motion from the start to the target point is used. Default: #SOURCE

3.283 $TCP_IPO

Description

Activation/deactivation of gripper-related interpolation

The interpolation mode $IPO_MODE can only be modified if gripper-related interpolation has been activated.

Writability

The variable can be modified in $option.dat in KRC:\STEU\Mada.

It cannot be modified via the variable display function or program.

Syntax

```
$TCP_IPO = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: gripper-related interpolation is switched on. • FALSE: gripper-related interpolation is switched off. Default: TRUE

3.284 $TECH

Description

Function parameters of the function generator in the advance run

The variable can be used to program up to 6 function generators . The function generator is only active for CP motions; only the main run variables are evaluated.

The function parameters can be modified in the robot program relative to the advance run.

Syntax

```
$TECH[Index]=Parameter
```

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator • **1 … 6**
Parameter	Type: Tech Definition of the function parameters and programming of the function evaluation

Tech

```
STRUC Tech TECHMODE mode, TECHCLASS class, TECHFCTCTRL
fctctrl, TECHFCT fct
```

Element	Description
mode	Technology mode: type of function evaluation • #OFF: no function evaluation • #SINGLE: the function is evaluated once. • #CYCLE: the function is evaluated cyclically.
class	Technology class: input variable for the function generator • #PATH: the input variable is the arc length of a CP motion $DISTANCE (unit: mm) • #SYSTIME: the input variable is the system time (unit: ms) • #VEL: the input variable is the current path velocity $VEL_ACT (unit: m/s) • #SENSOR: input size is the variable $TECHIN. Depending on the input values, the robot performs a position correction. • #DATALINK: the input variable is a correction frame that is written by the sensor task. Depending on the input values, the robot performs a correction.
fctctrl	Control structure for the parameters of the function generator `STRUC Fctctrl REAL scale_in, scale_out, offset_in, offset_out, TECHGEOREF georef` • REAL scale_in: scales the definition range of the function • REAL scale_out: scales the range of values of the function • REAL offset_in: offsets the zero point of the definition range of the function • REAL offset_out: offsets the zero point of the range of values of the function • TECHGEOREF georef: ENUM for the geometric reference of the technology function (>>> *"GeoRef" Page 190*) **Note**: The offsets and scaling refer to the technology class. The technology class is the input variable of the function generator.

Element	Description
fct	Structure for defining the function parameters of the function generator (>>> *"TechFct" Page 190*) **Note**: The definition range and range of values of the function `fct` are defined as follows: • Definition range: **0 … 1** • Range of values: **-1 … +1**

GeoRef

Parameter	Description
#NONE	The programmed function is evaluated, but not carried out. The function value is written to the variable $TECHVAL. **Exception**: if the technology class #SENSOR is used, the parameter has the effect that no function evaluation is carried out.
#X, #Y, #Z	Axis of the reference coordinate system (programmed by means of $TECHSYS and $TECHANGLE) used for weaving or sensor correction • During weaving or sensor correction, the TCP is offset by the function value in the direction of the X, Y or Z axis of the reference coordinate system.
#A, #B, #C	Only relevant if the technology class #SENSOR is used Axis angle of the reference coordinate system (programmed by means of $TECHSYS and $TECHANGLE) used for sensor correction • The orientation of the TCP changes: rotation by the function value about the Z, Y or X axis of the reference coordinate system (always in the mathematically positive direction)

TechFct

```
STRUC TechFct INT order, cpnum, TECHCPS cps1, cps2,
cps3, cps4, cps5
```

Element	Description
order	Degree of interpolation during function evaluation • **1**: Function is defined by a polygon
cpnum	Total number of valid control points with reference to the following 5 control point structures • **2 … 50** **Note**: No gaps are allowed between the valid control points.
cps1	List with control points 1 … 10 (type: REAL) • **X1, Y1, … X10, Y10**

Element	Description
cps2	List with control points 11 … 20 (type: REAL) • **X1, Y1, … X10, Y10**
cps3	List with control points 21 … 30 (type: REAL) • **X1, Y1, … X10, Y10**
cps4	List with control points 31 … 40 (type: REAL) • **X1, Y1, … X10, Y10**
cps5	List with control points 41 … 50 (type: REAL) • **X1, Y1, … X10, Y10**

Example

Using a distance sensor, a correction of max. ±20 mm is to be made in the Z direction of the TTS. The analog sensor input delivers a voltage between -10 V and +10 V; this should be adjusted to 0 V (factor = 0.1; offset = 1.0).

```
 1   SIGNAL Correction $ANIN[2]
 2
 3   INTERRUPT DECL 1 WHEN $TECHVAL[1] > 20.0 DO
Upper_Limit()
 4   INTERRUPT DECL 2 WHEN $TECHVAL[1] < -20.0 DO
Lower_Limit()
 5
 6   ANIN ON $TECHIN[1] = Factor * Correction + Offset
 7
 8   $TECHSYS = #TTS
 9   $TECH[1].FTCCTRL.GEOREF = #Z
 10
 11 $TECH.CLASS = #SENSOR
 12
 13 $TECH[1].FCTCTRL.SCALE_IN = 2.0
 14 $TECH[1].FCTCTRL.OFFSET_IN = 0.0
 15 $TECH[1].FCTCTRL.SCALE_OUT = 2.0
 16 $TECH[1].FCTCTRL.OFFSET_OUT = 0.0
 17 $TECH[1].FCT.ORDER = 1
 18 $TECH[1].FCT.CPNUM = 3
 19 $TECH[1].FCT.CPS1.X1 = 0.0
 20 $TECH[1].FCT.CPS1.Y1 = -1.0
 21 $TECH[1].FCT.CPS1.X2 = 0.5
 22 $TECH[1].FCT.CPS1.Y2 = 0.0
 23 $TECH[1].FCT.CPS1.Y3 = 3.0
 24 $TECHPAR[1,1] = 0.056
 25
 26 PTP Go_to_Workpiece
 27 INTERRUPT ON 1
 28 INTERRUPT ON 2
 29
 30 TECH[1].MODE = #CYCLE
 31 LIN P1 C_DIS
 32 LIN P2 C_DIS
 33 LIN P3
 34
 35 TECH[1].MODE = #OFF
 36 LIN_REL {X 0.0}
 37
```

