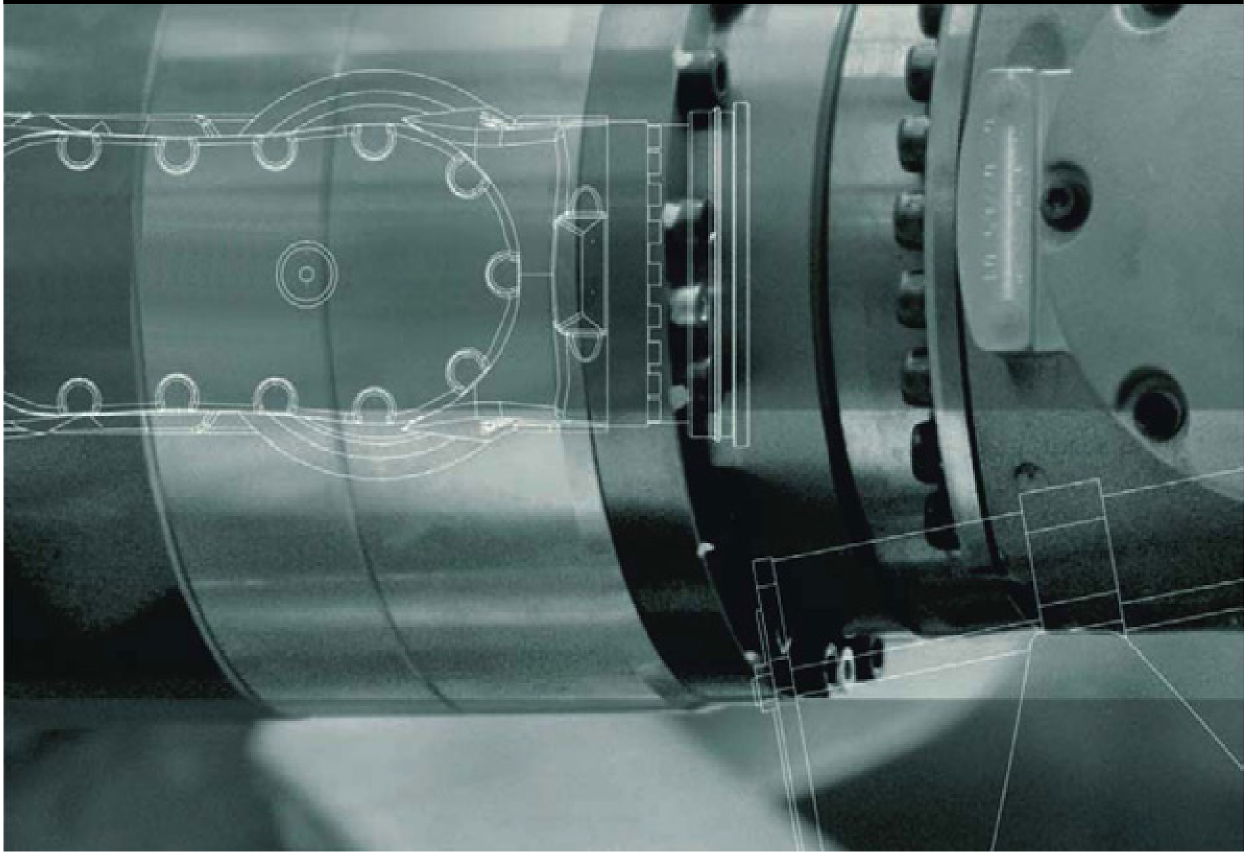
Expert Documentation

KUKA Roboter GmbH

# System Variables

## For KUKA System Software 8.1, 8.2, 8.3 and 8.4



Issued: 26.09.2016

Version: KSS 8.1, 8.2, 8.3, 8.4 System Variables V2

KUKA Roboter GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

Other functions not described in this documentation may be operable in the controller. The user has no claims to these functions, however, in the case of a replacement or service work.

We have checked the content of this documentation for conformity with the hardware and software described. Nevertheless, discrepancies cannot be precluded, for which reason we are not able to guarantee total conformity. The information in this documentation is checked on a regular basis, however, and necessary corrections will be incorporated in the subsequent edition.

Subject to technical alterations without an effect on the function.

Translation of the original documentation

KIM-PS5-DOC

| | |
|---|---|
| Publication: | Pub KSS 8.1, 8.2, 8.3, 8.4 Systemvariablen (PDF) en |
| Book structure: | KSS 8.1, 8.2, 8.3, 8.4 Systemvariablen V2.1 |
| Version: | KSS 8.1, 8.2, 8.3, 8.4 System Variables V2 |

# 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

> ℹ For optimal use of our products, we recommend that our customers take part in a course of training at 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 manipulator
- Documentation for the robot controller
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Parts catalog on storage medium

Each of these sets of instructions is a separate document.

## 1.3 Representation of warnings and notes

**Safety**

These warnings are relevant to safety 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 INSTRUCTIONS Procedures marked with this warning **must** be followed exactly.

**Notices**

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

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

## 1.4 Terms used

| Term | Description |
|---|---|
| HTTP | Hypertext Transfer Protocol<br>Protocol for transferring data via a network. |
| KCP | The KCP (KUKA Control Panel) teach pendant has all the operator control and display functions required for operating and programming the industrial robot.<br>The KCP variant for the KR C4 is called KUKA smartPAD. |
| SOAP | Simple Object Access Protocol<br>Protocol for exchanging XML-based messages via a network. Any transfer protocol can be used for sending the messages. Due to the greatest compatibility with other systems, HTTP is most commonly used. |
| TTS | Tool-based technological system<br>The TTS is a coordinate system that moves along the path with the robot. It is calculated every time a LIN or CIRC motion is executed. It is derived from the path tangent, the +X axis of the TOOL coordinate system and the resulting normal vector.<br>The tool-based moving frame coordinate system is defined as follows:<br>$\mathbf{X}_{\mathbf{TTS}}$: path tangent<br>$\mathbf{Y}_{\mathbf{TTS}}$: normal vector to the plane derived from the path tangent and the +X axis of the TOOL coordinate system<br>$\mathbf{Z}_{\mathbf{TTS}}$: vector of the right-angled system derived from $\mathbf{X}_{\mathbf{TTS}}$ and $\mathbf{Y}_{\mathbf{TTS}}$<br>The path tangent and the +X axis of the TOOL coordinate system must not be parallel, otherwise the TTS cannot be calculated. |

## 1.5 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.

⚠ The “Safety” chapter in the operating and programming instructions of the system software must be observed. Death to persons, severe injuries or considerable damage to property may otherwise result.

# 3 System variables

## 3.1 $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.

> The variable is write-protected and can only be read.

**Syntax**

$ABS_ACCUR=*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>■ #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.<br>■ #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).<br>■ #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. |

## 3.2 $ACC

**Description**

Acceleration of the TCP in the advance run

The variable of structure type CP contains the programmed Cartesian acceleration for the following components:

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

Limit values for Cartesian acceleration:

- **0.0 … $ACC_MA**
  The maximum Cartesian acceleration $ACC_MA is defined in the machine data (variable in the file …R1\Mada\$machine.dat).

> Further information about the variable $ACC_MA can be found in the documentation **Configuration of Kinematic Systems**.

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.

**Example**

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

## 3.3 $ACC_C

**Description**

Acceleration of the TCP in the main run

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

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

> The variable is write-protected and can only be read.

## 3.4 $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.

> Further information about the axis ramp-up time can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

$ACC_AXIS[*Axis number*]=*Acceleration*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6 |
| *Acceleration* | Type: INT; unit: %<br>- **1 … 100** |

## 3.5 $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.

> Further information about the axis ramp-up time can be found in the documentation **Configuration of Kinematic Systems**.

> The variable is write-protected and can only be read.

**Syntax**

$ACC_AXIS_C[*Axis number*]=*Acceleration*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6 |
| *Acceleration* | Type: INT; unit: %<br>- **1 … 100** |

## 3.6 $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 (variable in the file …R1\Mada\$machine.dat).

To ensure this, the monitoring of the Cartesian acceleration must be activated in the machine data: $ACC_CAR_STOP = TRUE (variable in the file …R1\Mada\$machine.dat)

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.

## 3.7 $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}
```

## 3.8 $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.

ℹ Further information about the axis ramp-up time can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

$ACC_EXTAX[*Axis number*]=*Acceleration*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: External axis E1 … E6 |
| *Acceleration* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.9 $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.

ℹ Further information about the axis ramp-up time can be found in the documentation **Configuration of Kinematic Systems**.

ℹ The variable is write-protected and can only be read.

**Syntax**

$ACC_EXTAX_C[*Axis number*]=*Acceleration*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: External axis E1 … E6 |
| *Acceleration* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.10 $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.

ℹ The variable is write-protected and can only be read.

**Syntax**

$ACCU_STATE=*Result*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Result* | Type: ENUM<br>- #CHARGE_OK: The battery test was positive.<br>- #CHARGE_OK_LOW: The battery test was positive but the battery was still not fully charged after the maximum charging time.<br>- #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.<br>- #CHARGE_TEST_NOK: The battery test was negative.<br>- #CHARGE_NOK: A battery test is not possible. The battery was still not fully charged after the maximum charging time.<br>- #CHARGE_OFF: No charging current. Either there is no battery present or the battery is defective. |

## 3.11 $ACT_ADVANCE

**Description**

Number of motion blocks currently planned in the main run

The maximum possible number of planned motion blocks depends on $ADVANCE (default: 3).

> The variable is write-protected and can only be read.

**Syntax**

`$ACT_ADVANCE=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT<br>- **1 … 5**<br>Default: **1 … 3** |

## 3.12 $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<br>- **1 … 5**<br>Default: **3** |

## 3.13 $ACT_EX_AX

**Description** Number of the current external BASE kinematic system

**Syntax** $ACT_EX_AX=*Kinematic system number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Kinematic system number* | Type: INT<br>■ **1 … 6** |

## 3.14 $ACT_BASE

**Description** Number of the current BASE coordinate system in the advance run

**Syntax** $ACT_BASE=*Base number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Base number* | Type: INT<br>■ **1 … 32**<br>**Note**: When selecting or resetting a robot program, the variable is set to the value -1. |

## 3.15 $ACT_BASE_C

**Description** Number of the current BASE coordinate system in the main run

The variable is write-protected and can only be read.

**Syntax** $ACT_BASE_C=*Base number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Base number* | Type: INT<br>■ **1 … 32**<br>**Note**: When selecting or resetting a robot program, the variable is set to the value -1. |

## 3.16 $ACT_TOOL

**Description** Number of the current TOOL coordinate system in the advance run

**Syntax** $ACT_TOOL=*Tool number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Tool number* | Type: INT<br>■ **1 … 16** |

## 3.17 $ACT_TOOL_C

**Description** Number of the current TOOL coordinate system in the main run

> The variable is write-protected and can only be read.

**Syntax**

`$ACT_TOOL_C=`*Tool number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Tool number* | Type: INT<br>■ **1 … 16** |

## 3.18 $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).

> The variable is write-protected and can only be read.

**Syntax**

`$ANIN[`*Input number*`]=`*Voltage*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>■ **1 … 32** |
| *Voltage* | Type: REAL<br>■ **-1.0 … +1.0** |

## 3.19 $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[`*Output number*`]=`*Voltage*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>■ **1 … 32** |
| *Voltage* | Type: REAL<br>■ **-1.0 … +1.0** |

## 3.20 $APO

**Description**

Approximation parameters in the advance run

This variable is used to define the approximation distance.

**Syntax**

`$APO={CVEL Velocity, CPTP DisPTP, CDIS DisCP, CORI Orientation}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| `CVEL` | Type: INT; unit: %<br>Velocity parameter<br>■ **1 … 100**<br>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: %<br>Approximation distance for PTP and PTP spline motions (= furthest distance before the end point at which approximate positioning can begin)<br>■ **1 … 100**<br>Explanation of the approximation parameter: (>>> "CPTP" Page 22)<br>**Note**: PTP spline motions (SPTP) can be programmed in KUKA System Software 8.3 or higher. |
| `CDIS` | Type: REAL; unit: mm<br>distance parameter<br>Approximation starts, at the earliest, when the distance to the end point falls below the value specified here. |
| `CORI` | Type: REAL; unit: °<br>Orientation parameter<br>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 (variable in the file …R1\Mada\$machine.dat). 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.

> Further information about the variable $APO_DIS_PTP can be found in the documentation **Configuration of Kinematic Systems**.

- 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.

## 3.21 $APO_C

**Description**

Approximation parameters in the main run

The variable contains the current approximation distance.

> The variable is write-protected and can only be read.

**Syntax**

`$APO_C={CVEL` *Velocity*`, CPTP` *DisPTP*`, CDIS` *DisCP*`, CORI` *Orientation*`}`

**Explanation of the syntax**

(>>> 3.20 "$APO" Page 21)

## 3.22 $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}
```