```
38 ANIN OFF Correction
```

Line	Description
1	Sensor at analog input 2
3, 4	Interrupts for monitoring the sensor correction
6	Cyclical reading of the analog input is started and the input value $TECHIN[1] is standardized to between 0.0 and 2.0.
8, 9	Correction in the Z direction of the TTS
13	Function generator with sensor correction functionality
14 … 23	Control parameters of the function generator
24	Smoothing constant (unit: s)
26 … 28	A motion is executed to a defined point in front of the workpiece and the interrupts for monitoring the sensor correction are activated.
30	Sensor correction is activated.
35	Sensor correction is deactivated.
36	The zero block is used to apply the advance run data to the main run data. This deactivates the function generator.
38	Cyclical reading of the analog input is started.

3.285 $TECH_ANA_FLT_OFF

Description

Analog value filter for $TECHVAL[.]

The variable $ANA_DEL_FILT can be used to set the robot controller response to use of an analog output (ANOUT) with a negative DELAY. An analog output with a negative DELAY is proportional either to the current velocity $VEL_ACT or to the current technology parameter $TECHVAL[.].

In more recent software releases you can configure whether the current values of $VEL_ACT or $TECHVAL[.] are calculated before or after the filter. The default settings $VEL_FLT_OFF=TRUE and $TECH_ANA_FLT_OFF[*x*]=TRUE ensure high-accuracy calculation of these values in all situations. Therefore these settings are strongly recommended.

The settings $VEL_FLT_OFF=FALSE or $TECH_ANA_FLT_OFF[*x*]=FALSE can be used to achieve behavior that is compatible with old software releases. Only in this way is it possible to generate the negative DELAY – at least in part – by reducing the analog value filter. This behavior is configured by setting $ANA_DEL_FLT=#OFF. In this way a small amount of cycle time is saved for each single block with exact positioning. This time saving however entails a reduction in the precision of the analog signal.

On the other hand, the default settings $VEL_FLT_OFF=TRUE or $TECH_ANA_FLT_OFF[*x*]=TRUE mean it is not possible to achieve a negative DELAY by reducing the analog value filter. The switch $ANA_DEL_FLT has no effect in this case.

Syntax

`$TECH_ANA_FLT_OFF[`*Axis number*`]=`*State*

Explanation of the syntax

Element	Description
Axis number	Type: INT • **1 … 6**: robot axis A1 ... A6
Status	Type: BOOL • TRUE: analog value filter is deactivated. • FALSE: analog value filter is activated. Default: TRUE

3.286 $TECH_C

Description

Function parameters of the function generator in the main run

The variable can be used to program up to 6 function generators. The function generator is only active for CP motions; only the main run variables are evaluated.

The function parameters can be modified relative to the main run by means of triggers, interrupts and the variable display function. The modifications are retained after a block change if these parameters have not been reprogrammed in the advance run.

The variable contains the currently used data of a function generator. This has the following effect:

- If the variable is modified, e.g. by a trigger, the change cannot be read immediately, but only after it has been accepted by the function generator.

Syntax

`$TECH_C[`*Index*`]=`*Parameter*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator • **1 … 6**
Parameter	Type: Tech Definition of the function parameters and programming of the function evaluation

Tech

```
STRUC Tech TECHMODE mode, TECHCLASS class, TECHFCTCTRL
fctctrl, TECHFCT fct
```

Element	Description
mode	Technology mode: type of function evaluation • #OFF: no function evaluation • #SINGLE: the function is evaluated once. • #CYCLE: the function is evaluated cyclically.
class	Technology class: input variable for the function generator • #PATH: the input variable is the arc length of a CP motion $DISTANCE (unit: mm) • #SYSTIME: the input variable is the system time (unit: ms) • #VEL: the input variable is the current path velocity $VEL_ACT (unit: m/s) • #SENSOR: input size is the variable $TECHIN. Depending on the input values, the robot performs a position correction. • #DATALINK: the input variable is a correction frame that is written by the sensor task. Depending on the input values, the robot performs a correction.
fctctrl	Control structure for the parameters of the function generator `STRUC Fctctrl REAL scale_in, scale_out, offset_in, offset_out, TECHGEOREF georef` • REAL scale_in: scales the definition range of the function • REAL scale_out: scales the range of values of the function • REAL offset_in: offsets the zero point of the definition range of the function • REAL offset_out: offsets the zero point of the range of values of the function • TECHGEOREF georef: ENUM for the geometric reference of the technology function (>>> *"GeoRef" Page 190*) **Note**: The offsets and scaling refer to the technology class. The technology class is the input variable of the function generator.
fct	Structure for defining the function parameters of the function generator (>>> *"TechFct" Page 190*) **Note**: The definition range and range of values of the function `fct` are defined as follows: • Definition range: **0 … 1** • Range of values: **-1 … +1**

3.287 $TECH_FUNC

Description

Active functions of the function generator

Syntax

```
$TECH_FUNC=Bit array
```

Explanation of the syntax

Element	Description
Bit array	Bit array with which individual functions can be activated. • **Bit n = 0**: Function is not active. • **Bit n = 1**: Function is active.

Bit n	5	4	3	2	1	0
Function	6	5	4	3	2	1

3.288 $TECH_OPT

Description

Operating state of the function generator

Writability

The variable can be modified in $option.dat in KRC:\STEU\Mada.
It cannot be modified via the variable display function or program.

Syntax

```
$TECH_OPT = state
```

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: function generator is active. • FALSE: function generator is not active. Default: TRUE

3.289 $TECHANGLE

Description

Rotational offset of the reference coordinate system of a function generator in the advance run

The system variable can be used to define the orientation of the reference coordinate system defined by $TECHSYS and to modify it, relative to the advance run, in the robot program.

This variable is not relevant for spline function generators.

Syntax

```
$TECHANGLE = {A z, B y, C x}
```

Explanation of the syntax

Element	Description
z	Type: REAL; unit: °
y	Angle by which the Z, Y or X axis of the reference coordinate system is rotationally offset (only positive direction permissible)
x	

3.290 $TECHANGLE_C

Description

Rotation of the reference coordinate system of a function generator in the main run

The system variable can be used to specify the orientation of the reference coordinate system defined by $TECHSYS and to modify it, relative to the main run, by means of triggers, interrupts and the variable display function. The modifications are retained after a block change if the orientation has not been reprogrammed in the advance run.

This variable is not relevant for spline function generators.

Syntax

```
$TECHANGLE_C = {A z, B y, C x}
```

Explanation of the syntax

Element	Description
z	Type: REAL; unit: °
y	Angle by which the Z, Y or X axis of the reference coordinate system is rotationally offset (only positive direction permissible)
x	

3.291 $TECHIN

Description

Input value for the function generator

The variable forms the interface between the analog sensor inputs of the robot controller and the function generator.

Information about function parameters in the main run of the function generator are contained in the variable **$TECH** (>>> *3.284 "$TECH" Page 188*).

This variable is not relevant for spline function generators.

Writability

The system variable is write-protected.

Precondition

- Technology class #SENSOR (>>> *"Tech" Page 189*)

Syntax

Input value = $TECHIN[*index*]

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator • **1 … 6**
Input value	Type: REAL Value loaded via the sensor input.

Example

Data are written to this variable cyclically using the following statement:

- Example of a sensor at analog input 2:

```
SIGNAL Korrektur $ANIN[2]
     ANIN ON $TECHIN[1] = Faktor * Korrektur + Offset
```

It is not possible to write data directly to the variable from the robot program.

3.292 $TECHPAR

Description

Parameters of the function generator in the advance run

The variable can be used to define up to 10 input or output parameters of a function generator. If a parameter is used to output function generator states, the current parameter value can be found in the main run variable.

The variable can be modified in the robot program relative to the advance run.

This variable is not relevant for spline function generators.

Syntax

$TECHPAR[*Index 1*,*Index 2*]=*Parameter value*

Explanation of the syntax

Element	Description
Index 1	Type: INT Number of the function generator • **1 … 6**
Index 2	Type: INT Number of the parameter • **1 … 10**
Parameter value	Type: REAL

3.293 $TECHPAR_C

Description

Parameters of the function generator in the main run

The variable can be used to define up to 10 input or output parameters of a function generator. If a parameter is used to output function generator states, the current parameter value can be found in the main run variable.

The function parameters can be modified relative to the main run by means of triggers, interrupts and the variable display function.

	This variable is not relevant for spline function generators.

Writability

The system variable is write-protected.

Syntax

`Parameter value = $TECHPAR_C[`*Index 1*`,`*Index 2*`]`

Explanation of the syntax

Element	Description
Index 1	Type: INT Number of the function generator • **1 … 6**
Index 2	Type: INT Number of the parameter • **1 … 10**
Parameter value	Type: REAL

3.294 $TECHSYS

Description

Reference coordinate system of a function generator in the advance run

The variable is used to define the reference coordinate system to which the function values calculated by the function generator refer.

The variable can be modified in the robot program relative to the advance run.

	This variable is not relevant for spline function generators.

Syntax

`$TECHSYS =` *coordinate system*

Precondition

- GEOREF<>#NONE
  (>>> *"GeoRef" Page 190*)

Explanation of the syntax

Element	Description
coordinate system	Type: ENUM TECHSYS • #BASE • #ROBROOT • #TCP • #TTS • #WORLD

3.295 $TECHSYS_C

Description

Reference coordinate system of a function generator in the main run

The variable is used to define the reference coordinate system to which the function values calculated by the function generator refer.

The variable can be modified relative to the main run by means of triggers, interrupts and the variable display function. The modification is retained after a block change if the reference coordinate system has not been reprogrammed in the advance run.

This variable is not relevant for spline function generators.

Precondition

- GEOREF<>#NONE
  (>>> *"GeoRef" Page 190*)

Syntax

`$TECHSYS_C =` *coordinate system*

Explanation of the syntax

Element	Description
coordinate system	Type: ENUM TECHSYS • #BASE • #ROBROOT • #TCP • #TTS • #WORLD

3.296 $TECHVAL

Description

Function value of a function generator .

The variable contains the result of the programmed function of a function generator. This can be a conventionally programmed function generator or a spline function generator.

Information about function parameters in the main run of the function generator are contained in the variable **$TECH** (>>> *3.284 "$TECH" Page 188*).

If the function value $TECHVAL of a spline function generator is written to an analog output ANOUT, no negative DELAY is possible.

Writability

The system variable is write-protected.

Syntax

Result=$TECHVAL[*Index*]

Explanation of the syntax

Element	Description
Index	Type: INT Number of the function generator: • **1 … 6**
Result	Type: REAL Function value of the function generator

3.297 $TIMER

Description

The variable allows a timer to be set and read out. The timer is started and stopped with the variable $TIMER_STOP (>>> *3.299 "$TIMER_STOP" Page 201*) and can then be read out using $TIMER.

The timer can be set forwards or backwards to any freely selected value.

Syntax

$TIMER[*Index*]=*Time*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the timer • **1 … 64**
Time	Type: INT; unit: ms Default: **0**

Example

```
$TIMER[4] = -50
$TIMER_STOP[4] = FALSE
WAIT FOR $IN[1] or $TIMER_FLAG[4]
```

3.298 $TIMER_FLAG

Description

Flag for the timer

The variable indicates whether the value of the timer is greater than or equal to zero.

$TIMER_FLAG can be used in wait or interrupt conditions that are to be triggered after a certain time has elapsed. If the corresponding timer is started with a negative value, $TIMER_FLAG changes from FALSE to TRUE at the zero crossing.

Syntax

`$TIMER_FLAG[`*Index*`]=`*State*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the timer • **1 … 64**
Status	Type: BOOL • TRUE: value greater than or equal to zero • FALSE: value less than zero

Example

```
$TIMER[4] = -50
$TIMER_STOP[4] = FALSE
WAIT FOR $IN[1] or $TIMER_FLAG[4]
```

3.299 $TIMER_STOP

Description

Starting and stopping of the timer

The timer is started or stopped when the advance run pointer has reached the line with the timer.

Syntax

`$TIMER_STOP[`*Index*`]=`*State*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the timer • **1 … 64**
State	Type: BOOL • TRUE: Timer stopped • FALSE: Timer started

Example

```
$TIMER[4] = -50
$TIMER_STOP[4] = FALSE
```

```
WAIT FOR $IN[1] or $TIMER_FLAG[4]
```

3.300 $TL_COM_VAL

Description

Tolerance time for limitation of the command speed

If after the tolerance time defined here the command velocity is greater than the limit defined by the variable $COM_VAL_MI, the error message *Command velocity {Axis number}* is displayed.

Syntax

`$TL_COM_VAL=`*Tolerance time*

Explanation of the syntax

Element	Description
Tolerance time	Type: INT; unit: ms Default: **50** **Note:** This value must not be changed.

3.301 $TOOL

Description

Position and orientation of the TOOL coordinate system (TCP) relative to the FLANGE coordinate system (reference coordinate system) in the advance run

For kinematic systems with fewer than 6 degrees of freedom, the degrees of freedom of $TOOL may also be limited (depending on the kinematic system type). In jog mode, the value for $TOOL is initially taken from the main run. If there is no main run, the value is taken from the advance run.

Main run copy of the variable: (>>> *3.302 "$TOOL_C" Page 203*)

Declaration

```
FRAME $TOOL
```

Explanation

Element	Description
$TOOL	Data type: FRAME TOOL coordinate system relative to the FLANGE coordinate system in the advance run

FRAME data type

```
STRUC FRAME REAL X, Y, Z, A, B, C
```

- X, Y and Z are the space coordinates, while A, B and C are the orientation of a coordinate system.
- The values are specified relative to a reference coordinate system.
- The rotations for the orientation are executed one after the other in the order A, B, C.
- The rotations following A refer to the coordinate system already rotated about A or about A and B.

Element	Description
X	Data type: REAL Translation along the X axis of the reference coordinate system [mm]
Y	Data type: REAL Translation along the Y axis of the reference coordinate system [mm]
Z	Data type: REAL Translation along the Z axis of the reference coordinate system [mm]
A	Data type: REAL Angle A: Rotation about the Z axis of the reference coordinate system [°]
B	Data type: REAL Angle B: Rotation about the Y axis of the reference coordinate system already rotated about A [°]
C	Data type: REAL Angle C: Rotation about the X axis of the reference coordinate system already rotated about A and B [°]

3.302 $TOOL_C

Description

Position and orientation of the TOOL coordinate system (TCP) relative to the FLANGE coordinate system (reference coordinate system) in the main run

Advance run copy of the variable: (>>> *3.301 "$TOOL" Page 202*)

Declaration

```
FRAME $TOOL_C
```

Explanation

Element	Description
$TOOL_C	Data type: FRAME TOOL coordinate system relative to the FLANGE coordinate system in the main run

FRAME data type

```
STRUC FRAME REAL X, Y, Z, A, B, C
```

- X, Y and Z are the space coordinates, while A, B and C are the orientation of a coordinate system.
- The values are specified relative to a reference coordinate system.
- The rotations for the orientation are executed one after the other in the order A, B, C.
- The rotations following A refer to the coordinate system already rotated about A or about A and B.

Element	Description
X	Data type: REAL Translation along the X axis of the reference coordinate system [mm]
Y	Data type: REAL Translation along the Y axis of the reference coordinate system [mm]
Z	Data type: REAL Translation along the Z axis of the reference coordinate system [mm]
A	Data type: REAL Angle A: Rotation about the Z axis of the reference coordinate system [°]
B	Data type: REAL Angle B: Rotation about the Y axis of the reference coordinate system already rotated about A [°]
C	Data type: REAL Angle C: Rotation about the X axis of the reference coordinate system already rotated about A and B [°]

3.303 $TORQ_DIFF

Description

Maximum torque deviation (force-induced torque)

During program execution, the values of $TORQ_DIFF (the difference between the setpoint torque and actual torque) are calculated. The calculated values are compared with the values from the previous program execution or with the default values. The highest value is saved.

If collision detection or torque monitoring is active, the system compares the values of $TORQ_DIFF with the saved values during the motion. The values are always calculated, even when collision detection or torque monitoring is deactivated.

Syntax

`$TORQ_DIFF[`*axis number*`] = `*deviation*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
deviation	Type: INT; unit: % **Note**: This value can only be set to the value "0" by the user. All other values are rejected.

3.304 $TORQ_DIFF2

Description

Maximum torque deviation (impact torque)

During program execution, the values of $TORQ_DIFF2 (the difference between the setpoint torque and the actual torque) are calculated. The calculated values are compared with the values from the previous program execution or with the default values. The highest value is saved.

If collision detection or torque monitoring is active, the system compares the values of $TORQ_DIFF2 with the saved values during the motion. The values are always calculated, even when collision detection or torque monitoring is deactivated.

Syntax

`$TORQ_DIFF2[`*axis number*`] = `*deviation*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
deviation	Type: INT; unit: % **Note**: This value can only be set to the value "0" by the user. All other values are rejected.

3.305 $TOOL_DIRECTION

Description

Configures the tool direction

By default, the X axis is defined as the tool direction. With this system variable, the tool direction can be configured as the X, Y or Z axis of the world coordinate system. The change in the tool direction is applicable to all tools. It is not possible to define different tool directions for different tools.

To ensure that the change in the tool direction takes effect for LIN and CIRC motions, the system variable $TOOL_DIRECTION_LIN_CIRC must also be used.

(>>> *3.306 "$TOOL_DIRECTION_LIN_CIRC" Page 206*)

Syntax

`$TOOL_DIRECTION = `*state*

Explanation of the syntax

Element	Description
state	Type: ENUM AXIS_OF_COORDINATES • #X • #Y • #Z Default: #X

3.306 $TOOL_DIRECTION_LIN_CIRC

Description

Configures the tool direction and the orientation control for LIN and CIRC motions

By default, the X axis is defined as the tool direction. With this system variable, the tool direction can be configured as the X, Y or Z axis of the tool coordinate system. The system variable uses the data stored in $TOOL_DIRECTION and is also included in calculation of the TTS for LIN and CIRC motions.

(>>> *3.305 "$TOOL_DIRECTION" Page 205*)

Syntax

`$TOOL_DIRECTION_LIN_CIRC =` *state*

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: the value stored in $TOOL_DIRECTION (#X, #Y, #Z) is used as the tool direction and for the orientation control. • FALSE: #X is used

3.307 $TORQMON

Description

Current factor for torque monitoring in program mode (force-induced torque)

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

The system variable $TORQMON contains the current tolerance range for the axis torques in program mode. This tolerance range is defined using the system variable $TORQMON_DEF in the file …STEU\Mada\$custom.dat.

(>>> *3.310 "$TORQMON_DEF" Page 208*)

Syntax

`$TORQMON[`*axis number*`] =` *factor*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
factor	Type: INT; unit: % Default: **0**

3.308 $TORQMON_COM

Description

Current factor of torque monitoring in jogging

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

The system variable $TORQMON_COM contains the current tolerance range for the axis torques in jogging. This tolerance range is defined using the system variable $TORQMON_COM_DEF in the file …STEU\Mada\$custom.dat.

(>>> *3.309 "$TORQMON_COM_DEF" Page 207*)

Syntax

`$TORQMON_COM[`*axis number*`] = `*factor*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
factor	Type: INT; unit: % Default: **0**

3.309 $TORQMON_COM_DEF

Description

Factor for torque monitoring in jogging

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

The system variable $TORQMON_COM_DEF defines the tolerance range for the axis torques in jogging. The width of the tolerance range is equal to the maximum torque [Nm] multiplied by the value of $TORQMON_COM_DEF. (Only for axes with valid model data)

Syntax

`$TORQMON_COM_DEF[`*axis number*`] = `*factor*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
factor	Type: INT; unit: % Default: **0**

3.310 $TORQMON_DEF

Description

Factor for torque monitoring in program mode (force-induced torque)

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

The system variable $TORQMON_DEF defines the tolerance range for the axis torques in program mode. The width of the tolerance range is equal to the maximum torque [Nm] multiplied by the value in $TORQMON_DEF. (Only for axes with valid model data)

Syntax

`$TORQMON_DEF[`*axis number*`] = `*factor*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
factor	Type: INT; unit: % Default: **0**

3.311 $TORQMON_TIME

Description

Response time for collision detection (force-induced torque monitoring)

Syntax

`$TORQMON_TIME = `*response time*

Explanation of the syntax

Element	Description
response time	Type: REAL; unit: ms Default: **0.0**

3.312 $TORQUE_AXIS_ACT

Description

Current torque of an axis (torque mode)

The displayed value is only relevant if the brakes are released. If the brakes are applied, it is virtually zero. (The state of the brakes can be displayed by means of the system variable $BRAKE_SIG. The value of $BRAKE_SIG is a bit array: bit 0 corresponds to A1, bit 6 corresponds to E1.)

(>>> *3.48 "$BRAKE_SIG" Page 38*)

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected. Its value is not dependent on the interpreter. In the robot program, the system variable triggers an advance run stop.

Syntax

torque = $TORQUE_AXIS_ACT[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
torque	Type: REAL; unit: Nm (for linear axes: N)

3.313 $TORQUE_AXIS_CMD

Description

Current setpoint torque of an axis (torque mode)

The displayed value is the setpoint torque calculated by the controller directly before it is transferred to the motor driver. This is the motor torque, but converted to output torque using the gear ratio. It thus also contains the correction due to elasticity compensation, for example.

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected. Its value is not dependent on the interpreter. In the robot program, the system variable triggers an advance run stop.

Syntax

torque = $TORQUE_AXIS_CMD[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
torque	Type: REAL; unit: Nm (for linear axes: N)

3.314 $TORQUE_AXIS_LIMITS

Description

Currently active motor torque limitation for an axis (torque mode)

The system variable contains the currently active limits programmed for torque mode with the SET_TORQUE_LIMITS() function.

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected and is primarily intended for diagnosis via the variable display or variable overview. In the robot program, the system variable triggers an advance run stop.

Syntax

torque limits = `$TORQUE_AXIS_LIMITS[`*axis number*`]`

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
torque limits	Type: STRUC TORQLIMITPARAM Currently active torque limits

TORQLIMITPARAM