## 3.23 $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.

## 3.24 $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
```

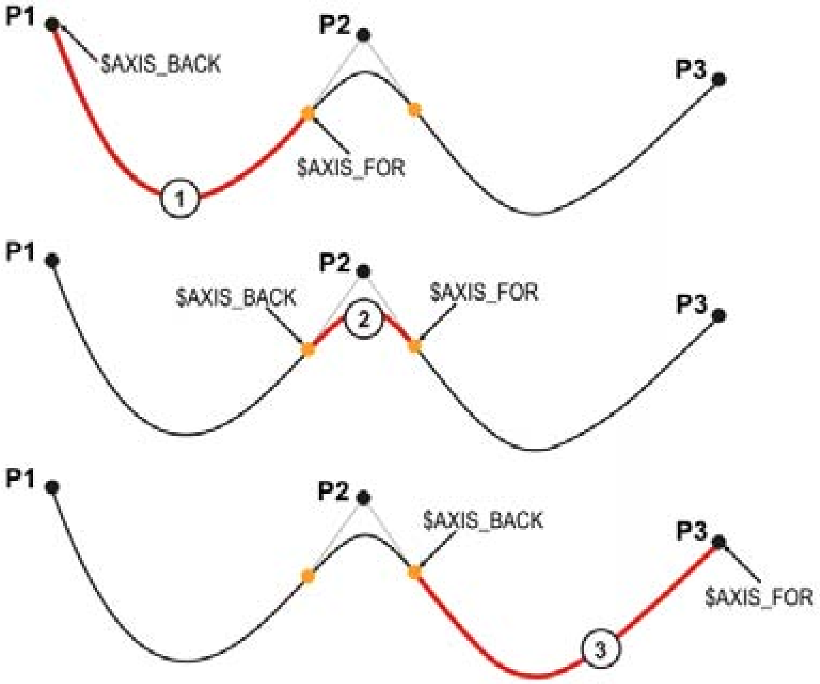


**Fig. 3-1: $AXIS_BACK, $AXIS_FOR – P2 is approximated**

| | | | |
|---|---|---|---|
| 1 | Single block | 3 | Following block |
| 2 | Intermediate block | | |

## 3.25 $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** (>>> 3.24 "$AXIS_BACK" Page 23)

## 3.26 $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.

## 3.27 $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 [°]

## 3.28 $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. The variable is write-protected.

## 3.29 $B_IN[]

**Description**

Value of a binary input

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>- **1 … 64** |
| *Value* | Type: INT<br>The range of values depends on the configuration of the binary input $BIN_IN[…] in the machine data (variable in the file …STEU\Mada\$custom.dat). |

## 3.30 $B_OUT[]

**Description**

Value of a binary output

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>- **1 … 64** |
| *Value* | Type: INT<br>The range of values depends on the configuration of the binary output $BIN_OUT[…] in the machine data (variable in the file …STEU\Mada\$custom.dat). |

**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<br>First bit – number of the first bit for which values can be set |
| LEN | Type: INT<br>Bit width – number of bits for the values to be set |
| PARITY | Type: ENUM<br>Parity bit<br>- #NONE: no parity<br>- #EVEN: even parity<br>- #ODD: odd parity |

## 3.31 $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 [°]

## 3.32 $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 [°]

> The variable is write-protected and can only be read.

## 3.33 $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.

> Information about the individual machine data can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

`$BASE_KIN[]="`*Information*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Information* | Type: CHAR<br>Name and external axes of the transformation: max. 29 characters |

## 3.34 $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.<br>■ **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.

## 3.35 $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**

For auxiliary points:

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

For end points:

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *BehaviorAUX* | Type: ENUM<br>■ **#INTERPOLATE**: The programmed orientation is accepted at the auxiliary point.<br>■ **#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.<br>■ **#CONSIDER**: The transition from the start orientation to the end orientation passes through the programmed orientation of the auxiliary point, i.e. the orientation of the auxiliary point is given at some point during the transition, but not necessarily at the auxiliary point.<br>Default: **#CONSIDER** |
| *BehaviorEND* | Type: ENUM<br>■ **#INTERPOLATE**: The programmed orientation of the end point is accepted at the actual end point.<br>(Only possibility for SCIRC without specification of circular angle. If #EXTRAPOLATE is set, #INTERPOLATE is nonetheless executed.)<br>■ **#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.<br>Default for SCIRC with specification of circular angle: **#EXTRAPOLATE** |

**Restrictions**

- 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°, i.e. 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**.

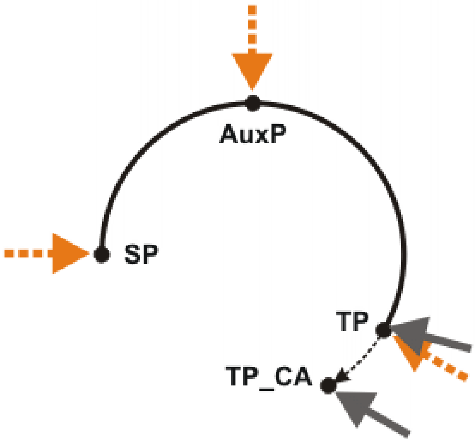


**Fig. 3-2: #INTERPOLATE**

| | |
|---|---|
| **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.

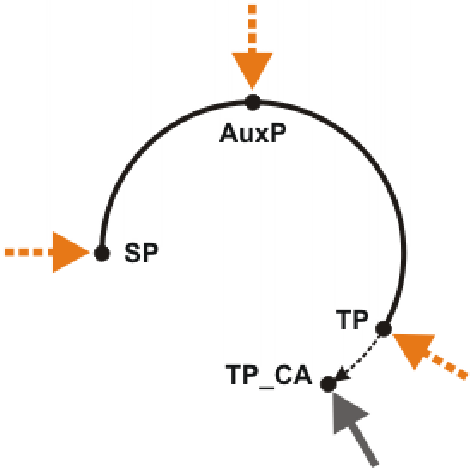


**Fig. 3-3: #EXTRAPOLATE**

## 3.36 $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<br>■ #BASE: Base-related orientation control<br>■ #PATH: Path-related orientation control |

## 3.37 $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.

> The variable is write-protected and can only be read.

**Syntax**

`$CIRC_TYPE_C=`*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>■ #BASE: Base-related orientation control<br>■ #PATH: Path-related orientation control |

## 3.38 $CMD

**Description**

Management number (handle) for command channel $CMD

The CWRITE() function can be used to write statements to the $CMD command channel. The variable itself is write-protected.

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

**Syntax** `$CMD=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT |

## 3.39 $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.

The variable is write-protected and can only be read.

**Syntax** `$COULD_START_MOTION=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: Robot can be moved.<br>■ FALSE: Robot cannot be moved. |

## 3.40 $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. The variable is write-protected.

**Syntax** `$CURR_ACT[`*Axis number*`]=`*Current*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 ... 6**: Robot axis A1 ... A6<br>■ **7 ... 12**: External axis E1 ... E6 |
| *Current* | Type: REAL; unit: %<br>■ **-100.0 … +100.0** |

## 3.41 $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<br>■ **1 … 256** |
| *Boolean expression* | Type: BOOL<br>■ Boolean expression: Cyclical flag is activated.<br>■ FALSE: Cyclical flag is deactivated.<br>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, i.e. 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.

## 3.42 $DATA_EXT_OBJ*x*

**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.

**Syntax**

`$DATA_LD_EXT_OBJ`*Index*=*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the data channel<br>■ **1 … 2** |
| *Number* | Type: INT<br>Number of data packets received via the channel |

## 3.43 $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.

The variable is write-protected and can only be read.

**Syntax**

$DATA_INTEGRITY=*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>Check of data consistency for input and output signals<br>■ TRUE: Check is activated.<br>■ FALSE: Check is not activated.<br>Default: TRUE |

## 3.44 $DATAPATH[]

**Description**

Extended compiler search path

Using the variable correction function, variables from the kernel system of the robot can be read and displayed. In order to display a runtime variable via the variable correction function, the compiler search path must be extended to the current program or the current interpreter environment. The name of the program is specified with $DATAPATH[].

**Syntax**

$DATAPATH[]="*Name*"

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Program name: max. 32 characters |

**Example**

**Program**: PALLETIZING.SRC

```
DEF Palletizing()
REAL Position
Position = 5.5
Pallet = 9.9
UP()
END

DEF UP()
REAL Start
Start = 1.1
END
```

**Data lists**: PALLETIZING.DAT and $CONFIG.DAT

```
DEFDAT Palletizing
REAL Origin = 7.7
ENDDAT

DEFDAT $CONFIG
REAL Pallet
ENDDAT
```

**Case 1**: The search path is extended to the program PALLETIZING.SRC.

```
...
$DATAPATH[] = "Palletizing"
...
```

If the program PALLETIZING.SRC is not selected, only the runtime variables of the program that are declared in the associated data list or in $CONFIG.DAT can be displayed using the variable correction function (Origin, Pallet).

**Case 2**: The search path is extended to the current interpreter environment.

```
...
$DATAPATH[] = "."
...
```

If the program PALLETIZING.SRC is not selected, all the runtime variables of the program, including the associated subprograms, can be displayed using the variable correction function (Position, Pallet, Start, Origin).

If the program PALLETIZING.SRC is selected, only the runtime variables declared in the program can be displayed using the variable correction function (Position).

## 3.45 $DATE

**Description**

System time and system date

**Syntax**

`$DATE={CSEC` *ms*`,SEC` *s*`,MIN` *min*`,HOUR` *h*`,DAY` *DD*`,MONTH` *MM*`,YEAR` *YYYY*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| — | Type: INT (all components) |

## 3.46 $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<br>■ 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.<br>■ FALSE: The positionally accurate robot model saved on the RDC is activated.<br>Default: FALSE |

## 3.47 $DEVICE

**Description**

Operating state of the connected teach pendant

**Syntax**

`$DEVICE=`*State*

**Explanation of the syntax**

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

## 3.48 $DISTANCE

**Description**

Arc length of a CP 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.

Type: REAL; unit: mm

> ℹ The variable is write-protected and can only be read.

## 3.49 $DIST_LAST – KUKA System Software 8.3 and higher

**Description**

Length of the path from the current TCP position to the previous taught point

Type: REAL. Unit:

- For CP motions (spline and conventional): mm
- For SPTP motions: No unit

> ℹ The variable is write-protected and can only be read.

**Use**

- $DIST_LAST cannot be used for PTP motions. The value is always zero in this case.
- $DIST_LAST can be used as an aid for programming PATH triggers with ONSTART. Further information about this is contained in the Operating and Programming Instructions for System Integrators.

## 3.50 $DIST_NEXT

**Description**

Length of the path from the current TCP position to the next taught point

Type: REAL. Unit:

- For CP motions (spline and conventional): mm
- For SPTP motions (KUKA System Software 8.3 and higher): No unit

> ℹ The variable is write-protected and can only be read.

**Use**

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

- $DIST_NEXT can be used as an aid for programming PATH triggers with ONSTART. Further information about this is contained in the Operating and Programming Instructions for System Integrators.

## 3.51 $DRIVES_ENABLE

**Description**

Switching drives on/off

**Syntax**

`$DRIVES_ENABLE=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Switches the drives on.<br>- FALSE: Switches the drives off. |

## 3.52 $ECO_LEVEL – KUKA System Software 8.3 and higher

**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)<br>With higher motion profile | Yes |
| CP spline motions (block and individual motion)<br>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 <> #NONE
- $OPT_MOVE <> #NONE

The default setting for both system variables is **<> #NONE**.

**Syntax** `$ECO_LEVEL=`*Level*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Level* | Type: ENUM<br>■ #OFF: Energy saving mode is deactivated.<br>■ #LOW: Low saving<br>■ #MIDDLE: Medium saving<br>■ #HIGH: High saving |

## 3.53 $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 fault service function.

The variable is write-protected and can only be read.

$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.

(>>> 3.53.2 "Examples of $ERR, ON_ERROR_PROCEED and ERR_RAISE()" Page 39)

**Syntax** $ERR=*Information*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Information* | Type: Error_T<br>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 mo-
dule[24], up_name[24], TRIGGER_UP_TYPE trigger_type
```

| Element | Description |
|---|---|
| number | Only in the event of a runtime error: Message number<br>If no error has occurred, the value zero is displayed. |
| interpreter | Current interpreter<br>The component can take the following values, regardless of whether the robot controller is operated in Single Submit mode (default operating mode) or in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher).<br>Robot controller in Single Submit mode:<br>■ #R_INT: Robot interpreter<br>■ #S_INT: Submit interpreter<br>Robot controller in Multi-Submit mode:<br>■ #R_INT: Robot interpreter<br>■ #S_INT: System submit interpreter<br>■ #EXT_S_INT1: Extended submit interpreter 1<br>■ #EXT_S_INT2: Extended submit interpreter 2<br>■ …<br>■ #EXT_S_INT7: Extended submit interpreter 7 |
| int_type | Current program type and interrupt state<br>■ #I_NORMAL: The program is not an interrupt program.<br>■ #I_INTERRUPT: The program is an interrupt program.<br>■ #I_STOP_INTERRUPT: Interrupt by means of $STOPMESS (error stop) |
| int_prio | Priority of the interrupt |
| line_nr | Only in the event of a runtime error: Number of the line that triggered the error<br>**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 “&”.<br>If no error has occurred, the value zero is displayed. |
| module[] | Name of the current program |
| up_name[] | Name of the current subprogram |
| trigger_type | Context in which the trigger belonging to a subprogram was triggered<br>■ #TRG_NONE: The subprogram is not a trigger subprogram.<br>■ #TRG_REGULAR: The trigger subprogram was switched during forward motion.<br>■ #TRG_BACKWARD: The trigger subprogram was switched during backward motion.<br>■ #TRG_RESTART: The trigger subprogram was switched on switching back to forward motion.<br>■ #TRG_REPLAY: The trigger subprogram was switched repeatedly after backward motion.<br>**Note**: This component is available in KUKA System Software 8.3 or higher. |

### 3.53.1 ON_ERROR_PROCEED

**Description**

ON_ERROR_PROCEED can be used to suppress a runtime error message triggered by the following program line. The robot controller skips the statement that triggers the error and fills the system variable $ERR with information about the error.

(>>> 3.53 "$ERR" Page 37)

Messages about internal errors or system errors cannot be suppressed.

ON_ERROR_PROCEED always applies to the following line, even if this is a blank line! Exception: If the following line contains the statement CONTINUE, ON_ERROR_PROCEED applies to the line after.

If the line after ON_ERROR_PROCEED is a subprogram call, the statement then refers to the call itself, and not to the first line of the subprogram.

**$ERR, ERR_RAISE()**

$ERR and ERR_RAISE() are important tools when working with ON_ERROR_PROCEED.

The function ERR_RAISE() can subsequently generate a suppressed runtime error message. It can only process the system variable $ERR or a variable derived from $ERR as an OUT parameter.

**Limitations**

ON_ERROR_PROCEED has no effect on motion statements:

SPLINE/ENDPLINE; PTP_SPLINE/ENDSPLINE; PTP; LIN; CIRC; PTP_REL; LIN_REL; CIRC_REL; ASYPTP; ASYSTOP; ASYCONT; ASYCANCEL; MOVE_EMI

ON_ERROR_PROCEED has no effect on the following control structures:

FOR/ENDFOR; GOTO; IF/ELSE/ENDIF; LOOP/ENDLOOP; REPEAT/UNTIL; SKIP/ENDSKIP; SWITCH/CASE/DEFAULT/ENDSWITCH; WHILE/ENDWHILE

**Syntax**

```
ON_ERROR_PROCEED
```

**Examples**

(>>> 3.53.2 "Examples of $ERR, ON_ERROR_PROCEED and ERR_RAISE()" Page 39)

**ON_ERROR_PROCEED with CONTINUE:**

```
ON_ERROR_PROCEED
CONTINUE
$OUT[1]=TRUE
```

```
CONTINUE
ON_ERROR_PROCEED
$OUT[1]=TRUE
```

The effect of both sequences of statements is identical. In both examples, ON_ERROR_PROCEED and CONTINUE act on $OUT[1]=TRUE.

### 3.53.2 Examples of $ERR, ON_ERROR_PROCEED and ERR_RAISE()

**Example 1**

If you do not wish to suppress all possible runtime error messages, but only specific ones, this distinction can be made using SWITCH … ENDSWITCH. In this example, only message 1422 is suppressed. Any other runtime error messages would be displayed.

```
DEF myProg ()
DECL E6POS myPos
INI
ON_ERROR_PROCEED
```

```
 5  myPos = $POS_INT
 6  SWITCH ($ERR.NUMBER)
 7    CASE 0
 8    CASE 1422
 9       ;program fault service function if required
         ...
10    DEFAULT
11       ERR_RAISE ($ERR)
12  ENDSWITCH
    ...
13  END
```

| Line | Description |
|---|---|
| 4, 5 | Line 5 triggers the message 1422 *{$variable} value invalid* (unless the program is called by an interrupt).<br>ON_ERROR_PROCEED in the preceding line suppresses the error message. |
| 6 … 12 | Differentiation dependent on $ERR.NUMBER |
| 7 | If no error occurred in line 5, $ERR.NUMBER==0. In this case, no action is required. |
| 8, 9 | If message 1422 has been triggered, $ERR.NUMBER==1422. If required, a fault service function can be programmed. |
| 10, 11 | If a message other than 1422 was triggered, this message is now (subsequently) generated via ERR_RAISE. |

**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 triggers the message 1422 *{$variable} value invalid* because myVar is not initialized and can thus not be transferred to a subprogram.<br>ON_ERROR_PROCEED in the preceding line suppresses the error message. |

| Line | Description |
|---|---|
| 6 | If $ERR is read here using the variable correction function, the following components have the following values:<br>$ERR.number == 1422<br>$ERR.line_nr == 15<br>$ERR.module[] == "MYMAINPROG"<br>$ERR.up_name[] == "MYMAINPROG" |
| 12 | If $ERR is read here in the subprogram using the variable correction function, the following components have the following values:<br>$ERR.number == 0<br>$ERR.line_nr == 0<br>$ERR.module[] == "MYMAINPROG"<br>$ERR.up_name[] == "MYSUBPROG"<br>This clearly indicates that $ERR always has the information from the current level (from the subprogram MySubProg in this case). The information from MyMainProg, on the other hand, is unknown. |

**Example 3**

This example also shows that each program level has its own representation of $ERR. The example also shows how the $ERR information can be transferred to a different level.

```
 1  DEF myMainProg2 ()
 2  INI
 3  ON_ERROR_PROCEED
 4  $OUT[-10] = TRUE
 5  myHandleErr ($ERR, $ERR)
 6  END
---------------------------------------
 7  DEF myHandleErr (inErr:IN, outErr:OUT)
 8  DECL Error_T inErr, outErr
 9  ON_ERROR_PROCEED
10  $OV_PRO=100/0
11  ERR_RAISE($ERR)
12  ERR_RAISE(outErr)
13  ERR_RAISE(inErr)
    ...
14  END
```

| Line | Description |
|---|---|
| 3, 4 | Line 4 triggers the message 1444 *Array index inadmissible*.<br>ON_ERROR_PROCEED in the preceding line suppresses the error message. |
| 5, 7 | The contents of $ERR are transferred to a subprogram twice: once as an IN parameter and once as an OUT parameter. |
| 9, 10 | Line 10 triggers the message 1451 *Division by 0*.<br>ON_ERROR_PROCEED in the preceding line suppresses the error message. |
| 11 | ERR_RAISE($ERR) generates the message from line 10, and not that from line 4.<br>$ERR always has the information from the current level. In this case, from the subprogram myHandleErr. |

| Line | Description |
|---|---|
| 12 | ERR_RAISE(outErr) generates the message from line 4 of the main program, as outErr is a reference to $ERR in the main program. |
| 13 | ERR_RAISE(inErr) is not permissible and thus triggers the message 1451 *{(Variable name)} invalid argument*.<br>ERR_RAISE can only process $ERR or an OUT variable derived from $ERR. |

**Example 4**

$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 ()
...
mySUB (55)
...
END
```

Submit program:

```
DEF SPS ()
...
LOOP
   mySUB (33)
   ...
ENDLOOP
...
END
```

Subprogram:

```
GLOBAL DEF mySUB (par:IN)
INT par
INI
IF ($ERR.INTERPRETER==#R_INT) THEN
   $OUT_C[par] = TRUE
   ELSE
   $OUT[par] = TRUE
ENDIF
...
END
```

## 3.54 $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.<br>■ **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.

## 3.55 $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.

**Syntax**

`$FAST_MEAS_COUNT[`*Index*`]=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the input for fast measurement<br>- **1 … 8** |
| *Number* | Type: INT<br>Number of measurements |

## 3.56 $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=`*Reset*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reset* | Type: BOOL<br>- TRUE: Resets counter and time stamp.<br>- FALSE: Reset is not active.<br>Default: FALSE |

## 3.57 $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[]. The variable is write-protected.

**Syntax**

`$FAST_MEAS_COUNT_TIME[`*Index*`]=`*Time stamp*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the input for fast measurement<br>- **1 … 8** |
| *Time stamp* | Type: REAL |

## 3.58 $FILTER

**Description**

Programmed smooth ramp in the advance run

The variable can be used to set the filter value for motions in the robot program. The filter prevents an abrupt increase to the maximum acceleration value. (Jerk is limited.) The motion time of a motion instruction is extended by the value of $FILTER. The value is always a multiple of (IPO cycle - 1).

**Syntax**

`$FILTER=`*Filter value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Filter value* | Type: INT; unit: ms<br>■ **0**: Filter is deactivated.<br>■ **1 … 851**<br>Default: 150 |

## 3.59 $FILTER_C

**Description**

Currently valid smooth ramp in the main run

The variable displays the currently valid filter value for motions.

> The variable is write-protected and can only be read.

**Syntax**

`$FILTER_C=`*Filter value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Filter value* | Type: INT; unit: ms<br>■ **0**: Filter is deactivated.<br>■ **1 … 851**<br>Default: 150 |

## 3.60 $FLAG[]

**Description**

Management of flags

A Boolean expression can be freely assigned to the variable $FLAG[] in a robot or submit program. 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<br>■ **1 … 1,024** |
| *Boolean expression* | Type: BOOL<br>■ TRUE, FALSE or other Boolean expression<br>Default: FALSE |

## 3.61 $FOL_ERROR[]

**Description**

Following error of an axis relative to the velocity

**Syntax**

`$FOL_ERROR[`*Axis number*`]=`*Following error*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Following error* | Type: REAL; unit: ms |

## 3.62 $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. The variable itself is write-protected.

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

**Syntax**

`$FCT_CALL=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT |

## 3.63 $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*`]=`*Gear jerk*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 … E6 |
| *Gear jerk* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.64 $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.

> The variable is write-protected and can only be read.

**Syntax**

`$GEAR_JERK_C[`*Axis number*`]=`*Gear jerk*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 … E6 |
| *Gear jerk* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.65 $HOLDING_TORQUE[] – KUKA System Software 8.2 and higher

> 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<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 … E6 |
| *Holding torque* | Type: REAL; unit: Nm (for linear axes: N)<br>**Note**: For external axes, the value 0 N is always returned, as there are currently no model data defined for them. |

## 3.66 $HOLDING_TORQUE_MAND[] – KUKA System Software 8.2 and higher

> 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<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 ... E6 |
| *State* | Type: BOOL<br>- TRUE: Monitoring is activated.<br>- FALSE: Monitoring is deactivated.<br>Default: TRUE |

## 3.67 $HOME[]

**Description**

HOME directory of the compiler

**Syntax**

`$HOME[]="`*Directory*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Directory* | Type: CHAR (max. 3 characters)<br>- **/R1**: Robot system 1<br>- **/R2**: Robot system 2<br>Default: **/R1** |

## 3.68 $IN[]

**Description**

Digital input states

The variable can be used to poll the current state of a digital input in a robot or submit program or via the variable correction function. The variable is write-protected.

**Syntax**

`$IN[`*Input number*`]=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>- **1 … 4 096**: KUKA System Software 8.1<br>- **1 … 8 192**: KUKA System Software 8.2 and higher<br>**Note**: The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be monitored via $NUM_IN/$NUM_OUT. |
| *State* | Type: BOOL<br>- TRUE or FALSE |

## 3.69 $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.<br>- **Bit n = 0**: Axis still moving<br>- **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 |

## 3.70 $INTERPRETER

**Overview**

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.

**Single Submit mode**

A robot controller runs in Single Submit mode by default. This means that 2 tasks run on it simultaneously:

- Robot interpreter

  The motion programs run in the robot interpreter.
- System submit interpreter

  The program SPS.SUB runs via the system submit interpreter. The SPS.SUB program is intended for controller-internal submit tasks.

**Multi-Submit mode**

KUKA System Software 8.3 can be used with the KUKA.MultiSubmitInterpreter technology package and the robot controller can be operated in Multi-Submit mode. This means: in addition to the system submit interpreter, there are 7 further submit interpreters available, so-called extended submits. Overall, the following interpreters therefore exist:

- 1 robot interpreter

  The motion programs run in the robot interpreter.
- 1 system submit interpreter

  The program SPS.SUB runs via the system submit interpreter. The SPS.SUB program is intended for controller-internal submit tasks.
- 7 extended submit interpreters

  The extended submits are intended for user-specific submit tasks. Users can create their own SUB programs and assign these to the extended submits.

Further information about the Multi-Submit mode can be found in the **KUKA.MultiSubmitInterpreter** documentation.

### 3.70.1 $INTERPRETER in Single Submit mode

**Description**

Selecting the interpreter

By default, the execution of a selected SUB program is not displayed in the editor. This can be modified by selecting the submit interpreter via the variable $INTERPRETER. A motion program must be selected at the same time so that the SUB program is displayed in the editor.

**Syntax**

`$INTERPRETER=`*Interpreter*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Interpreter* | Type: INT<br>Selecting the interpreter<br>- 0: Submit interpreter<br>- 1: Robot interpreter<br>Default: 1 |

### 3.70.2 $INTERPRETER in Multi-Submit mode

**Description**

Selecting the interpreter

By default, the execution of a selected SUB program is not displayed in the editor. This can be modified by selecting the corresponding submit interpreter (system submit or extended submit) via the variable $INTERPRETER. A motion program must be selected at the same time so that the SUB program is displayed in the editor.

**Syntax**

`$INTERPRETER=`*Interpreter*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Interpreter* | Type: INT<br>Selecting the interpreter<br>- 1: Robot interpreter<br>- 2: System submit interpreter<br>- 3: Extended submit interpreter 1<br>- 4: Extended submit interpreter 2<br>- …<br>- 9: Extended submit interpreter 7<br>Default: 1 |

**Changed system variables**

In Multi-Submit mode, the meaning of certain system variables is changed compared with the Single Submit mode. Other system variables have been added or removed.

Changed system variables:

- $INTERPRETER
- $PRO_STATE0

  Represents the state of the group indicator in the status bar. The variable is write-protected.

- $ERR

  $ERR is a system variable of structure type **Error_T**. The type **Error_T** has a component which represents the current interpreter.

  In Single Submit mode, only the values #R_INT and #S_INT are possible for this component.

  For Multi-Submit mode, the range of possible values has been expanded to include the extended submits:

  - #EXT_S_INT1
  - …
  - #EXT_S_INT7

**Removed system variables**

Removed system variables (with the postfix “0”):

- $PRO_IP0
- $PRO_NAME0[]
- $PRO_MODE0
- $WAIT_FOR_ON0
- $WAIT_FOR0[]

**Explanation:**

The states represented by the variables $PRO_*xxx*0 in Single Submit mode are represented in Multi-Submit mode by $PROG_INFO[].

The system variables of the same name with the postfix “1” ($PRO_IP1, etc.) are retained unchanged.

The variables $WAIT_*xxx*0 have been removed altogether.

**New system variable**

New system variable:

- $PROG_INFO[] (>>> 3.143 "$PROG_INFO[]" Page 84)

## 3.71 $IOBUS_INFO[]

**Description**

Structure with information about the bus driver

> The variable is write-protected and can only be read.

**Syntax**

`$IOBUS_INFO[`*Index*`]=`*Information*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Bus driver number<br>■ **1 … 32**<br>The serial number is automatically assigned to the bus drivers. |
| *Information* | Type: Iobus_Info_T<br>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 |
| bus_ok | ■ TRUE: Bus driver is OK.<br>■ FALSE: Bus driver is faulty or incompatible. |
| bus_installed | ■ TRUE: Bus driver is installed.<br>■ FALSE: Bus driver is not installed. |

## 3.72 $IOSIM_IN[]

**Description**

State of the digital inputs with simulation calculated

The variable can be used to poll the current state of a digital input in the robot program or via the variable correction function. It also specifies whether or not the input is simulated. The variable is write-protected.

**Syntax**

`$IOSIM_IN[`*Input number*`]="`*State*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>■ **1 … 4 096**: KUKA System Software 8.1<br>■ **1 … 8 192**: KUKA System Software 8.2 and higher<br>**Note**: The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be monitored via $NUM_IN/$NUM_OUT. |
| *State* | Type: CHAR<br>■ **0**: FALSE and not simulated<br>■ **1**: TRUE and not simulated<br>■ **2**: FALSE and simulated<br>■ **3**: TRUE and simulated<br>■ **4**: FALSE and not simulated and system input<br>■ **5**: TRUE and not simulated and system input<br>■ **6**: FALSE and simulated and system input<br>■ **7**: TRUE and simulated and system input |

## 3.73 $IOSIM_OPT

**Description**

Activation or deactivation of simulation

**Precondition**

- KUKA.OfficeLite is used or an image of the system software is running on the office PC.

**Syntax**

`$IOSIM_OPT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ FALSE: The simulation is deactivated.<br>■ TRUE: Simulation is activated.<br>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.
- The robot controller processes both simulated input signals and real input signals. If an input has been mapped to a robot controller output, the simulated input also sets the physical output!
- 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.

### 3.73.1 Simulating inputs/outputs – KUKA System Software 8.2 and higher

**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.

### 3.73.2 Simulating inputs/outputs – KUKA System Software 8.1

**Example 1**

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

1. The simulated state of the output is set to TRUE. ($OUTSIM_TBL[8]=#SIM_TRUE)
2. Simulation is activated. ($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]==#NONE, 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.

## 3.74 $IOSIM_OUT[]

**Description**

State of the digital outputs with simulation calculated

The variable can be used to poll the current state of a digital output in the robot program or via the variable correction function. It also specifies whether or not the output is simulated. The variable is write-protected.

**Syntax**

`$IOSIM_OUT[`*Output number*`]="`*State*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>■ **1 … 4 096**: KUKA System Software 8.1<br>■ **1 … 8 192**: KUKA System Software 8.2 and higher<br>**Note**: The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be monitored via $NUM_IN/$NUM_OUT. |
| *State* | Type: CHAR<br>■ **0**: FALSE and not simulated<br>■ **1**: TRUE and not simulated<br>■ **2**: FALSE and simulated<br>■ **3**: TRUE and simulated<br>■ **4**: FALSE and not simulated and system input<br>■ **5**: TRUE and not simulated and system input<br>■ **6**: FALSE and simulated and system input<br>■ **7**: TRUE and simulated and system input |

## 3.75 $IOSYS_IN_FALSE

**Description**

Number of the system input that is always FALSE

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

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

> The variable is write-protected and can only be read.

**Syntax**

$IOSYS_IN_FALSE=*Input number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>Default: **1026** |

## 3.76 $IOSYS_IN_TRUE

**Description**

Number of the system input that is always TRUE

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

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

> The variable is write-protected and can only be read.

**Syntax**

$IOSYS_IN_TRUE=*Input number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>Default: **1025** |

## 3.77 $IPO_MODE

**Description**

Programmed interpolation mode in the advance run

**Syntax**

$IPO_MODE=*Mode*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mode* | Type: ENUM<br>■ #TCP: The tool is a fixed tool.<br>■ #BASE: The tool is mounted on the mounting flange.<br>Default: #BASE |

## 3.78 $IPO_MODE_C

**Description**

Current interpolation mode in the main run

> The variable is write-protected and can only be read.

**Syntax**

$IPO_MODE_C=*Mode*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mode* | Type: ENUM<br>■ #TCP: The tool is a fixed tool.<br>■ #BASE: The tool is mounted on the mounting flange.<br>Default: #BASE |

## 3.79 $IS_OFFICE_LITE

**Description**

Identifier of the System Software as OfficeLite version

**Syntax**

`$IS_OFFICE_LITE=`*Identifier*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Identifier* | Type: BOOL<br>- TRUE: Software is KUKA.OfficeLite.<br>- FALSE: Software is KUKA System Software. |

## 3.80 $I2T_OL

**Description**

Deactivation of $I^2$t monitoring (only permissible in KUKA.OfficeLite)

**Syntax**

`$I2T_OL=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM SW_ONOFF<br>- #ON: $I^2$t monitoring is activated.<br>- #OFF: $I^2$t monitoring is not activated.<br>Default: #ON |

## 3.81 $JERK – KUKA System Software 8.1

**Description**

Cartesian jerk limitation for SPLINE in the advance run

The variable of structure type JERK_STRUC limits the change of the acceleration over time during CP spline motions (SLIN, SCIRC).

> The variable is only relevant in the case of CP spline motions that are planned without a dynamic model. (For KUKA System Software 8.2 and higher, the variable is no longer relevant.)

The aggregate consists of the following components:

- CP: Change to the path acceleration in [m/s$^3$]
- ORI: Change to the orientation acceleration in [°/s$^3$]
- AX: Change to the axis acceleration in [°/s$^3$] for rotational axes or in [m/s$^3$] for linear axes

The maximum permissible jerk for spline motions is defined in the machine data (variable $JERK_MA in the file …R1\Mada\$machine.dat).

**Example**