```
STRUC TORQLIMITPARAM REAL LOWER, UPPER, SW_ONOFF MONI-
TOR, REAL MAX_VEL, MAX_LAG
```

Element	Description
LOWER	Type: REAL Lower torque limit Unit: Nm (for linear axes: N) Default: -1E10 (unlimited)
UPPER	Type: REAL Upper torque limit Unit: Nm (for linear axes: N) Default: 1E10 (unlimited)

Element	Description
MONITOR	Type: ENUM SW_ONOFF State of the regular monitoring functions • #ON: regular monitoring functions are activated. • #OFF: regular monitoring functions are deactivated. Instead, the monitoring functions MAX_VEL and MAX_LAG are activated. Default: #ON
MAX_VEL	Type: REAL Maximum permissible actual velocity in torque mode (only relevant if the regular monitoring functions are deactivated) Only a positive value may be programmed. Unit: degrees (for linear axes: mm) Default value (valid for all operating modes): T1 jog velocity * internal safety factor In T1, the maximum velocity with which jogging can be carried out is the default value, even if a higher value is programmed. **Note:** Only set a higher value than the default value if absolutely necessary.
MAX_LAG	Type: REAL Maximum permissible following error in torque mode (only relevant if the regular monitoring functions are deactivated) Only a positive value may be programmed. Unit: degrees (for linear axes: mm) Default value: 5 degrees (for linear axes: 100 mm) **Note:** Only set a higher value than the default value if absolutely necessary.

Properties:

- If there are currently no limits active, `UPPER` and `LOWER` remain non-initialized.
- The component `MONITOR` is always initialized unless the axis does not exist.

  This is relevant, for example, in the case of 4-axis and 5-axis robots: if the entire array is displayed, the non-existent axes can be easily identified.

  Non-existent external axes are simply not displayed when the entire array is displayed.
- `MAX_VEL` and `MAX_LAG` are non-initialized if `MONITOR` = #ON, as the regular monitoring functions are active in this case.

  If `MONITOR` = #OFF, the values of `MAX_VEL` and `MAX_LAG` are displayed. The display is irrespective of whether they have been set explicitly in the current program or whether the default values are being used.

The fact that certain components remain non-initialized under certain conditions, simplifies diagnosis for the user.
If the system variable $TORQUE_AXIS_LIMITS is accessed via KRL, however, the robot controller may regard the access as “invalid”. Recommendation: Check the state of the system variable with VARSTATE() prior to access.

3.315 $TORQUE_AXIS_MAX

Description

Absolute maximum torque of an axis (torque mode)

The value is a constant and is provided by the motor driver. It corresponds to the maximum motor characteristic (converted to output coordinates) in which the functional relationship between velocity and achievable drive torque are represented by a partially linear function.

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected. Its value is not dependent on the interpreter.

Syntax

torque = $TORQUE_AXIS_MAX[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
torque	Type: REAL; unit: Nm (for linear axes: N) The value specifies an interval: from -*value* to +*value*

3.316 $TORQUE_AXIS_MAX_0

Description

Maximum permanent torque of an axis at velocity 0 (torque mode)

The value is a constant and is provided by the motor driver. It does not normally correspond to the value of the motor characteristic at velocity 0, but is lower and takes into consideration derating effects of the inverters.

Further information about torque mode and the use of system variables is contained in the Operating and Programming Instructions for System Integrators.

Writability

The system variable is write-protected. Its value is not dependent on the interpreter.

Syntax

torque = $TORQUE_AXIS_MAX0[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
torque	Type: REAL; unit: Nm (for linear axes: N) The value specifies an interval: from -*value* to +*value*

3.317 $TRACE

Description

Trace recording (Oscilloscope)

The variable of structure type TRACE is written during a trace recording with the oscilloscope. Components of the variable can be used, for example, to start or stop the trace recording via a program.

Syntax

$TRACE = {NAME[] "*trace*", CONFIG[]"*config*", MODE *mode*, STATE *state*}

Explanation of the syntax

Element	Description
trace	Type: CHAR Name of the trace recording: max. 64 characters
config	Type: CHAR Name of the trace configuration: max. 64 characters
mode	Type: ENUM TRACE_MODE Recording mode: • #T_START: starts the recording. • #T_STOP: stops the recording. • #T_TRIGGER: starts the trigger process.
state	Type: ENUM TRACE_STATE Recording status: • #T_END: recording completed. • #TRIGGERED: recording in progress. • #T_WAIT: the recording is started and is waiting for the trigger. • #T_WRITING: the recorded data are written to the hard drive.

Example

```
WAIT FOR TRUE
$TRACE.MODE = #T_STOP
WAIT FOR $TRACE.STATE == #T_END
$TRACE.CONFIG[] = "TraceDEF_All.xml"
$TRACE.NAME[] = "TraceAll"
$TRACE.MODE = #T_START
WAIT FOR $TRACE.STATE <> #T_END
```

3.318 $TRACE_FOLDER

Description

This variable can be used to specify a subdirectory under **C:\KRC\Roboter\Trace** in which trace files are stored.

Syntax

$TRACE_FOLDER = "folder"

Explanation of the syntax

Element	Description
state	Type: CHAR • Subdirectory for trace files

Example

`$TRACE_FOLDER =" myTrace"`

Trace files are stored under **C:\KRC\Roboter\Trace\myTrace**.

`$TRACE_FOLDER = " "`

Trace files are stored under **C:\KRC\Roboter\Trace**.

3.319 $TRAFONAME

Description

Name of coordinate transformation

This name of the coordinate transformation is compared with the $ROBTRAFO[] robot name programmed on the RDC.

Writability

The system variable is write-protected.

Syntax

`$TRAFONAME[]="`*Name*`"`

Explanation of the syntax

Element	Description
Name	Type: CHAR (maximum 32 characters)

Example

```
$TRAFONAME[]="#KR16 C2 FLR ZH16"
```

3.320 $TSYS

Description

Position of the reference coordinate system of the function generator relative to BASE

The variable of structure type FRAME contains the current position of the reference coordinate system of a function generator relative to the BASE coordinate system.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

The variable is write-protected and is updated cyclically.

This variable is not relevant for spline function generators.

3.321 $TURN

Description

Dial mastering display

The variable indicates whether dial mastering is currently being performed. In the case of dial mastering, the output $ALARM_STOP is reset. The output $ALARM_STOP is set again when the dial mastering is completed.

Syntax

`$TURN` = *state*

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: mastering is performed. • FALSE: no mastering is performed. Default: FALSE

3.322 $T1

Description

Signal declaration for T1 mode

This output is set when operating mode T1 is selected.

Syntax

`SIGNAL $T1 $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **993**

3.323 $T2

Description

Signal declaration for T2 mode

This output is set when T2 mode is selected.

Syntax

```
SIGNAL $T2 $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **994**

3.324 $T2_ENABLE

Description

Signal declaration for reduction of the program override

This input is used by the higher-level controller to check the program override. If the higher-level controller resets the input, the program override is reduced to 10%.

Syntax

```
SIGNAL $T2_ENABLE $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

3.325 $T2_OV_REDUCE

Description

Override reduction when switching to T2 mode

If this option is activated, the override is automatically reduced to 10% when switching to T2. If the mode is changed from T1 to T2, the override value from T1 is saved and is available again the next time that T1 mode is set. This does not apply after a restart of the robot controller. After a restart, the default value for the override in T1 is 100%.

Syntax

```
$T2_OV_REDUCE=State
```

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: override is reduced. • FALSE: override is not reduced. Default: TRUE

3.326 $USER_ACTIVITY_COUNTER

Description

Counts the interactions of the user on the smartHMI. Interactions are:

- Moving a finger over the touch screen
- Pressing buttons

In the case of a MotionCoop Roboteam, the kernel system distributes the value to all controllers in the group. If a controller receives a new value, it derives the maximum from the value just received and the existing value and then sets the maximum. This information is important in order to be able to set $USER_LEVEL (>>> *3.327 "$USER_LEVEL" Page 217*) correctly at all times.

Writability

The system variable is write-protected.

Syntax

```
counter = $USER_ACTIVITY_COUNTER
```

Explanation of the syntax

Element	Description
counter	Type: INT Number of interactions

3.327 $USER_LEVEL

Description

User group currently set on the KUKA smartHMI

The system variable can, for example, be read in the program in order to trigger functions dependent on the user group.

When using a MotionCoop RoboTeam, the kernel system distributes the value to all controllers in the group. If a controller receives a new value, this is applied.

Writability

The system variable is write-protected.

Syntax

level = $USER_LEVEL

Explanation of the syntax

Element	Description
level	Type: INT For KSS: • **5**: Operator • **10**: User • **20**: Expert • **27**: Safety recovery • **29**: Safety maintenance • **30**: Administrator For VSS: • **5**: Operator • **10**: User • **29**: Safety maintenance • **30**: Expert

3.328 $USER_SAF

Description

Signal declaration for the safety fence monitoring

This output is reset if the safety fence monitoring switch is opened (AUT mode) or an enabling switch is released (T1 or T2 mode).

Syntax

SIGNAL $USER_SAF $OUT[*Output number*]

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1011**

3.329 $UTILIZATION: components for the Instantaneous load functionality

Description

$UTILIZATION specifies the components for the **Instantaneous load** functionality.

KSS:

VSS:

Data type: **Utilization_Struc**

All components are always valid.

$UTILIZATION cannot be used in interrupt declarations.

Writability

Only the components LONGTERM_MAX_RESET and MOVING_MAX_DURATION can be written.

Reading of the individual components and the entire structure is permissible.

Reading and writing is possible via the robot interpreter, submit interpreter and variable correction function. Reading or writing from the robot interpreter triggers an advance run stop.

A direct value assignment in the trigger is not possible.

Utilization_Struc

Component	Description
MOVING_MAX_SATURATED	Type: BOOL • TRUE: The buffer is completely filled with valid values. This is the case if values have been collected over at least the time period of MOVING_MAX_DURATION. • FALSE: Whenever not TRUE
LONGTERM_MAX_RESET Writable	Type: BOOL • TRUE: LONGTERM_MAX has been reset since the last cold start. • FALSE: No reset since the last cold start Setting to TRUE resets LONGTERM_MAX (again). This means that the reset is also carried out if LONGTERM_MAX_RESET is already TRUE. Writing FALSE has no effect.
	Setting to TRUE corresponds to pressing the **Reset** button in the **Instantaneous load** window.
CURRENT_VALUE	Type: REAL Current indicator value, i.e. the current maximum load value of all axes. The load value of an individual axis, in turn, is the maximum of internal indicator values for gear unit, drive and thermal loads. CURRENT_VALUE generally does not fall below an apparently relatively high “base load” even with a stationary robot. This can be explained as follows: • After a stop, the temperature generally decreases slowly, seldom reaching the minimum values. • The drive value decreases faster, but not immediately. • The dynamic gear torque disappears immediately, while the holding torque remains. If the temperature is the maximum of the 3 values during motion, CURRENT_VALUE will therefore only decrease slowly after the stop. If the drive vale is the maximum of the 3 values during motion, CURRENT_VALUE will decrease faster, but also not immediately.

Component	Description
MOVING_MAX	Type: REAL Maximum load that occurred in the last MOVING_MAX_DURATION interval (sliding maximum) The sliding maximum is reset by the following actions: Restart with hibernate, reconfiguration of the I/O drivers, modification of the machine data
	Corresponds to the **Maximum load in** [**Interval [s]**] display in the **Instantaneous load** window.
MOVING_MAX_CAUSE[]	Type: STRING Indicator of the cause of MOVING_MAX, e.g. “01T” The digits specify the dominant axis. The letter specifies the component (**D**rive, **G**ear or **T**emperature).
MOVING_MAX_DURATION Writable	Type: REAL, unit: s Interval length, writable: • **0.012 … 300** Default: 60
	Corresponds in principle to the **Interval [s]** box in the **Instantaneous load** window. However, a smaller range of values is available in the box than with MOVING_MAX_DURATION.
LONGTERM_MAX	Type: REAL Maximum load that occurred since the last system start or since the last setting of LONGTERM_MAX_RESET to TRUE
	Corresponds to the **Maximum load since** display in the **Instantaneous load** window.
LONGTERM_MAX_TIME[]	Type: STRING Time stamp of the maximum LONGTERM_MAX Format: YYYY.MM.DD HH:mm:ss.uuu
LONGTERM_START_TIME[]	Type: STRING Time stamp of the last reset by LONGTERM_MAX_RESET Format: YYYY.MM.DD HH:mm:ss.uuu
SHORTTERM_MAX	Type: REAL Maximum indicator value. The value is refreshed every 500 ms and refers to the period of 500 ms that has just elapsed. The value remains active until the next period of 500 ms has elapsed and is then refreshed again, etc.

3.330 $V_R1MADA

Description

Version identifier of the robot-specific machine data …\R1\Mada\$machine.dat

Syntax

`$V_R1MADA[]="`*Identifier*`"`

Explanation of the syntax

Element	Description
Identifier	Type: CHAR Version identifier: max. 32 characters

Example

```
$V_R1MADA[]="V1.1.2/KUKA8.2"
```

The identifier consists of the following components:

- Version of the file …\R1\Mada\$machine.dat
- Version of the KUKA System Software

3.331 $V_STEUMADA

Description

Version identifier of the controller-specific machine data …\STEU\Mada\$machine.dat

Syntax

`$V_STEUMADA[]="`*Identifier*`"`

Explanation of the syntax

Element	Description
Identifier	Type: CHAR Version identifier: max. 32 characters

Example

```
$V_STEUMADA[]="V1.1.2/KUKA8.2"
```

The identifier consists of the following components:

- Version of the file …\STEU\Mada\$machine.dat
- Version of the KUKA System Software

3.332 $VAR_TCP_IPO

Description

Remote laser on/off (optional)

If this option is activated, there must be a valid file VarTcpIpo.INI in the INIT directory of the robot controller.

Syntax

`$VAR_TCP_IPO=`*State*

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: option is activated. • FALSE: option is not activated. Default: FALSE

3.333 $VEL

Description

Velocity of the TCP in the advance run

The variable of structure type CP contains the programmed Cartesian velocity for the following components:

- CP: Path velocity in [m/s]
- ORI1: Swivel velocity in [°/s]
- ORI2: Rotational velocity in [°/s]

Limit values for the Cartesian velocity:

- **0.0 … $VEL_MA**
  The maximum Cartesian velocity $VEL_MA is defined in the machine data.

If $VEL violates the limit values, the message *Value assignment inadmissible* is displayed. Program execution is stopped or the associated motion instruction is not executed during jogging.

Example