```
$JERK={CP 50.0,ORI 50000.0,AX {A1 1000.0,A2 1000.0,A3 1000.0,A4
1000.0,A5 1000.0,A6 1000.0,E1 1000.0,E2 1000.0,E3 1000.0,E4 1000.0,E5
1000.0,E6 1000.0}}
```

## 3.82 $JERK_C – KUKA System Software 8.1

**Description**

Cartesian jerk limitation for SPLINE in the main run

The variable of structure type JERK_STRUC contains the current value for the change of the acceleration over time during a CP SPLINE motion (SLIN, SCIRC).

> The variable is only relevant in the case of CP spline motions that are planned without a dynamic model. (For KUKA System Software 8.2 and higher, the variable is no longer relevant.)

The aggregate consists of the following components:

- CP: Change to the path acceleration in [m/s$^3$]
- ORI: Change to the orientation acceleration in [°/s$^3$]
- AX: Change to the axis acceleration in [°/s$^3$] for rotational axes or in [m/s$^3$] for linear axes

> The variable is write-protected and can only be read.

## 3.83 $KCP_CONNECT – KUKA System Software 8.1

> The system variable is still present in higher versions of the software, but always delivers TRUE. For KUKA System Software 8.2 and higher, use the system variable $KCP_TYPE instead.

**Description** Displays whether a KUKA smartPAD is connected.

**Syntax** `$KCP_CONNECT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: smartPAD is connected.<br>■ FALSE: No smartPAD is connected. |

## 3.84 $KCP_IP – KUKA System Software 8.2 and higher

**Description** IP address of the currently connected KUKA smartPAD

**Syntax** `$KCP_IP=`*IP address*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *IP address* | Type: INT<br>If the value zero is displayed, no smartPAD is connected. |

## 3.85 $KCP_TYPE – KUKA System Software 8.2 and higher

**Description** Currently connected teach pendant

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

**Syntax** `$KCP_TYPE=`*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>- #NO_KCP: No teach pendant is connected.<br>- #KCP: KUKA smartPAD is connected.<br>- #VRP: A virtual KUKA smartPAD is connected (KUKA.VirtualRemotePendant). |

## 3.86 $KDO_ACT

**Description** Indication of whether a command motion is currently active

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

**Syntax** `$KDO_ACT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Command motion active<br>- FALSE: No command motion active |

## 3.87 $KR_SERIALNO

**Description** Robot serial number saved on the RDC

**Syntax** `$KR_SERIALNO=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT<br>Maximum 24 characters |

## 3.88 $LDC_ACTIVE – KUKA System Software 8.2 and higher

**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 |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Online load data verification is activated.<br>- FALSE: Online load data verification is deactivated.<br>Default: TRUE |

## 3.89 $LDC_LOADED – KUKA System Software 8.2 and higher

**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 |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Online load data verification is loaded.<br>- FALSE: Online load data verification is not loaded. |

## 3.90 $LDC_RESULT[] – KUKA System Software 8.2 and higher

**Description**

Current result of the online load data verification

**Syntax**

`$LDC_RESULT[`*Index*`]=` *Result*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Tool number<br>- **1 … 32** |
| *Result* | Type: CHAR<br>- #OK: The payload is OK. (Neither overload nor underload.)<br>- #OVERLOAD: There is an overload.<br>- #UNDERLOAD: There is an underload.<br>- #CHECKING: The load data verification is still active.<br>- #NONE: There is currently no result, e.g. because the tool has been changed. |

## 3.91 $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 59)

**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<br>- **X**, **Y**, **Z**: Position of the center of gravity relative to the flange<br>- **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange |
| *Inertia* | Type: INERTIA<br>- **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

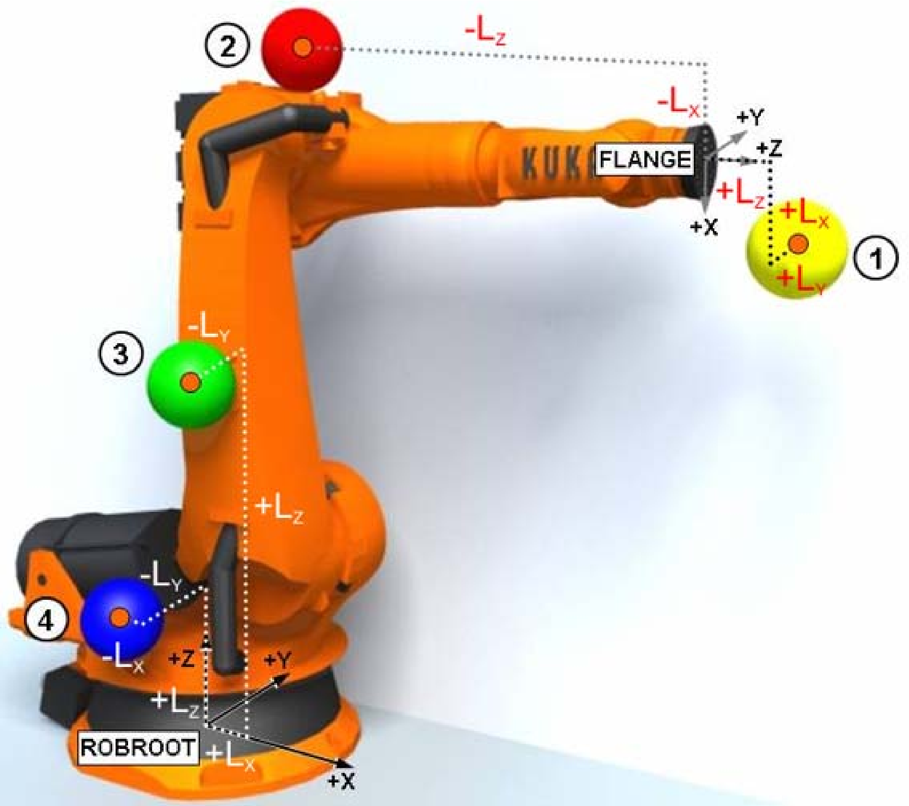


**Fig. 3-4: Loads on the robot**

1 Payload

2 Supplementary load on axis 3

3 Supplementary load on axis 2

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<br>A4 = 0°, A5 = 0°, A6 = 0° |
| Supplementary load A2 | ROBROOT coordinate system<br>A2 = -90° |
| Supplementary load A1 | ROBROOT coordinate system<br>A1 = 0° |

## 3.92 $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 59)

> The variable is write-protected and can only be read.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mass* | Type: REAL; unit: kg |
| *Center of gravity* | Type: FRAME<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the flange<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange |
| *Inertia* | Type: INERTIA<br>■ **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 |

## 3.93 $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 in the robot controller.

(>>> "Loads on the robot" Page 59)

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<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base |
| *Inertia* | Type: INERTIA<br>■ **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. |

## 3.94 $LOAD_A1_C

**Description**

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

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

(>>> "Loads on the robot" Page 59)

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

> The variable is write-protected and can only be read.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mass* | Type: REAL; unit: kg |
| *Center of gravity* | Type: FRAME<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base |
| *Inertia* | Type: INERTIA<br>■ **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. |

## 3.95 $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 in the robot controller.

(>>> "Loads on the robot" Page 59)

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<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base |
| *Inertia* | Type: INERTIA<br>■ **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. |

## 3.96 $LOAD_A2_C

**Description**

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

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

(>>> "Loads on the robot" Page 59)

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°.

> The variable is write-protected and can only be read.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mass* | Type: REAL; unit: kg |
| *Center of gravity* | Type: FRAME<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the robot base<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the robot base |
| *Inertia* | Type: INERTIA<br>■ **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. |

## 3.97 $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 in the robot controller.

(>>> "Loads on the robot" Page 59)

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 |

| Element | Description |
|---|---|
| *Center of gravity* | Type: FRAME<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the flange<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange |
| *Inertia* | Type: INERTIA<br>■ **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 |

## 3.98 $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 in the robot controller.

(>>> "Loads on the robot" Page 59)

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

> The variable is write-protected and can only be read.

**Syntax**

$LOAD_A3_C={M *Mass*, CM *Center of gravity*, J *Inertia*}

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mass* | Type: REAL; unit: kg |
| *Center of gravity* | Type: FRAME<br>■ **X**, **Y**, **Z**: Position of the center of gravity relative to the flange<br>■ **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange |
| *Inertia* | Type: INERTIA<br>■ **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 |

## 3.99 $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 (variable in the file …R1\Mada\$machine.dat).

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 (variable in the file …R1\Mada\$machine.dat). 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*].

ℹ The variable is write-protected and can only be read.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 ... E6 |
| *Axis value* | Type: REAL; unit: ° (for linear axes: mm)<br>- **-180° … +180°** |

**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

## 3.100 $MEAS_PULSE[]

**Description**

Activation of the inputs for fast measurement

The variable can be used to activate fast measurement via an interrupt. When the interrupt is activated, $MEAS_PULSE[] must have the value FALSE, otherwise an acknowledgement message is generated and the program is stopped.

ℹ Further information about the inputs for fast measurement (interface X33) can be found in the documentation **Optional interfaces** for the KR C4.

**Syntax**

`$MEAS_PULSE[`*Measurement input number*`]=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Measurement input number* | Type: INT<br>- **1 … 8** |
| *State* | Type: BOOL<br>- TRUE: Measurement input is active.<br>- FALSE: Measurement input is not active.<br>Default: FALSE |

## 3.101 $MODE_OP

**Description**

Current operating mode

**Syntax** `$MODE_OP=`*Operating mode*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Operating mode* | Type: ENUM<br>- #AUT: Automatic<br>- #EX: Automatic external<br>- #T1<br>- #T2 |

## 3.102 $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 (variable in the file …\R1\STEU\Mada\$option.dat)

**Syntax** `$MOT_STOP=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: External start is blocked.<br>- FALSE: External start is not blocked.<br>Default: FALSE |

## 3.103 $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.

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

**Explanation of the syntax**

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

## 3.104 $MOUSE_ACT

**Description**

Operating state of the Space Mouse

**Syntax** `$MOUSE_ACT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Space Mouse is active.<br>- FALSE: Space Mouse is not active.<br>Default: FALSE |

## 3.105 $MOUSE_DOM

**Description** Jog mode of the Space Mouse

**Syntax** `$MOUSE_DOM=`*Mode*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Mode* | Type: BOOL<br>- TRUE: Dominant mode is active.<br>Only the coordinate axis with the greatest deflection of the Space Mouse is moved.<br>- FALSE: Dominant mode is not active.<br>Depending on the axis selection, either 3 or 6 axes can be moved simultaneously.<br>Default: TRUE |

## 3.106 $MOUSE_ON

**Description** Activation/deactivation of the Space Mouse on the KUKA smartPAD

**Syntax** `$MOUSE_ON=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Space Mouse is activated.<br>- FALSE: Space Mouse is deactivated.<br>Default: TRUE |

## 3.107 $MOUSE_ROT

**Description** Rotational motions with the Space Mouse On/Off

**Syntax** `$MOUSE_ROT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Rotational motions are possible.<br>- FALSE: Rotational motions are not possible.<br>Default: TRUE |

## 3.108 $MOUSE_TRA

**Description** Translational motions with the Space Mouse On/Off

**Syntax** `$MOUSE_TRA=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: Translational motions are possible.<br>■ FALSE: Translational motions are not possible.<br>Default: TRUE |

## 3.109 $MOVE_BCO

**Description** Indication of whether a BCO run is currently being executed

**Syntax** `$MOVE_BCO=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: BCO run is being carried out.<br>■ FALSE: BCO run is not being carried out. |

## 3.110 $MOVE_STATE

**Description** Current motion state

The variable is used in path planning to identify the individual path sections.

The identifier consists of 2 parts:

- Designation for the current motion type (PTP, LIN or CIRC)
- Designation for the current motion path section

> The variable is write-protected and can only be read.

**Syntax** `$MOVE_STATE=`*Identifier*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Identifier* | Type: ENUM<br>■ **#PTP_AP01, #LIN_AP01, #CIRC_AP01**: Motion in the first approximate positioning range (start to middle)<br>■ **#PTP_AP02, #LIN_AP02, #CIRC_AP02**: Motion in the second approximate positioning range (middle to end)<br>■ **#PTP_SINGLE, #LIN_SINGLE, #CIRC_SINGLE**: Motion outside the approximate positioning range<br>■ **#NONE**: No program is selected, or a program has been reset or deselected<br>Default: **#NONE** |

## 3.111 $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
```

## 3.112 $NUM_IN

**Description**

Number of available digital inputs $IN[]

> The variable is write-protected and can only be read.

**Syntax**

$NUM_IN=*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT |

## 3.113 $NUM_OUT

**Description**

Number of available digital outputs $OUT[]

> The variable is write-protected and can only be read.

**Syntax**

$NUM_OUT=*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT |

## 3.114 $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<br>- #CONSTANT: The orientation of the TCP remains constant during the motion.<br>- #VAR: The orientation of the TCP changes continuously during the motion.<br>- #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.<br>**Note**: If $ORI_TYPE = #JOINT, the variable $CIRC_TYPE is ignored. |

## 3.115 $ORI_TYPE_C

**Description**

Orientation control of a CP motion in the main run ℹ The variable is write-protected and can only be read.

**Syntax** `$ORI_TYPE_C=`*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>■ #CONSTANT: The orientation of the TCP remains constant during the motion.<br>■ #VAR: The orientation of the TCP changes continuously during the motion.<br>■ #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.<br>**Note**: If $ORI_TYPE_C = #JOINT, the variable $CIRC_TYPE_C is ignored. |

## 3.116 $OUT[]

**Description**

Digital output states

The 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 poll the current state of a digital output in the robot program or via the variable correction function.

**Syntax** `$OUT[`*Output number*`]=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>■ **1 … 4 096**: KUKA System Software 8.1<br>■ **1 … 8 192**: KUKA System Software 8.2 and higher<br>**Note**: The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be monitored via $NUM_IN/$NUM_OUT. |
| *State* | Type: BOOL<br>■ TRUE or FALSE |

## 3.117 $OUT_C[]

**Description**

Setting of digital outputs in the main run

The 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*`]=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>- **1 … 4 096**: KUKA System Software 8.1<br>- **1 … 8 192**: KUKA System Software 8.2 and higher<br>**Note**: The range of values contains the maximum possible number of digital I/Os. The number of digital I/Os actually available can be monitored via $NUM_IN/$NUM_OUT. |
| *State* | Type: BOOL<br>- TRUE or FALSE |

## 3.118 $OV_PRO

**Description**

Program override is the velocity of the robot during program execution. The program override is specified as a percentage of the programmed velocity.

In T1 mode, the maximum velocity is 250 mm/s, irrespective of the value that is set.

**Syntax**

$0V_PR0=*Override*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Override* | Type: INT; unit: %<br>- **0 … 100**<br>Default: **100** |

## 3.119 $OV_ROB

**Description**

Robot override

The robot override is the current velocity of the robot in program execution. The variable is write-protected.

The robot override is determined by the program override $OV_PRO as a function of the reduction factor for the program override $RED_VEL. If $RED_VEL=100, i.e. the program override is not reduced, $OV_ROB is identical to $OV_PRO.

**Syntax**

$0V_R0B=*Override*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Override* | Type: INT; unit: %<br>- **0 … 100** |

## 3.120 $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 the palletizing mode is contained in the Operating and Programming Instructions for System Integrators.

**Syntax**

`$PAL_MODE=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: Palletizing mode is active.<br>■ FALSE: Palletizing mode is not active.<br>Default: Dependent on the robot model |

## 3.121 $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<br>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 *n* of time components |
| 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 |

## 3.122 $POS_ACT

**Description**

Current Cartesian robot position

The variable of structure type E6POS defines the setpoint position of the TCP in relation 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. In the robot program, the variable triggers an advance run stop.

## 3.123 $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 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 [°]

## 3.124 $POS_BACK

**Description**

Cartesian start position of the current motion block

The variable of structure type E6POS defines the start position of the TCP in relation 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 [°]

$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.

**Example**

Approximated PTP motion

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

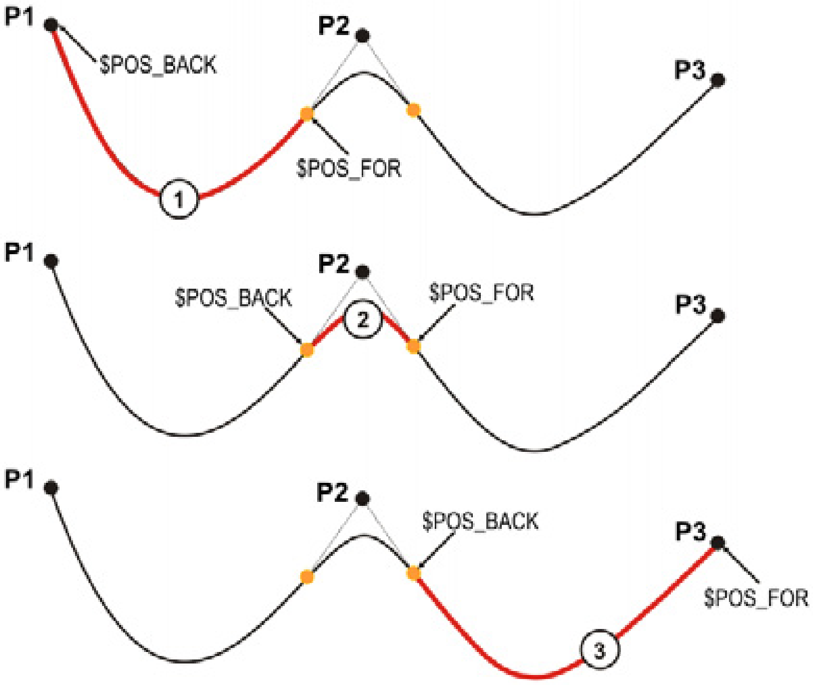


**Fig. 3-5: $POS_BACK, $POS_FOR – P2 is approximated**

| | | | |
|---|---|---|---|
| 1 | Single block | 3 | Following block |
| 2 | Intermediate block | | |

## 3.125 $POS_FOR

**Description**

Cartesian target position of the current motion block

The variable of structure type E6POS defines the target position of the TCP in relation 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 [°]

$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.

**Example**

(>>> 3.124 "$POS_BACK" Page 73)

## 3.126 $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.

## 3.127 $POS_RET

**Description**

Cartesian robot position when leaving the path

The variable of structure type E6POS defines the position of the TCP in relation to the BASE coordinate system at the time that the programmed path was left.

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

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

## 3.128 $POWER_FAIL

**Description**

Display of power failure

**Syntax**

`$POWER_FAIL=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: Power failure<br>■ FALSE: No power failure |

## 3.129 $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<br>- **1 … 30,000**<br>- **0**: The robot controller is shut down despite external power supply.<br>Default: **3** |

## 3.130 $PRO_I_O[] – KUKA System Software 8.3 and higher

**Description**

Path to the submit program

The variable represents the current configuration of $PRO_I_O_SYS.MODULE[] (variable in the file …STEU\Mada\$custom.dat). It contains the path to the submit program that is to be automatically started at a cold start of the robot controller. The program SPS.SUB starts in the system submit interpreter by default.

The variable is write-protected.

## 3.131 $PRO_IP

**Description**

Structure with the data of the process pointer with reference to an interpreter

The variable contains the data of the block that will be executed next in an interpreter.

Depending on the specific interpreter, access to the data is as follows:

- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.
- Reading/writing to the variable by means of the variable correction 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 (default operating mode):

  - 0: Submit interpreter
  - 1: Robot interpreter

  Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

  - 1: Robot interpreter
  - 2: System submit interpreter
  - 3: Extended submit interpreter 1
  - 4: Extended submit interpreter 2
  - …
  - 9: Extended submit interpreter 7

**Access to P_Arrived in a submit program:**

$PRO_IP contains the following initialized components in the submit interpreter:

- $PRO_IP.SNR
- $PRO_IP.Name[]
- $PRO_IP.I_Executed

The component $PRO_IP.P_Arrived is not initialized in a submit interpreter. Reading the component P_Arrived in a submit program triggers the error message *{$variable} value invalid*.

In order to be able to read the robot interpreter component P_Arrived in a submit program, the variable $PRO_IP1 must be used:

```
IF ($PRO_IP1.P_Arrived == 1) THEN …
```

**Syntax**

`$PRO_IP=`*Process data*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Process data* | Type: Pro_Ip<br>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 S101, S102, …S110
```

| Element | Description |
|---|---|
| name[] | Name of the module in which the interpreter is in the advance run |
| snr | 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[] | Name of the module in which the interpolator is in the main run |
| snr_c | Number of the block in which the interpolator is in the main run |
| i_executed | Indicates whether the block has already been executed by the interpreter (= TRUE) |
| p_arrived | Indicates point on the path where the robot is located (only relevant for motion instructions)<br>■ **0**: Arrived at the target or auxiliary point of the motion<br>■ **1**: Target point not reached (robot is somewhere on the path)<br>■ **2**: Not relevant<br>■ **3**: Arrived at the auxiliary point of a CIRC or SCIRC motion<br>■ **4**: On the move in the section between the start and the auxiliary point |
| p_name[] | Name or aggregate of the target or auxiliary point at which the robot is located |
| S101 … S110 | Caller stack in which the interpreter is situated |

## 3.132 $PRO_IP0

**Description**

Structure with the data of the process pointer in the submit interpreter

The variable contains the data of the block that will be executed next by the submit interpreter. The variable can be read by means of both a robot program and a submit program. Data can also be written to it using the variable correction function.

> If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), the variable is irrelevant. The system states of a robot controller in Multi-Submit mode are grouped in the variable $PROG_INFO[].

**Syntax**

`$PRO_IP0=`*Process data*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Process data* | Type: Pro_Ip<br>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 S101, S102, …S110
```

| Element | Description |
|---|---|
| name[] | Name of the module in which the interpreter is in the advance run |
| snr | 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[] | Name of the module in which the interpolator is in the main run |
| snr_c | Number of the block in which the interpolator is in the main run |
| i_executed | Indicates whether the block has already been executed by the interpreter (= TRUE) |
| p_arrived | Indicates point on the path where the robot is located (only relevant for motion instructions)<br>■ **0**: Arrived at the target or auxiliary point of the motion<br>■ **1**: Target point not reached (robot is somewhere on the path)<br>■ **2**: Not relevant<br>■ **3**: Arrived at the auxiliary point of a CIRC or SCIRC motion<br>■ **4**: On the move in the section between the start and the auxiliary point |
| p_name[] | Name or aggregate of the target or auxiliary point at which the robot is located |
| S101 …<br>S110 | Caller stack in which the interpreter is situated |

## 3.133 $PRO_IP1

**Description**

Structure with the data of the process pointer in the robot interpreter

The variable contains the data of the block that will be executed next by 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 using the variable correction function.

**Syntax**

`$PRO_IP1=`*Process data*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Process data* | Type: Pro_Ip<br>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 S101, S102, …S110
```

| Element | Description |
|---|---|
| name[] | Name of the module in which the interpreter is in the advance run |
| snr | 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[] | Name of the module in which the interpolator is in the main run |
| snr_c | Number of the block in which the interpolator is in the main run |
| i_executed | Indicates whether the block has already been executed by the interpreter (= TRUE) |
| p_arrived | Indicates point on the path where the robot is located (only relevant for motion instructions)<br>- **0**: Arrived at the target or auxiliary point of the motion<br>- **1**: Target point not reached (robot is somewhere on the path)<br>- **2**: Not relevant<br>- **3**: Arrived at the auxiliary point of a CIRC or SCIRC motion<br>- **4**: On the move in the section between the start and the auxiliary point |
| p_name[] | Name or aggregate of the target or auxiliary point at which the robot is located |
| S101 …<br>S110 | Caller stack in which the interpreter is situated |

## 3.134 $PRO_MODE

**Description**

Program run mode with reference to an interpreter

Depending on the specific interpreter, access to the information is as follows:

- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.
- Reading/writing to the variable by means of the variable correction 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 (default operating mode):

  - 0: Submit interpreter

- 1: Robot interpreter

Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

**Syntax**

`$PRO_MODE=`*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>■ #GO: The program is executed through to the end without stopping.<br>■ #MSTEP: Motion Step<br>The program is executed with a stop at each point, including auxiliary points and the points of a spline segment. The program is executed without advance processing.<br>■ #ISTEP: Incremental Step<br>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.<br>■ #BSTEP: Backward Step<br>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.<br>■ #PSTEP: Program Step<br>The program is executed step by step without advance processing. Subprograms are executed completely.<br>■ #CSTEP: Continuous Step<br>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.<br>Default: #GO |

## 3.135 $PRO_MODE0

**Description**

Program run mode in the submit interpreter

The variable can be read by means of both a robot program and a submit program. Data can also be written to it using the variable correction function.

> ℹ If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), the variable is irrelevant. The system states of a robot controller in Multi-Submit mode are grouped in the variable $PROG_INFO[].

**Syntax**

`$PRO_MODE0=`*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>- #GO: The program is executed through to the end without stopping.<br>- #ISTEP: Incremental Step<br>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.<br>- #PSTEP: Program Step<br>The program is executed step by step without advance processing. Subprograms are executed completely.<br>Default: #GO |

ℹ The program run modes #MSTEP, #BSTEP and #CSTEP are not available in a submit interpreter.

## 3.136 $PRO_MODE1

**Description**

Program run mode in 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 using the variable correction function.

**Syntax**

$PRO_MODE1=*Type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Type* | Type: ENUM<br>- #GO: The program is executed through to the end without stopping.<br>- #MSTEP: Motion Step<br>The program is executed with a stop at each point, including auxiliary points and the points of a spline segment. The program is executed without advance processing.<br>- #ISTEP: Incremental Step<br>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.<br>- #BSTEP: Backward Step<br>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.<br>- #PSTEP: Program Step<br>The program is executed step by step without advance processing. Subprograms are executed completely.<br>- #CSTEP: Continuous Step<br>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.<br>Default: #GO |

## 3.137 $PRO_NAME[]

**Description**

Name of the selected program with reference to an interpreter

Depending on the specific interpreter, access to the information is as follows:

- The variable is write-protected.
- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.
- Reading the variable by means of the variable correction 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 (default operating mode):

  - 0: Submit interpreter
  - 1: Robot interpreter

  Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

  - 1: Robot interpreter
  - 2: System submit interpreter
  - 3: Extended submit interpreter 1
  - 4: Extended submit interpreter 2

- …
- 9: Extended submit interpreter 7

**Syntax**

`$PRO_NAME[]=`*Name*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Program name: max. 24 characters |

## 3.138 $PRO_NAME0[]

**Description**

Name of the selected program in the submit interpreter

> The variable is write-protected and can only be read.

> If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), the variable is irrelevant. The system states of a robot controller in Multi-Submit mode are grouped in the variable $PROG_INFO[].

**Syntax**

`$PRO_NAME1[]=`*Name*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Program name: max. 24 characters |

## 3.139 $PRO_NAME1[]

**Description**

Name of the selected program in the robot interpreter

> The variable is write-protected and can only be read.

**Syntax**

`$PRO_NAME1[]=`*Name*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Program name: max. 24 characters |

## 3.140 $PRO_STATE

**Description**

Program state with reference to an interpreter

Depending on the specific interpreter, access to the information is as follows:

- The variable is write-protected.
- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.

- Reading the variable by means of the variable correction 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 (default operating mode):

  - 0: Submit interpreter
  - 1: Robot interpreter

  Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

  - 1: Robot interpreter
  - 2: System submit interpreter
  - 3: Extended submit interpreter 1
  - 4: Extended submit interpreter 2
  - …
  - 9: Extended submit interpreter 7

**Syntax**

`$PRO_STATE=`*State*

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>An interpreter can have the following states:<br>- #P_ACTIVE: Program is selected and is running.<br>- #P_FREE: Program is deselected.<br>- #P_END: Program is selected and the end of the program has been reached.<br>- #P_RESET: Program is selected and has been stopped and reset.<br>- #P_STOP: Program is selected and has been stopped. |

## 3.141 $PRO_STATE0

**Description**

Program state in the submit interpreter

The status bar of the KUKA smartHMI contains a display of the state of the submit interpreter. $PRO_STATE0 reflects this state. The variable is write-protected.

If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), the status bar contains a group indicator of the state of all submit interpreters. $PRO_STATE0 then reflects the state of the group indicator.

**Syntax**

`$PRO_STATE0=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>In Single Submit mode, a submit interpreter can have the following states:<br>■ #P_ACTIVE: Submit program is selected and is running.<br>■ #P_FREE: Submit program is deselected.<br>■ #P_END: Submit program is selected and the end of the program has been reached.<br>■ #P_RESET: Submit program is selected and has been stopped and reset.<br>■ #P_STOP: Submit program is selected and has been stopped.<br>In Multi-Submit mode, a submit interpreter can have the following states:<br>■ #P_ACTIVE: At least 1 submit program is selected and is running.<br>■ #P_FREE: All submit programs are deselected.<br>■ #P_RESET: At least 1 submit program is selected and has been stopped and reset.<br>No submit interpreter is in the #P_ACTIVE or #P_STOP state.<br>■ #P_STOP: At least 1 submit program is selected and has been stopped.<br>No submit interpreter is in the #P_ACTIVE state. |

## 3.142 $PRO_STATE1

**Description** Program state in the robot interpreter

The variable is write-protected and can only be read.

**Syntax** `$PRO_STATE1=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>A robot interpreter can have the following states:<br>■ #P_ACTIVE: Robot program is selected and is running.<br>■ #P_FREE: Robot program is deselected.<br>■ #P_END: Robot program is selected and the end of the program has been reached.<br>■ #P_RESET: Robot program is selected and has been stopped and reset.<br>■ #P_STOP: Robot program is selected and has been stopped. |

## 3.143 $PROG_INFO[]

**Description** System states of a robot controller in Multi-Submit mode

$PROG_INFO[] groups certain system states of a robot controller operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher) together in a structure. The states refer to an interpreter.

**Syntax**

`$PROG_INFO[`*Interpreter*`] = `*Information*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Interpreter* | Type: INT<br>■ 1: Robot interpreter<br>■ 2: System submit interpreter<br>■ 3: Extended submit interpreter 1<br>■ 4: Extended submit interpreter 2<br>■ …<br>■ 9: Extended submit interpreter 7 |
| *Information* | Type: Prog_Info<br>List of the system states, referred to *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[] | Name of the selected program |
| p_state | Program state<br>The possible values are the same as for $PRO_STATE. |
| p_mode | Program run mode<br>The possible values are the same as for $PRO_MODE. |
| pro_ip_name[] | Name of the current module |
| pro_ip_snr | Current block in the current module |

**Example**