```
$VEL={CP 2.0,ORI1 300.0,ORI2 300.0}
```

3.334 $VEL_C

Description

Velocity of the TCP in the main run

The variable of structure type CP contains the current Cartesian velocity for the following components:

- CP: Path velocity in [m/s]
- ORI1: Swivel velocity in [°/s]
- ORI2: Rotational velocity in [°/s]

Writability

The system variable is write-protected.

3.335 $VEL_ACT

Description

Current path velocity

Writability

The system variable is write-protected.

Syntax

velocity = $VEL_ACT

Explanation of the syntax

Element	Description
velocity	Type: REAL; unit: m/s • **0.0 … $VEL_MA.CP**

3.336 $VEL_APPL: changing the velocity of a CP motion in the main run

Description

$VEL_APPL can be used to modify the process velocity in the main run (i.e. during motion execution). The originally planned profile represents the upper limit and cannot be exceeded.

Field of application:

$VEL_APPL is particularly suitable for processes in which the required velocity only becomes known during program execution. Example: Sensor processes in which the velocity must be varied depending on the circumstances detected by the sensor.

In such processes, the range within which the velocity will lie overall is generally known. Recommendation:

- Define the maximum required process velocity using $VEL.CP.
- Define the lower velocity currently required in the main run using $VEL_APPL.CP.

Properties:

- $VEL_APPL can be applied to all CP motions (LIN, CIRC, CP spline blocks, SLIN, SCIRC).
- $VEL_APPL does not modify the acceleration and braking ramps.
- $VEL_APPL is reset to the default values in the following cases:
  Cold start, **I/O drivers** > **Reconfigure**, loading the machine data, selecting a program, resetting a program, deselecting a program

Constraints:

$VEL_APPL is not effective for BCO runs.

$VEL_APPL cannot be used in the WITH line of a spline command.

With KUKA.RoboTeam:

- $VEL_APPL cannot be used if the synchronization type MotionSync is active.
- $VEL_APPL cannot be used for motions on the LK base coordinate system.

Writability

The variable can be modified via the variable display and via the program.

Writing to the variable in the robot program triggers an advance run stop.

Writing in interrupt programs or via triggers and in the submit program does not trigger an advance run stop.

Syntax

```
$VEL_APPL.MODE = status
$VEL_APPL.CP = velocity
```

Explanation of the syntax

Element	Description
status	Type: ENUM • #OFF (default) • #ON
velocity	Type: REAL; unit: m/s • **0 … 3.0** Default: **0.0** **Note:** The originally planned profile represents the upper limit and cannot be exceeded. If $VEL_APPL.CP is greater than $VEL.CP, the controller takes $VEL.CP.

Example: simple velocity reduction

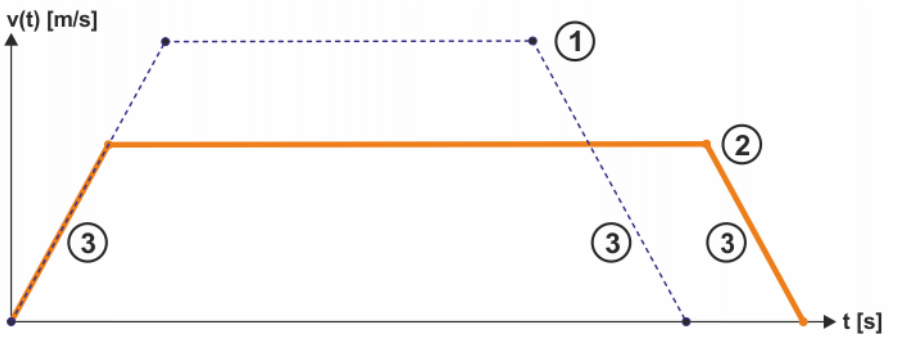


Fig. 3-10: Simple velocity reduction

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	The acceleration and braking ramps of the new profile are always identical to the original ramps.

Example: multiple changes in velocity

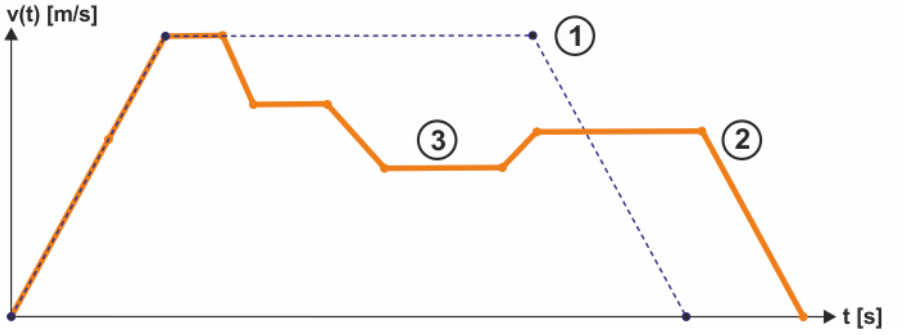


Fig. 3-11: Multiple change in velocity

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	Multiple changes in velocity are possible.

Example: Change in velocity in the original profile

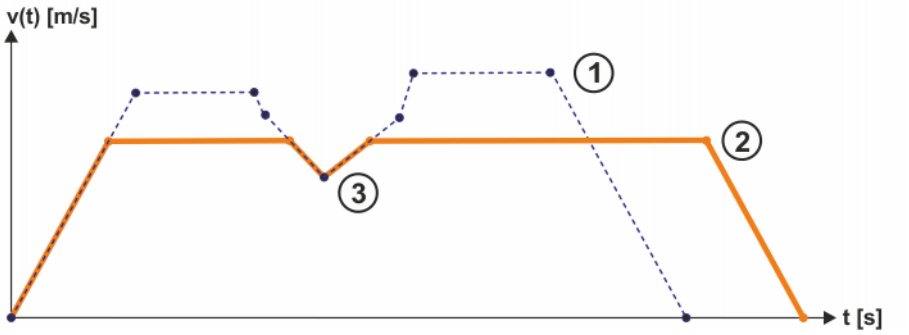


Fig. 3-12: Change in velocity in the original profile

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	Changes in velocity in the original profile only have an effect in the new profile where they lie below the limit set by $VEL_APPL.CP.

3.337 $VEL_AXIS

Description

Velocity of the robot axes in the advance run

The variable contains the programmed axis velocity as a percentage of the maximum motor speed $VEL_AXIS_MA.

Syntax

`$VEL_AXIS[`*axis number*`] = `*velocity*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6
velocity	Type: INT; unit: % • **1 … 100**

3.338 $VEL_AXIS_ACT

Description

Current velocity of the robot axis or external axis

The variable contains the current axis velocity as a percentage of the maximum motor speed $VEL_AXIS_MA.

Writability

The system variable is write-protected.

Syntax

speed = $VEL_AXIS_ACT[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
speed	Type: REAL; unit: % • **-100.0 … +100.0**

3.339 $VEL_AXIS_C

Description

Velocity of the robot axes in the main run

The variable contains the axis velocity of the motion currently being executed as a percentage of the maximum motor speed $VEL_AXIS_MA.

Writability

The system variable is write-protected.

Syntax

velocity = $VEL_AXIS_C[*axis number*]

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: robot axis A1 ... A6
velocity	Type: INT; unit: % • **1 … 100**

3.340 $VEL_EXTAX

Description

Velocity of the external axes in the advance run

The variable contains the programmed axis velocity as a percentage of the maximum motor speed $VEL_AXIS_MA.

Syntax

`$VEL_EXTAX[`*axis number*`] = `*velocity*

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: external axis E1 ... E6
velocity	Type: INT; unit: % • **1 … 100**

3.341 $VEL_EXTAX_C

Description

Velocity of the external axes in the main run

The variable contains the axis velocity of the motion currently being executed as a percentage of the maximum motor speed $VEL_AXIS_MA.

Writability

The system variable is write-protected.

Syntax

velocity `= $VEL_EXTAX_C[`*axis number*`]`

Explanation of the syntax

Element	Description
axis number	Type: INT • **1 … 6**: external axis E1 ... E6
velocity	Type: INT; unit: % • **1 … 100**

3.342 $WAIT_FOR

Description

WAIT FOR statement at which an interpreter is currently waiting

The variable $WAIT_FOR[] can be used to monitor which WAIT FOR statement the interpreter that is currently selected via $INTERPRETER is waiting at. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter

- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

Writability

The system variable is write-protected.

Example

TEST.SRC