```
DEF myProgr()
...
WAIT FOR $PROG_INFO[4].P_STATE == #P_ACTIVE
```

Meaning: Wait until extended submit 2 has selected and started a program.

## 3.144 $RCV_INFO[]

**Description**

Version identifier of the kernel system

**Syntax**

`$RCV_INFO[]="`*Identifier*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Identifier* | Type: CHAR<br>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.

## 3.145 $RED_VEL

**Description** Reduction factor for program override in the advance run

**Syntax** `$RED_VEL=`*Reduction factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reduction factor* | Type: INT; unit: %<br>■ **1 … 100**<br>Default: **100** |

## 3.146 $RED_VEL_C

**Description** Reduction factor for program override in the main run

> The variable is write-protected and can only be read.

**Syntax** `$RED_VEL_C=`*Reduction factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reduction factor* | Type: INT; unit: %<br>■ **1 … 100**<br>Default: **100** |

## 3.147 $REVO_NUM[]

**Description** Counter for infinitely rotating axes

**Syntax** `$REVO_NUM[`*Axis number*`]=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Number* | Type: INT<br>Number of revolutions |

## 3.148 $RINT_LIST[]

**Description** Structure with the data for a robot interrupt

These data can be displayed via the variable correction function or by means of the diagnosis function in the main menu.

**Precondition**

- “Expert” user group

**Procedure**

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

In robot and submit programs, a maximum of 32 interrupts can be declared simultaneously and up to 16 interrupts can be active at the same time.

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

**Syntax**

`$RINT_LIST[`*Index*`]={INT_PRIO` *Priority*`,INT_STATE` *State*`,INT_TYPE` *Type*`,PROG_LINE` *Line*`,PROG_NAME[]` "*Name*"`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Index of the interrupt<br>■ **1 … 32** |
| `INT_PRIO` | Type: INT<br>Priority of the interrupt<br>■ **1, 2, 4 … 39**<br>■ **81 … 128** |
| `INT_STATE` | Bit array for interrupt states<br>■ **Bit 0 = 1**: Interrupt is declared and activated.<br>■ **Bit 1 = 1**: Interrupt is activated and enabled.<br>■ **Bit 2 = 1**: Interrupt is globally declared. |
| `INT_TYPE` | Type: INT<br>Type of interrupt<br>■ **0**: Standard interrupt<br>■ **1**: Interrupt due to an EMERGENCY STOP ($EMSTOP)<br>■ **2**: Interrupt due to activation of the Fast Measurement inputs ($MEAS_PULSE)<br>■ **3**: Interrupt due to an error stop ($STOPMESS)<br>■ **4**: Interrupt due to a trigger (subprogram call) |
| `PROG_LINE` | Type: INT<br>Line number of the robot program in which the interrupt is declared |
| `PROG_NAME` | Type: CHAR<br>Directory and name of the robot program in which the interrupt is declared: max. 32 characters |

## 3.149 $ROB_TIMER

**Description**

Clock generator for measuring program runtimes

The variable is write-protected and counts in a cycle of 1 ms.

$ROB_TIMER can only be assigned to an integer variable or compared with an integer variable. If a real variable is used, this results in values that are too high by a factor of around 100.

**Syntax**

`$ROB_TIMER=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT<br>Number of cycles |

## 3.150 $ROBNAME[]

**Description**

User-defined robot name

The robot name entered in the robot data by the user is written to this variable.

**Syntax**

`$ROBNAME[]="`*Name*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Robot name: max. 50 characters |

## 3.151 $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 [°]

> The variable is write-protected and can only be read.

## 3.152 $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.

> Information about the individual machine data can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

`$ROBROOT_KIN[]="`*Information*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Information* | Type: CHAR<br>Name and external axes of the transformation: max. 29 characters |

## 3.153 $ROBRUNTIME

**Description**

Operating hours meter

The operating hours meter is running as long as the drives are switched on.

**Syntax** `$ROBRUNTIME=`*Operating hours*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Operating hours* | Type: INT; unit: min |

## 3.154 $ROBTRAFO[]

**Description** Robot name

The variable contains the robot name programmed 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<br>Robot name: max. 32 characters |

## 3.155 $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 relative motions and in jogging.

**Syntax** `$ROTSYS=`*Reference system*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reference system* | Type: ENUM<br>■ #AS_TRA: Rotation in the coordinate system $TRANSSYS<br>■ #BASE: Rotation in the BASE coordinate system<br>■ #TCP: Rotation in the TOOL coordinate system<br>Default: #AS_TRA |

## 3.156 $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 in relative motions and in jogging.

> ℹ The variable is write-protected and can only be read.

**Syntax** `$ROTSYS_C=`*Reference system*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reference system* | Type: ENUM<br>■ #AS_TRA: Rotation in the coordinate system $TRANS-SYS<br>■ #BASE: Rotation in the BASE coordinate system<br>■ #TCP: Rotation in the TOOL coordinate system<br>Default: #AS_TRA |

## 3.157 $RUNTIME_DATA0

> This system variable is only available for reasons of compatibility. It is advisable to poll the runtime data via the variable $ERR.

**Description**

Structure with the runtime data of the submit interpreter

The variable can be used to display the runtime data via the variable correction function. The variable is write-protected.

**Syntax**

`$RUNTIME_DATA0={VISIBLE` *Visibility*`, NAME[] "`*Module*`", SNR` *Block number*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Visibility* | Type: BOOL<br>Visibility of the program in editor<br>■ TRUE: Program is visible.<br>■ FALSE: Program is not visible. |
| *Module* | Type: CHAR<br>Name of the module in which the interpreter is situated: max. 32 characters |
| *Block number* | Type: INT<br>Block number in which the interpreter is situated. |

## 3.158 $RUNTIME_DATA1

> This system variable is only available for reasons of compatibility. It is advisable to poll the runtime data via the variable $ERR.

**Description**

Structure with the runtime data of the robot interpreter

The variable can be used to display the runtime data via the variable correction function. The variable is write-protected.

**Syntax**

`$RUNTIME_DATA1={VISIBLE` *Visibility*`, NAME[] "`*Module*`", SNR` *Block number*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Visibility* | Type: BOOL<br>Visibility of the program in editor<br>- TRUE: Program is visible.<br>- FALSE: Program is not visible. |
| *Module* | Type: CHAR<br>Name of the module in which the interpreter is situated: max. 32 characters |
| *Block number* | Type: INT<br>Block number in which the interpreter is situated. |

## 3.159 $RUNTIME_ERROR0

This system variable is only available for reasons of compatibility. It is advisable to poll the runtime data via the variable $ERR.

**Description** Runtime error of the submit interpreter

**Syntax** `$RUNTIME_ERROR0=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: A runtime error has occurred.<br>- FALSE: No runtime error has occurred.<br>Default: FALSE |

## 3.160 $RUNTIME_ERROR1

This system variable is only available for reasons of compatibility. It is advisable to poll the runtime data via the variable $ERR.

**Description** Runtime error of the robot interpreter

**Syntax** `$RUNTIME_ERROR1=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: A runtime error has occurred.<br>- FALSE: No runtime error has occurred.<br>Default: FALSE |

## 3.161 $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<br>■ TRUE: Resonance avoidance is active.<br>■ FALSE: Resonance avoidance is not active.<br>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.

## 3.162 $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<br>■ TRUE: Drives are enabled.<br>■ FALSE: Drives are not enabled.<br>Default: FALSE |

## 3.163 $SAFETY_SW

**Description**

State of the enabling switches

**Syntax**

$SAFETY_SW=*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>- #PRESSED: An enabling switch is pressed (center position).<br>- #RELEASED: No enabling switch is pressed or an enabling switch is fully pressed (panic position). |

## 3.164 $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<br>- TRUE: The safety controller is not operable.<br>- FALSE: The safety controller is running without errors.<br>Default: FALSE |

## 3.165 $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<br>- TRUE: Start-up mode is active.<br>- FALSE: Start-up mode is not active.<br>Default: FALSE |

## 3.166 $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. The variable is write-protected.

**Syntax**

`$SAFE_IBN_ALLOWED=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Start-up mode is allowed.<br>- FALSE: Start-up mode is not allowed.<br>Default: FALSE |

## 3.167 $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<br>Index of the variable<br>- **1 … 20** |
| *Value* | Type: INT |

## 3.168 $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<br>Index of the variable<br>- **1 … 20** |
| *Value* | Type: INT |

## 3.169 $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<br>Index of the variable<br>■ **1 … 20** |
| *Value* | Type: REAL |

## 3.170 $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<br>Index of the variable<br>■ **1 … 20** |
| *Value* | Type: REAL |

## 3.171 $SERVO_SIM

**Description**

Simulation of the axis motions

**Syntax**

`$SERVO_SIM=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ TRUE: Simulation is active.<br>■ FALSE: No simulation active.<br>Default: FALSE |

## 3.172 $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.

**Description**

Number of digital inputs/outputs available

- KUKA System Software 8.1: up to 4,096 digital I/Os
- KUKA System Software 8.2 and higher: up to 8,192 digital I/Os

> The variable is write-protected and can only be read.

**Syntax** $SET_IO_SIZE=*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT<br>■ **1**: 1 … 1 024<br>■ **2**: 1 … 2 048<br>■ **4**: 1 … 4 096<br>■ **8**: 1 … 8 192 (KSS 8.2 and higher)<br>Default: **4** |

## 3.173 $SINGUL_DIST[]

**Description**

Standardized distance from the singularity

The variable specifies the standardized distance of the current robot position from the singularity in question. The variable is write-protected.

If the standardized distance at a robot position ≤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 (variable in the file …R1\Mada\$machine.dat).

**Syntax** $SINGUL_DIST[*Index*]=*Distance*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Type of singularity position:<br>■ **1**: Overhead singularity $SINGUL_POS[1]<br>■ **2**: Extended position singularity $SINGUL_POS[2]<br>■ **3**: Wrist axis singularity $SINGUL_POS[3] |
| *Distance* | Type: INT<br>Standardized distance from the specified singularity |

## 3.174 $SINT_LIST[]

**Description**

Structure with the data for a submit interpreter

These data can be displayed via the variable correction function or by means of the diagnosis function in the main menu.

**Precondition**

- “Expert” user group

**Procedure**

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

> In robot and submit programs, a maximum of 32 interrupts can be declared simultaneously and up to 16 interrupts can be active at the same time.

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

**Syntax** $SINT_LIST[*Index*]={INT_PRIO *Priority*,INT_STATE *State*,INT_TYPE *Type*,PROG_LINE *Line*,PROG_NAME[] "*Name*"}

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Index of the interrupt<br>■ **1 … 32** |
| `INT_PRIO` | Type: INT<br>Priority of the interrupt<br>■ **1, 2, 4 … 39**<br>■ **81 … 128** |
| `INT_STATE` | Bit array for interrupt states<br>■ **Bit 0 = 1**: Interrupt is declared and activated.<br>■ **Bit 1 = 1**: Interrupt is activated and enabled.<br>■ **Bit 2 = 1**: Interrupt is globally declared. |
| `INT_TYPE` | Type: INT<br>Type of interrupt<br>■ **0**: Standard interrupt<br>■ **1**: Interrupt due to an EMERGENCY STOP ($EMSTOP)<br>■ **2**: Interrupt due to activation of the Fast Measurement inputs ($MEAS_PULSE)<br>■ **3**: Interrupt due to an error stop ($STOPMESS)<br>■ **4**: Interrupt due to a trigger (subprogram call) |
| `PROG_LINE` | Type: INT<br>Line number of the submit program in which the interrupt is declared |
| `PROG_NAME` | Type: CHAR<br>Directory and name of the submit program in which the interrupt is declared: max. 32 characters |

## 3.175 $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<br>Index of the variable<br>■ **1 … 1024** |
| *Value* | Type: BOOL |

## 3.176 $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<br>Index of the variable<br>- **1 … 1024** |
| *Value* | Type: INT |

## 3.177 $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<br>Index of the variable<br>- **1 … 1024** |
| *Value* | Type: REAL |

## 3.178 $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<br>Index of the variable<br>- **1 … 8**: ProConOS events 0 … 7 (= KRC event bits 0 … 7) |
| *State* | Type: BOOL<br>- TRUE: Event has been called.<br>- FALSE: No event has been called. |

## 3.179 $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 correction 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<br>Number of the function generator:<br>- **1 … 6** |
| *Parameters* | Type: Spl_Tech<br>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 mo-
de, TECHCLASS class, SPL_TECHSYS ref_sys, SPL_FCTCTRL fct-
ctrl, 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.<br>- TRUE: Wrist axis weaving is activated.<br>- FALSE: Wrist axis weaving is deactivated.<br>Default: FALSE |

| Element | Description |
|---|---|
| lim_full_hand_weaving | Limitation of the deviation from the programmed correction direction during wrist axis weaving. Only positive angles can be programmed. (unit: °)<br>■ If the deviation is less than the angle `lim_full_hand_weaving`, only the wrist axis weaving correction is calculated.<br>■ 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.<br>■ 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<br>■ #OFF: No function evaluation<br>■ #SINGLE: The function is evaluated once.<br>■ #CYCLE: The function is evaluated cyclically. |
| class | Technology class – input variable for the spline function generator<br>■ #PATH: The input variable is the arc length of a spline motion $DISTANCE (unit: mm)<br>■ #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<br>`STRUC Spl_Techsys TECHSYS sys, TECHANGLE angles, TECHGEOREF georef`<br>■ Reference coordinate system selection (TECHSYS sys)<br>▪ #WORLD<br>▪ #BASE<br>▪ #ROBROOT<br>▪ #TCP<br>▪ #TTS<br>■ Rotation of the reference coordinate system (TECHANGLE Angles; type: REAL; unit: °)<br>▪ A, B, C: Angles about which the Z, Y and X axes of the reference coordinate system are rotated<br>■ Reference coordinate system axis used for weaving (TECHGEOREF Georef)<br>▪ #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[].<br>▪ #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.<br>▪ #A, #B, #C: These components are not permissible. |

| Element | Description |
|---|---|
| fctctrl | Control structure for the weave parameters of the spline function generator<br>`STRUC Spl_Fctctrl BOOL adjust_wavelength, REAL scale_in, scale_out, offset_out`<br>■ BOOL adjust_wavelength: The wavelength `scale_in` can be adapted to an integer multiple of the remaining path for a single spline block (= TRUE)<br>■ REAL scale_in: wavelength of the weave pattern<br>■ REAL scale_out: amplitude of the weave pattern<br>■ REAL offset_out: position of the weave pattern, e.g.:<br>▪ $SPL_TECH[1].FCTCTRL.OFFSET_OUT = 0.0: The zero point of the weave motion is on the spline path. |
| fct | Structure for defining the weave pattern for the spline function generator<br>`STRUC Spl_TechFct SPL_TECHFCT_MODE mode, TECHFCT Polynomial, SPL_FCTDEF fct_def, REAL blend_in, blend_out, phase_shift`<br>■ ENUM for specification of the weave pattern type (SPL_TECHFCT_MODE mode)<br>▪ #FCT_POLYNOMIAL: polynomial with control points<br>▪ #FCT_SIN: asymmetrical sine (both half-waves can have different amplitudes and wavelengths)<br>■ Structure for defining the weave parameters for the weave pattern type #FCT_POLYNOMIAL (TECHFCT Polynomial)<br>(>>> "TechFct" Page 102)<br>■ Structure for defining the weave parameters for the weave pattern type #FCT_SIN (SPL_FCTDEF fct_def)<br>(>>> "Spl_FctDef" Page 102)<br>■ 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.<br>■ 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.<br>■ REAL phase_shift: phase offset for execution of the weave pattern, e.g.:<br>▪ $SPL_TECH[1].FCT.PHASE_SHIFT = 0.0: Weaving commences at the start of the spline path.<br>**Note**: The definition range and range of values of the function `fct` are defined as follows:<br>■ Definition range: **0 … 1**<br>■ Range of values: **-1 … +1** |

| Element | Description |
|---|---|
| fade_in | Smoothing length for start of weaving and end of weaving<br>- **0.0 … 1.0**<br>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.<br>The length of these intervals is defined using `fade_in` and `fade_out`. The longer the interval, the smoother the motion. |
| fade_out | |

**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<br>- **1**: Weave pattern is defined by a polygon (sufficient in the case of thermal weaving alone)<br>- **3**: Weave pattern is defined by a cubic spline (required in the case of mechanical weaving)<br>**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<br>- **2 … 40**<br>**Note**: No gaps are allowed between the valid control points. |
| cps1 | List with control points 1 … 10 (type: REAL)<br>- **X1, Y1, … X10, Y10** |
| cps2 | List with control points 11 … 20 (type: REAL)<br>- **X1, Y1, … X10, Y10** |
| cps3 | List with control points 21 … 30 (type: REAL)<br>- **X1, Y1, … X10, Y10** |
| cps4 | List with control points 31 … 40 (type: REAL)<br>- **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)<br>- #CYCLIC: Cyclical boundary conditions (required in the case of mechanical weaving)<br>- #NATURAL: Natural boundary conditions (sufficient in the case of thermal weaving alone) |

**Spl_FctDef**

```
STRUC Spl_FctDef REAL amplitude1, amplitude2,
wavelength_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) |
| wavelength _ratio | Ratio of the wavelengths of the 2nd half-wave to the wavelength of the 1st half-wave |

## 3.180 $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 correction 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.
(>>> 3.181 "$SPL_TECH_LINK" Page 104)

**Syntax**

`$SPL_TECH_C[`*Index*`]=`*Parameters*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the function generator:<br>- **1 … 6** |
| *Parameters* | Type: Spl_Tech<br>Definition of the function parameters and programming of the function evaluation |

**Spl_Tech**

Declaration of the structure and description of the structure elements:
(>>> "Spl_Tech" Page 99)

**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
```

## 3.181 $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.

## 3.182 $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**

Declaration of the structure and description of the linking:
(>>> "Spl_Tech_Map" Page 104)

## 3.183 $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. The variable is write-protected.

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.

**Syntax**

`$SPL_TSYS[`*Index*`]=`*Position*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the function generator:<br>■ **1 … 6** |
| *Position* | Type: FRAME<br>■ **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]<br>■ **A**, **B**, **C**: Rotational offset of the axis angles in [°] |

## 3.184 $SPL_VEL_MODE – KUKA System Software 8.2 and higher

**Description**

Motion profile for spline motions

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

**Syntax**

`$SPL_VEL_MODE=`*Motion profile*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Motion profile* | Type: ENUM<br>■ #OPT: Higher motion profile<br>This motion profile is activated by default when System Software 8.2 or higher is installed for the first time.<br>■ #CART: Conventional motion profile<br>This motion profile is activated by default if System Software 8.1 had previously been installed before the installation of System Software 8.2 or higher. |

## 3.185 $SPL_VEL_RESTR – KUKA System Software 8.2 and higher

**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<br>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}
```

## 3.186 $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 |

## 3.187 $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

**Syntax**

`$STOPNOAPROX=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE<br>  - Operating mode T1 or T2: Acknowledgement message<br>  - Operating mode AUT or AUT EXT: Notification message<br>- FALSE: Notification message only<br>Default: FALSE |

## 3.188 $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<br>Number of the function generator<br>- **1 … 6** |
| *Parameter* | Type: Tech<br>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<br>- #OFF: No function evaluation<br>- #SINGLE: The function is evaluated once.<br>- #CYCLE: The function is evaluated cyclically. |
| class | Technology class – input variable for the function generator<br>- #PATH: The input variable is the arc length of a CP motion $DISTANCE (unit: mm)<br>- #SYSTIME: The input variable is the system time (unit: ms)<br>- #VEL: The input variable is the current path velocity $VEL_ACT (unit: m/s)<br>- #SENSOR: The input variable is the variable $TECHIN[]. Depending on the input values, the robot performs a position correction.<br>- #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. |

| Element | Description |
|---|---|
| fctctrl | Control structure for the parameters of the function generator<br>`STRUC Fctctrl REAL scale_in, scale_out, offset_in, offset_out, TECHGEOREF georef`<br>■ REAL scale_in: scales the definition range of the function<br>■ REAL scale_out: scales the range of values of the function<br>■ REAL offset_in: offsets the zero point of the definition range of the function<br>■ REAL offset_out: offsets the zero point of the range of values of the function<br>■ TECHGEOREF georef: ENUM for the geometric reference of the technology function (>>> "GeoRef" Page 108)<br>**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<br>(>>> "TechFct" Page 108)<br>**Note**: The definition range and range of values of the function `fct` are defined as follows:<br>■ Definition range: **0 … 1**<br>■ 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[].<br>**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<br>■ 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<br>Axis angle of the reference coordinate system (programmed by means of $TECHSYS and $TECHANGLE) used for sensor correction<br>■ 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<br>■ **1**: Function is defined by a polygon |
| cpnum | Total number of valid control points with reference to the following 5 control point structures<br>■ **2 … 50**<br>**Note**: No gaps are allowed between the valid control points. |
| cps1 | List with control points 1 … 10 (type: REAL)<br>■ **X1, Y1, … X10, Y10** |
| cps2 | List with control points 11 … 20 (type: REAL)<br>■ **X1, Y1, … X10, Y10** |
| cps3 | List with control points 21 … 30 (type: REAL)<br>■ **X1, Y1, … X10, Y10** |
| cps4 | List with control points 31 … 40 (type: REAL)<br>■ **X1, Y1, … X10, Y10** |
| cps5 | List with control points 41 … 50 (type: REAL)<br>■ **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).

```
SIGNAL Correction $ANIN[2]

INTERRUPT DECL 1 WHEN $TECHVAL[1] > 20.0 DO Upper_Limit()
INTERRUPT DECL 2 WHEN $TECHVAL[1] < -20.0 DO Lower_Limit()

ANIN ON $TECHIN[1] = Factor * Correction + Offset

$TECHSYS = #TTS
$TECH[1].FTCCTRL.GEOREF = #Z

$TECH.CLASS = #SENSOR

$TECH[1].FCTCTRL.SCALE_IN = 2.0
$TECH[1].FCTCTRL.OFFSET_IN = 0.0
$TECH[1].FCTCTRL.SCALE_OUT = 2.0
$TECH[1].FCTCTRL.OFFSET_OUT = 0.0
$TECH[1].FCT.ORDER = 1
$TECH[1].FCT.CPNUM = 3
$TECH[1].FCT.CPS1.X1 = 0.0
$TECH[1].FCT.CPS1.Y1 = -1.0
$TECH[1].FCT.CPS1.X2 = 0.5
$TECH[1].FCT.CPS1.Y2 = 0.0
$TECH[1].FCT.CPS1.Y3 = 3.0
$TECHPAR[1,1] = 0.056

PTP Go_to_Workpiece
INTERRUPT ON 1
INTERRUPT ON 2

TECH[1].MODE = #CYCLE
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. |

## 3.189 $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 correction 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<br>Number of the function generator<br>- **1 … 6** |
| *Parameter* | Type: Tech<br>Definition of the function parameters and programming of the function evaluation |

**Tech**

Declaration of the structure TECH and description of the structure elements: (>>> "Tech" Page 107)

## 3.190 $TECHANGLE

**Description**

Rotation of the reference coordinate system of a function generator in the advance run

The 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 |
|---|---|
| A | Type: REAL; unit: °<br>Angle about which the Z, Y or X axis of the reference coordinate system is rotated (only positive direction permissible) |
| B | |
| C | |

## 3.191 $TECHANGLE_C

**Description**

Rotation of the reference coordinate system of a function generator in the main run

The variable can be used to define 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 correction 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 |
|---|---|
| A | Type: REAL; unit: °<br>Angle about which the Z, Y or X axis of the reference coordinate system is rotated (only positive direction permissible) |
| B | |
| C | |

## 3.192 $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.

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.

> This variable is not relevant for spline function generators.

**Precondition**

- Technology class #SENSOR (>>> "Tech" Page 107)

**Syntax**

$TECHIN[*Index*]=*Input value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the function generator<br>■ **1 … 6** |
| *Input value* | Type: REAL<br>Value loaded via the sensor input. |

## 3.193 $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<br>Number of the function generator<br>■ **1 … 6** |
| *Index 2* | Type: INT<br>Number of the parameter<br>■ **1 … 10** |
| *Parameter value* | Type: REAL |

## 3.194 $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 correction function.

> This variable is not relevant for spline function generators.

**Syntax**

`$TECHPAR_C[`*Index 1*`,`*Index 2*`]=`*Parameter value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index 1* | Type: INT<br>Number of the function generator<br>■ **1 … 6** |
| *Index 2* | Type: INT<br>Number of the parameter<br>■ **1 … 10** |
| *Parameter value* | Type: REAL |

## 3.195 $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 108)

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Coordinate system* | Type: ENUM<br>■ #BASE<br>■ #ROBROOT<br>■ #TCP<br>■ #TTS<br>■ #WORLD |

## 3.196 $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 correction 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 108)

**Syntax**

$TECHSYS_C=*Coordinate system*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Coordinate system* | Type: ENUM<br>■ #BASE<br>■ #ROBROOT<br>■ #TCP<br>■ #TTS<br>■ #WORLD |

## 3.197 $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.

> If the function value $TECHVAL[] of a spline function generator is written to an analog output ANOUT, no negative DELAY is possible.

**Syntax**

$TECHVAL[*Index*]=*Result*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the function generator:<br>■ **1 … 6** |
| *Result* | Type: REAL<br>Function value of the function generator |

## 3.198 $TIMER[]

**Description**

Timer for cycle time measurement

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<br>Number of the timer<br>- **1 … 64** |
| *Time* | Type: INT; unit: ms<br>Default: **0** |

## 3.199 $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 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 edge in the case of a zero passage.

**Syntax**

`$TIMER_FLAG[`*Index*`]=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the timer<br>- **1 … 64** |
| *State* | Type: BOOL<br>- TRUE: Value greater than zero<br>- FALSE: Value equal to zero |

## 3.200 $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<br>Number of the timer<br>- **1 … 64** |
| *State* | Type: BOOL<br>- TRUE: Timer stopped<br>- FALSE: Timer started |

## 3.201 $TOOL

**Description**

TOOL coordinate system in the advance run

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

- **X**, **Y**, **Z**: Offset of the origin along the axes in [mm]

- **A**, **B**, **C**: Rotational offset of the axis angles in [°]

## 3.202 $TOOL_C

**Description**

TOOL coordinate system in the main run

The variable of structure type FRAME defines the current actual position of the TOOL coordinate system in relation to the FLANGE 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 can only be read.

## 3.203 $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. These 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<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 … E6 |
| *Deviation* | Type: INT; unit: %<br>**Note**: This value cannot be changed by the user. |

## 3.204 $TORQ_DIFF2[]

**Description**

Maximum torque deviation (impact torque)

During program execution, the values of $TORQ_DIFF2[] (the difference between the setpoint torque and actual torque) are calculated. These 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<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 … E6 |
| *Deviation* | Type: INT; unit: %<br>**Note**: This value cannot be changed by the user. |

## 3.205 $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 variable $TORQMON[] contains the current tolerance range for the axis torques in program mode. This tolerance range is defined using the variable $TORQMON_DEF[] in the file …STEU\Mada\$custom.dat.

**Syntax**

`$TORQMON[`*Axis number*`]=`*Factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 … E6 |
| *Factor* | Type: INT; unit: %<br>Default: **200** |

## 3.206 $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 variable $TORQMON_COM[] contains the current tolerance range for the axis torques in jogging. This tolerance range is defined using the variable $TORQMON_COM_DEF[] in the file …STEU\Mada\$custom.dat.

**Syntax**

`$TORQMON_COM[`*Axis number*`]=`*Factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 … E6 |
| *Factor* | Type: INT; unit: %<br>Default: **200** |

## 3.207 $TORQUE_AXIS_ACT[] – KUKA System Software 8.2 and higher

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

**Description**

Current motor 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.)

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

**Syntax**

`$TORQUE_AXIS_ACT[`*Axis number*`]=`*Motor torque*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Motor torque* | Type: REAL; unit: Nm (for linear axes: N) |

## 3.208 $TORQUE_AXIS_LIMITS[] – KUKA System Software 8.2 and higher

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

**Description**

Currently active motor torque limitation for an axis (torque mode)

The variable contains the currently active limits programmed with the function SET_TORQUE_LIMITS() for torque mode.

The variable is primarily intended for diagnosis via the variable correction function or variable overview. In the robot program, the variable triggers an advance run stop.

**Syntax**

`$TORQUE_AXIS_LIMITS[`*Axis number*`]=`*Limits*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Limits* | Type: TorqLimitParam<br>Currently active torque limits |

**TorqLimitParam**

```
STRUC TorqLimitParam REAL lower, upper, SW_ONOFF monitor,
REAL max_vel, max_lag
```

| Element | Description |
|---|---|
| lower | Lower torque limit<br>Unit: Nm (for linear axes: N)<br>Default: -1E10 (unlimited) |
| upper | Upper torque limit<br>Unit: Nm (for linear axes: N)<br>Default: 1E10 (unlimited) |
| monitor | State of the regular monitoring functions<br>■ #ON: Regular monitoring functions are activated.<br>■ #OFF: Regular monitoring functions are deactivated. Instead, the monitoring functions max_vel and max_lag are activated.<br>Default: #ON |
| max_vel | Maximum permissible actual velocity in torque mode (only relevant if the regular monitoring functions are deactivated)<br>Only a positive value may be programmed.<br>Unit: Degrees (for linear axes: mm)<br>Default value (valid for all operating modes): T1 jog velocity * internal safety factor<br>In T1, the maximum velocity with which jogging can be carried out is the default value, even if a higher value is programmed.<br>**Note:** Only set a higher value than the default value if absolutely necessary. |
| max_lag | Maximum permissible following error in torque mode (only relevant if the regular monitoring functions are deactivated)<br>Only a positive value may be programmed.<br>Unit: Degrees (for linear axes: mm)<br>Default value: 5 degrees (for linear axes: 100 mm)<br>**Note:** Only set a higher value than the default value if absolutely necessary. |

**Characteristics:**

- 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 variable $TORQUE_AXIS_LIMITS[] is accessed via KRL, however, the robot controller may regard the access as “invalid”. Recommendation: Check the state of the variable with VARSTATE() prior to access.

## 3.209 $TORQUE_AXIS_MAX[] – KUKA System Software 8.2 and higher

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

**Description**

Absolute maximum motor 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.

The variable is write-protected. Its value is not dependent on the interpreter.

**Syntax**

`$TORQUE_AXIS_MAX[`*Axis number*`]=`*Value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 ... E6 |
| *Value* | Type: REAL; unit: Nm (for linear axes: N)<br>The value specifies an interval: from -*value* to +*value* |

## 3.210 $TORQUE_AXIS_MAX_0[] – KUKA System Software 8.2 and higher

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

**Description**

Maximum permanent motor 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.

The variable is write-protected. Its value is not dependent on the interpreter.

**Syntax**

`$TORQUE_AXIS_MAX_0[`*Axis number*`]=`*Value*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 ... E6 |
| *Value* | Type: REAL; unit: Nm (for linear axes: N)<br>The value specifies an interval: from -*value* to +*value* |

## 3.211 $TRACE

**Description**

Parameters for the TRACE function of the oscilloscope

The variable of structure type TRACE is written in the case of data recording with the oscilloscope. Components of the variable can be used, for example, to start or stop the recording via a program.

**Syntax**

`$TRACE={NAME[]"`*Name*`",MODE `*Mode*`,STATE `*State*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Name of the TRC file: Maximum 7 characters |
| *Mode* | Type: ENUM<br>Recording mode:<br>■ #T_START: Starts the recording.<br>■ #T_STOP: Stops the recording.<br>■ #T_TRIGGER: Starts the trigger process. |
| *Status* | Type: ENUM<br>Status of the recording process:<br>■ #T_END: No recording is currently running.<br>■ #TRIGGERED: Recording in progress.<br>■ #T_WAIT: Waiting for the trigger.<br>■ #T_WRITING: The recorded data are written to the hard drive. |

## 3.212 $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.

## 3.213 $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 (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_MA can be found in the documentation **Configuration of Kinematic Systems**.

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}
```

## 3.214 $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]

The variable is write-protected and can only be read.

## 3.215 $VEL_ACT

**Description**

Current path velocity

**Syntax**

$VEL_ACT=*Velocity*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Velocity* | Type: REAL; unit: m/s<br>■ **0.0 … $VEL_MA.CP** |

Further information about the variable $VEL_MA can be found in the documentation **Configuration of Kinematic Systems**.

## 3.216 $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[] (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_AXIS_MA[] can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

$VEL_AXIS[*Axis number*]=*Velocity*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6 |
| *Velocity* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.217 $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[] (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_AXIS_MA[] can be found in the documentation **Configuration of Kinematic Systems**.

The variable is write-protected and can only be read.

**Syntax**

`$VEL_AXIS_C[`*Axis number*`]=`*Velocity*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6 |
| *Velocity* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.218 $VEL_AXIS_ACT[]

**Description**

Current motor speed

The variable contains the direction of rotation and the speed of the motor as a percentage of the maximum motor speed $VEL_AXIS_MA[] (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_AXIS_MA[] can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

`$VEL_AXIS_ACT[`*Axis number*`]=`*Speed*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Speed* | Type: REAL; unit: %<br>■ **-100.0 … +100.0** |

## 3.219 $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[] (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_AXIS_MA[] can be found in the documentation **Configuration of Kinematic Systems**.

**Syntax**

$VEL_EXTAX[*Axis number*]=*Velocity*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: External axis E1 ... E6 |
| *Velocity* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.220 $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[] (variable in the file …R1\Mada\$machine.dat).

Further information about the variable $VEL_AXIS_MA[] can be found in the documentation **Configuration of Kinematic Systems**.

The variable is write-protected and can only be read.

**Syntax**

$VEL_EXTAX_C[*Axis number*]=*Velocity*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: External axis E1 ... E6 |
| *Velocity* | Type: INT; unit: %<br>■ **1 … 100** |

## 3.221 $WAIT_FOR[]

**Description**

WAIT FOR statement at which an interpreter is currently waiting

The variable can be used to monitor which WAIT FOR statement an interpreter is currently waiting at. The variable is write-protected.

Depending on the specific interpreter, access to the information is as follows:

- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.

- Reading the variable by means of the variable correction 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 (default operating mode):

  - 0: Submit interpreter
  - 1: Robot interpreter

  Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

  - 1: Robot interpreter
  - 2: System submit interpreter
  - 3: Extended submit interpreter 1
  - 4: Extended submit interpreter 2
  - …
  - 9: Extended submit interpreter 7

**Syntax**

$WAIT_FOR[]="*Statement*"

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Statement* | Type: CHAR<br>WAIT FOR statement: max. 2,047 characters |

## 3.222 $WAIT_FOR0[] – KUKA System Software 8.2 and higher

**Description**

WAIT FOR statement at which the submit interpreter is currently waiting

The variable can be used to monitor which WAIT FOR statement the submit interpreter is currently waiting at. The variable is write-protected.

The variable is only available in Single Submit mode. If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), it is not available.

**Syntax**

$WAIT_FOR0[]="*Statement*"

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Statement* | Type: CHAR<br>WAIT FOR statement: max. 2,047 characters |

**Example**

WAIT FOR statement in the submit interpreter

```
DEF submit()
  LOOP
    WAIT FOR $IN[1024 + 2]
  ENDLOOP
END
```

The index resolution for $WAIT_FOR0[] is always active. If $WAIT_FOR_ON0 == TRUE, the submit interpreter waits at the WAIT FOR statement. Then $WAIT_FOR0[] = "WAIT FOR $IN[1026]".

## 3.223 $WAIT_FOR1[] – KUKA System Software 8.2 and higher

**Description**

WAIT FOR statement at which the robot interpreter is currently waiting

The variable can be used to monitor which WAIT FOR statement the robot interpreter is currently waiting at. The variable is write-protected.

**Syntax**

`$WAIT_FOR1[]="`*Statement*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Statement* | Type: CHAR<br>WAIT FOR statement: max. 2,047 characters |

**Example**

WAIT FOR statement in the robot interpreter

```
DEF robot()
  LOOP
    WAIT FOR $IN[1024 + 2]
  ENDLOOP
END
```

The index resolution for $WAIT_FOR1[] is always active. If $WAIT_FOR_ON1 == TRUE, the robot interpreter waits at the WAIT FOR statement. Then $WAIT_FOR1[] = "WAIT FOR $IN[1026]".

## 3.224 $WAIT_FOR_INDEXRES

**Description**

State of the index resolution with reference to the WAIT FOR statement

The variable can be read and changed via the variable correction function.

**Syntax**

`$WAIT_FOR_INDEXRES=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>■ #NO_RESOLUTION: Interpreter is waiting at a condition that cannot be resolved.<br>■ #NO_WAIT: No wait condition is active, and thus also no index resolution.<br>■ #WAIT_INDEX_RES: Interpreter is waiting at a condition and the index resolution is active.<br>■ #WAIT_NO_INDEX_RES: Interpreter is waiting at a condition and the index resolution is not active. |

## 3.225 $WAIT_FOR_ON

**Description**

State of an interpreter with reference to the WAIT FOR condition

The variable can be used to monitor whether an interpreter is currently waiting at a WAIT FOR condition. The variable is write-protected.

Depending on the specific interpreter, access to the information is as follows:

- Reading the variable in a robot program refers to the state of the robot interpreter.
- Reading the variable in a submit program refers to the state of the associated submit interpreter.
- Reading the variable by means of the variable correction 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 (default operating mode):

- 0: Submit interpreter
- 1: Robot interpreter

Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- …
- 9: Extended submit interpreter 7

**Syntax** `$WAIT_FOR_ON=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Interpreter is waiting.<br>- FALSE: Interpreter is not waiting. |

## 3.226 $WAIT_FOR_ON0 – KUKA System Software 8.2 and higher

**Description**

State of the submit interpreter with reference to the WAIT FOR condition

The variable can be used to monitor whether the submit interpreter is currently waiting at a WAIT FOR condition. The variable is write-protected.

The variable is only available in Single Submit mode. If the robot controller is operated in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher), it is not available.