```
; ...
WAIT FOR $OUT[1]
; ...
```

If you let this program run until the wait condition, you can have it displayed in the variable display by entering $WAIT_FOR[] – the result received is “WAIT FOR $OUT[1]” as a string

3.343 $WAIT_FOR1

Description

WAIT FOR statement at which the robot interpreter is currently waiting
The variable $WAIT_FOR1[] can be used to monitor which WAIT FOR statement the robot interpreter is currently waiting at.

Writability

The system variable is write-protected.

Example

TEST.SRC

```
; ...
WAIT FOR $OUT[1]
; ...
```

If you let this program run until the wait condition, you can have it displayed in the variable display by entering $WAIT_FOR1[] – the result received is “WAIT FOR $OUT[1]” as a string

3.344 $WAIT_FOR_INDEXRES

Description

State of the index resolution with reference to the WAIT FOR statement
The variable can be read and modified via the variable display function.

Syntax

`$WAIT_FOR_INDEXRES =` *state*

Explanation of the syntax

Element	Description
state	Type: ENUM WAITTYPE • #NO_RESOLUTION: interpreter is waiting at a condition that cannot be resolved. • #NO_WAIT: no wait condition is active, and thus also no index resolution. • #WAIT_INDEX_RES: interpreter is waiting at a condition and the index resolution is active. • #WAIT_NO_INDEX_RES: interpreter is waiting at a condition and the index resolution is not active.

3.345 $WAIT_FOR_ON

Description

State of an interpreter with reference to the WAIT FOR condition

The variable can be used to monitor whether the interpreter currently selected via $INTERPRETER is waiting at a WAIT FOR condition. The possible values for $INTERPRETER depend on the Submit mode.

Robot controller in Single Submit mode (default up to KSS 8.3):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (optional for KSS 8.3, default as of KSS 8.5):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

Writability

The system variable is write-protected.

Syntax

state = `$WAIT_FOR_ON`

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: interpreter is waiting. • FALSE: interpreter is not waiting.

3.346 $WAIT_FOR_ON1

Description

State of the robot interpreter with reference to the WAIT FOR condition

The variable can be used to monitor whether the robot interpreter is currently waiting at a WAIT FOR condition.

Writability

The system variable is write-protected.

Syntax

state = $WAIT_FOR_ON1

Explanation of the syntax

Element	Description
state	Type: BOOL • TRUE: interpreter is waiting. • FALSE: interpreter is not waiting.

3.347 $WAIT_STATE

Description

Active wait condition with reference to an interpreter

The variable can be used to monitor whether a wait condition is active and what type it is.

Depending on the specific interpreter, access to the information is as follows:

- Reading the variable in a robot program refers to the status of the robot interpreter.
- Reading the variable in a submit program refers to the status of the associated submit interpreter.
- Reading the variable by means of the variable display function refers to the current value of $INTERPRETER.

  The possible values for $INTERPRETER depend on the Submit mode that the robot controller is in.

  Robot controller in Single Submit mode:

  – 0: submit interpreter
  – 1: robot interpreter

  Robot controller in Multi-Submit mode:

  – 1: robot interpreter
  – 2: system submit interpreter
  – 3: extended submit interpreter 1
  – 4: extended submit interpreter 2
  – …
  – 9: extended submit interpreter 7

Writability

The system variable is write-protected.

Syntax

state = $WAIT_STATE

Explanation of the syntax

Element	Description
state	Type: ENUM WAIT_CMD_E • #NOT_WAITING : No wait condition is active. • #WAIT_WORKSPACE: Robot waits for a workspace to be enabled. • #WAIT_PROGSYNC: When a PROGSYNC command is reached, the robot waits for all cooperating robots to reach this command (only relevant in RoboTeam). • #WAIT_REMOTECMD: Robot waits for a remote statement, e.g. via the network. • #WAIT_BOOL_EXPR: Robot waits for a Boolean expression.

3.348 $WBOXDISABLE

Description

State of workspace monitoring

Syntax

`$WBOXDISABLE=`*State*

Explanation of the syntax

Element	Description
State	Type: BOOL • TRUE: Monitoring is active. • FALSE: Monitoring is not active. Default: FALSE

3.349 $WORKSTATE

Description

Signal declaration for monitoring of Cartesian workspaces

Each configured workspace must be assigned to a signal output. The output is set if a Cartesian workspace is violated.

Further information about configuring workspaces is contained in the Operating and Programming Instructions for System Integrators.

Syntax

`SIGNAL $WORKSTATE`*Index* `$OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Index	Type: INT Number of workspace • **1 … 8**
Output number	Type: INT By default, the output is deactivated with **FALSE**.

3.350 $WORLD

Description

WORLD coordinate system

The variable of structure type FRAME is write-protected and contains the origin coordinate system for the ROBROOT and BASE coordinate system.

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]
- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

By definition, the variable components are set to zero.

```
$WORLD={X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
```

Writability

The system variable is write-protected.

4 KUKA Service

4.1 Requesting support

Introduction

This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Information

The following information is required for processing a support request:

- Description of the problem, including information about the duration and frequency of the fault
- The greatest possible amount of information about the hardware and software components of the overall system

  The following list gives an indication of the information which is relevant in many cases:

  – Model and serial number of the kinematic system, e.g. the manipulator
  – Model and serial number of the controller
  – Model and serial number of the energy supply system
  – Designation and version of the system software
  – Designations and versions of other software components or modifications
  – System Software diagnosis package

    Additionally for KUKA Sunrise: Existing projects including applications

    For versions of KUKA System Software older than V8: Archive of the software (Diagnosis package is not yet available here.)
  – Application used
  – External axes used

4.2 KUKA Customer Support

The contact details of the local subsidiaries can be found at:
www.kuka.com/customer-service-contacts

Index

C

D

I

K

L

M

P

S

T

W