**Syntax** `$WAIT_FOR_ON0=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Interpreter is waiting.<br>- FALSE: Interpreter is not waiting. |

## 3.227 $WAIT_FOR_ON1 – KUKA System Software 8.2 and higher

**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. The variable is write-protected.

**Syntax** `$WAIT_FOR_ON1=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Interpreter is waiting.<br>- FALSE: Interpreter is not waiting. |

## 3.228 $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. The variable is write-protected.

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 correction 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 (default operating mode):

  - 0: Submit interpreter
  - 1: Robot interpreter

  Robot controller in Multi-Submit mode (only possible with KUKA System Software 8.3 and higher):

  - 1: Robot interpreter
  - 2: System submit interpreter
  - 3: Extended submit interpreter 1
  - 4: Extended submit interpreter 2
  - …
  - 9: Extended submit interpreter 7

**Syntax**

`$WAIT_STATE=`*Condition*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Condition* | Type: ENUM<br>- #NOT_WAITING : No wait condition is active.<br>- #WAIT_WORKSPACE: Robot is waiting for a workspace to be enabled.<br>- #WAIT_PROGSYNC: When a PROGSYNC command is reached, the robot waits for all cooperating robots to reach this command (only relevant in RoboTeam).<br>- #WAIT_REMOTECMD: Robot waits for a remote statement, e.g. via the network.<br>- #WAIT_BOOL_EXPR: Robot waits for a Boolean expression. |

## 3.229 $WBOXDISABLE

**Description**

State of workspace monitoring

**Syntax**

`$WBOXDISABLE=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>- TRUE: Monitoring is active.<br>- FALSE: Monitoring is not active.<br>Default: FALSE |

## 3.230 $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}
```

The variable is write-protected and can only be read.

# 4 Machine data ...STEU\Mada\$machine.dat

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

- 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. in the **SafeOperation** documentation).

> **WARNING** These signals are not redundant in design and can supply incorrect information. 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.

Input signals that are not required can be deactivated with FALSE.

**Output signals**

Some output signals are preset to FALSE. It is not essential to assign output numbers to them. This only has to be done if it is desirable for the signals to be read (e.g. via the variable correction function or program run).

Some output signals are already assigned to output numbers. These signals can be assigned to other output numbers. Signals that are not required can be deactivated with FALSE.

## 4.1 $ALARM_STOP

**Description**

Signal declaration for the EMERGENCY STOP

There is no EMERGENCY STOP if this output is set.

The output is reset in the following EMERGENCY STOP situations:

- For an EMERGENCY STOP via the EMERGENCY STOP device on the smartPAD
- For an EMERGENCY STOP via the external EMERGENCY STOP device

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>Default: **1013** |

## 4.2 $ALARM_STOP_INTERN

**Description**

Signal declaration for the internal EMERGENCY STOP

This output is set in the case of an internal EMERGENCY STOP.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Output number* | Type: INT<br>Default: **853** |

## 4.3 $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<br>Default: **995** |

## 4.4 $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<br>Default: **1026** |

## 4.5 $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<br>Default: **1025** |

## 4.6 $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<br>Default: **140** |

## 4.7 $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<br>Default: **996** |

## 4.8 $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<br>Default: **1026** |

## 4.9 $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<br>Default: **1025** |

## 4.10 $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<br>Default: **140** |

## 4.11 $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<br>Default: **1000** |

## 4.12 $IN_HOME1 … $IN_HOME5

**Description**

Signal declaration for reaching the HOME position 1 ... 5

The variable $AXIS_HOME allows up to 5 HOME positions to be defined in the machine data (in addition to the HOME position defined using $H_POS). By setting the output, the robot controller communicates to the higher-level controller that the robot is located in HOME position 1 ... 5.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Index of the HOME position<br>■ **1 … 5** |
| *Output number* | Type: INT<br>Default: **977 … 981** |

## 4.13 $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<br>Default: **1025** |

## 4.14 $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<br>Default: **150** |

## 4.15 $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<br>Default: **997** |

## 4.16 $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<br>Default: **147** |

## 4.17 $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<br>Default: **1012** |

## 4.18 $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<br>Default: **1021** |

## 4.19 $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<br>Default: **1022** |

## 4.20 $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<br>Default: **137** |

## 4.21 $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<br>Default: **1001** |

## 4.22 $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<br>Default: **1023** |

## 4.23 $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<br>Default: **1010** |

## 4.24 $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<br>Default: **993** |

## 4.25 $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<br>Default: **994** |

## 4.26 $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<br>Default: **1011** |

## 4.27 $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<br>Default: **1026** |

## 4.28 $IMM_STOP

**Description**

Signal declaration for the EMERGENCY STOP

By setting this input, the higher-level controller can trigger an EMERGENCY STOP.

**Syntax**

`SIGNAL $IMM_STOP $IN[`*Input number*`]`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Input number* | Type: INT<br>Default: **1025** |

## 4.29 $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<br>Default: **138** |

## 4.30 $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<br>Default: **1025** |

## 4.31 $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<br>Default: **139** |

## 4.32 $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<br>Default: **1025** |

## 4.33 $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 $AXWORKSTATEIndex $OUT[Output number]
```

**Explanation of the syntax**

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

## 4.34 $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 $WORKSTATEIndex $OUT[Output number]
```

**Explanation of the syntax**

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

## 4.35 $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<br>■ #ON: Path-maintaining EMERGENCY STOP is activated.<br>■ #OFF: Path-maintaining EMERGENCY STOP is deactivated.<br>Default: #ON |

## 4.36 $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<br>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

# 5 Machine data ...R1\Mada\$machine.dat

## 5.1 $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}
```

## 5.2 $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<br>- TRUE: Monitoring is active.<br>- FALSE: Monitoring is not active.<br>Default: FALSE |

## 5.3 $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}
```

## 5.4 $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}
```

## 5.5 $AXIS_HOME[]

**Description**

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

The variable of structure type E6AXIS allows 5 further HOME positions to be defined (in addition to the HOME position defined using $H_POS).

- 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}
```

## 5.6 $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<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 ... E6 |
| *Limit value* | Type: REAL; unit: %<br>Default: **150.0**<br>**Note:** This value must not be changed. |

## 5.7 $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<br>■ **Bit n = 0**: The reference point is approached in the positive direction.<br>■ **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.

## 5.8 $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<br>Default: **100** |

## 5.9 $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}
```

## 5.10 $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}
```

## 5.11 $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}
```

## 5.12 $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.

> 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: %<br>■ **150 … 500**<br>Default: **200**<br>**Note:** If the value zero is entered here, the monitoring is deactivated. |

## 5.13 $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=`*Offset*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Offset* | Type: ENUM<br>■ **#NONE**: No offset is saved.<br>■ **#RDC**: An offset is saved.<br>During mastering, the robot controller accesses the offset data on the hard drive and calculates the exact mastering position.<br>**Note**: The offset is not taken into consideration during reference mastering.<br>**Note**: If #RDC is used, the mastering position of the manipulator must be stored in $MAMES[*x*]. |

## 5.14 $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<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 ... E6 |
| *Axis value* | Type: REAL; unit: ° (for linear axes: mm)<br>■ **-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
```

## 5.15 $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<br>- **TRUE**: Monitoring is activated.<br>- **FALSE**: Monitoring is not activated.<br>Default: **TRUE** |

## 5.16 $NUM_AX

**Description** Number of robot axes

**Syntax** `$NUM_AX=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Number* | Type: INT<br>- **1 … 6** |

**Example** Robot with 6 axes

```
$NUM_AX=6
```

## 5.17 $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<br>- **0**: The orientation at the end point is not checked (default setting for 6-axis robots).<br>- **1**: A check is made to see if the taught end point has been reached (default setting for 5-axis robots). |

## 5.18 $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<br>- **1 … 12** |
| *Bit array* | Bit array that specifies the axis to be mastered<br>- **Bit n = 0**: The axis is not mastered.<br>- **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'
```

## 5.19 $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<br>■ **0.0 … 0.25**<br>Default: **12:25 AM a.m.** |

## 5.20 $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<br>Default: **50**<br>**Note:** This value must not be changed. |

## 5.21 $TRAFONAME[]

**Description**

Name of coordinate transformation

This variable can be used to assign the coordinate transformation a symbolic name. This name is compared with the $ROBTRAFO[] robot name programmed on the RDC.

**Syntax**

`$TRAFONAME[]="`*Name*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Name of the coordinate transformation: max. 32 characters |

**Example**

```
$TRAFO_NAME[]="#KR16 C2 FLR ZH16"
```

## 5.22 $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<br>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

# 6 Machine data …STEU\Mada\$custom.dat

## 6.1 $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<br>■ **TRUE**: Conversion to positionally accurate Cartesian robot poses<br>■ **FALSE**: No conversion to positionally accurate Cartesian robot poses<br>Default: **FALSE** |

## 6.2 $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<br>■ **1 … 20** |
| *Start bit* | Type: INT<br>■ **1 … 4 096**<br>**Note**: This is the maximum possible range of inputs. $SET_IO_SIZE can be used to reduce the range of inputs.<br>Default: **1** |

| Element | Description |
|---|---|
| *Bit width* | Type: INT<br>- **0 … 32**<br>Default: **0** |
| *Type* | Type: ENUM<br>- **#NONE**: Parity bit is not activated.<br>- **#EVEN**: Parity bit is activated.<br>  - If the parity sum is even, the parity bit has the value 0.<br>  - If the parity sum is odd, the parity bit has the value 1.<br>- **#ODD**: Parity bit is activated.<br>  - If the parity sum is odd, the parity bit has the value 0.<br>  - If the parity sum is even, the parity bit has the value 1.<br>Default: **#NONE** |

## 6.3 $BIN_OUT[]

**Description**

Configuration of 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<br>- **1 … 20** |
| *Start bit* | Type: INT<br>- **1 … 4 096**<br>**Note**: This is the maximum possible range of outputs. $SET_IO_SIZE can be used to reduce the range of outputs.<br>Default: **1** |
| *Bit width* | Type: INT<br>- **0 … 32**<br>Default: **0** |
| *Type* | Type: ENUM<br>- **#NONE**: Parity bit is not activated.<br>- **#EVEN**: Parity bit is activated.<br>  - If the parity sum is even, the parity bit has the value 0.<br>  - If the parity sum is odd, the parity bit has the value 1.<br>- **#ODD**: Parity bit is activated.<br>  - If the parity sum is odd, the parity bit has the value 0.<br>  - If the parity sum is even, the parity bit has the value 1.<br>Default: **#NONE** |

## 6.4 $COUNT_I[]

**Description**

Freely usable variable for a counter

**Syntax**

`$COUNT_I[`*Index*`]=`*Number*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Counter number<br>■ **1 … 64** |
| *Number* | Type: INT |

## 6.5 $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°.

$CP_STATMON can be edited via the variable correction function or in the $CUSTOM.DAT file.

**Syntax**

`$CP_STATMON=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: ENUM<br>■ **#CHECK_S**: Status is checked.<br>■ **#CHECK_TS**: Status and Turn are checked.<br>■ **#NONE**: The check is deactivated.<br>Default: **#NONE** |

## 6.6 $CP_VEL_TYPE

**Description**

Reduction of the axis velocity for CP motions

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=*Reduction type*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Reduction type* | Type: ENUM<br>■ **#CONSTANT**: No reduction of axis velocity<br>■ **#VAR_ALL**: Reduction of axis velocity in all modes<br>■ **#VAR_ALL_MODEL**: Model-based reduction of axis velocity in program mode<br>■ **#VAR_T1**: Reduction of axis velocity in T1<br>**Note**: Reduction of axis velocity is always active in Cartesian jogging. |

## 6.7 $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: °<br>Default: **0.0** |

## 6.8 $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 |

## 6.9 $PRO_I_O[] – KUKA System Software 8.1 and 8.2

**Description**

Path to the submit program

At a cold start of the robot controller, the submit interpreter automatically starts the program specified here. By default, this is SPS.SUB.

**Syntax**

`$PRO_I_O[]="`*Name*`"`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Directory and name of the program: max. 64 characters |

**Example**

Default path to the submit program

```
$PRO_I_O[]="/R1/SPS()"
```

## 6.10 $PRO_I_O_SYS – KUKA System Software 8.3 and higher

**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 this assignment. It is necessary to enable the controller-internal submit tasks to be executed. If this 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 using the variable correction function.

**Syntax**

`$PRO_I_O_SYS={MODULE[] "`*Name*`", COLD_BOOT_RUN `*Cold start*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Name* | Type: CHAR<br>Directory and name of the submit program: max. 64 characters |
| *Cold start* | Type: ENUM SW_ONOFF<br>Start of the submit program at a cold start<br>■ #ON: Submit program is started.<br>■ #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}
```

## 6.11 $PRO_I_O_PROC_ID3…9 – KUKA System Software 8.3 and higher

**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 using the variable correction function.

**Syntax**

`$PRO_I_O_PROC_ID`*Index*`={MODULE[] "`*Name*`", COLD_BOOT_RUN `*Cold start*`}`

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Index* | Type: INT<br>Number of the extended submit interpreter<br>■ 3: Extended submit interpreter 1<br>■ 4: Extended submit interpreter 2<br>■ …<br>■ 9: Extended submit interpreter 7 |
| *Name* | Type: CHAR<br>Directory and name of the submit program: max. 64 characters |
| *Cold start* | Type: ENUM SW_ONOFF<br>Start of the submit program at a cold start<br>■ #ON: Submit program is started.<br>■ #OFF: Submit program is not started. |

## 6.12 $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<br>■ TRUE: Reduction by $RED_T1<br>■ FALSE: Reduction to $VEL_CP_T1<br>Default: TRUE |

## 6.13 $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<br>- **#SOURCE**: The Status of the start point is adopted.<br>- **#BEST**: The 8 possible Status combinations are calculated. The combination resulting in the shortest motion from the start to the target point is used.<br>Default: **#SOURCE** |

## 6.14 $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<br>- **1 … 6**: Robot axis A1 ... A6 |
| *State* | Type: BOOL<br>- **TRUE**: Analog value filter is deactivated.<br>- **FALSE**: Analog value filter is activated.<br>Default: **TRUE** |

## 6.15 $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.<br>■ **Bit n = 0**: Function is not active.<br>■ **Bit n = 1**: Function is active. |

| | | | | | | |
|---|---|---|---|---|---|---|
| Bit n | 5 | 4 | 3 | 2 | 1 | 0 |
| Function | 6 | 5 | 4 | 3 | 2 | 1 |

## 6.16 $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 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)

> Further information about torque monitoring is contained in the “Operating and Programming Instructions for System Integrators”.

**Syntax**

`$TORQMON_COM_DEF[`*Axis number*`]=`*Factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>■ **1 … 6**: Robot axis A1 ... A6<br>■ **7 … 12**: External axis E1 … E6 |
| *Factor* | Type: INT; unit: %<br>Default: **200** |

## 6.17 $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 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)

> Further information about torque monitoring is contained in the “Operating and Programming Instructions for System Integrators”.

**Syntax**

`$TORQMON_DEF[`*Axis number*`]=`*Factor*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Axis number* | Type: INT<br>- **1 … 6**: Robot axis A1 ... A6<br>- **7 … 12**: External axis E1 … E6 |
| *Factor* | Type: INT; unit: %<br>Default: **200** |

## 6.18 $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<br>Default: **0.0** |

# 7 Machine data …STEU\Mada\$option.dat

## 7.1 $CHCK_MOVENA

**Description**

Checking the input number of $MOVE_ENABLE

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

**Syntax**

`$CHCK_MOVENA=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: $MOVE_ENABLE is checked.<br>■ **FALSE**: $MOVE_ENABLE is not checked.<br>Default: **TRUE** |

## 7.2 $IDENT_OPT

**Description**

Enabling of load data determination

**Syntax**

`$IDENT_OPT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: Load data can be determined.<br>■ **FALSE**: No load data can be determined.<br>Default: **TRUE** |

## 7.3 $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 |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: Improved approximate positioning is activated.<br>■ **FALSE**: Improved approximate positioning is deactivated.<br>Default: **TRUE** |

## 7.4 $LOOP_CONT

**Description**

Simulation of the condition for terminating a wait statement

If a wait message $LOOP_MSG is active, the variable $LOOP_CONT is set to TRUE. The variable $LOOP_CONT is reset again if the **Simulate** softkey is used to simulate that the condition programmed in a wait statement is fulfilled.

**Syntax**

`$LOOP_CONT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: A wait message relating to a wait statement has been activated.<br>■ **FALSE**: Condition for terminating the wait statement has been simulated (**Simulate** softkey).<br>Default: **FALSE** |

## 7.5 $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** softkey 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 by pressing the **Simulate** softkey. Precondition: Operating mode T1 or T2.

**Syntax**

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

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Message* | Type: CHAR<br>Message text: max. 128 characters |

## 7.6 $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.

(>>> 3.102 "$MOT_STOP" Page 66)

**Syntax**

`$MOT_STOP_OPT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: Option is activated.<br>■ **FALSE**: Option is not activated.<br>Default: **FALSE** |

## 7.7 $SINGUL_STRATEGY

**Description** Strategy for singularity-free motion

**Syntax** `$SINGUL_STRATEGY=`*Strategy*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *Strategy* | Type: INT<br>■ **0**: No strategy<br>■ **1**: Approximation strategy (= moving through a singularity by means of changes in orientation)<br>Default: **0** |

## 7.8 $TCP_IPO

**Description** Interpolation mode

The interpolation mode is specified in the **Frames** option window (programming of motions).

**Syntax** `$TCP_IPO=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: The tool is a fixed tool.<br>■ **FALSE**: The tool is mounted on the mounting flange.<br>Default: **FALSE** |

## 7.9 $TECH_OPT

**Description** Operating state of the function generator

**Syntax** `$TECH_OPT=`*State*

**Explanation of the syntax**

| Element | Description |
|---|---|
| *State* | Type: BOOL<br>■ **TRUE**: Function generator is active.<br>■ **FALSE**: Function generator is not active.<br>Default: **TRUE** |

## 7.10 $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 |
|---|---|
| *State* | Type: BOOL<br>- **TRUE**: Override is reduced.<br>- **FALSE**: Override is not reduced.<br>Default: **TRUE** |

## 7.11 $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 |
|---|---|
| *State* | Type: BOOL<br>- **TRUE**: Option is activated.<br>- **FALSE**: Option is not activated.<br>Default: **FALSE** |

# 8 KUKA Service

## 8.1 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
- As comprehensive information as possible 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
  - Diagnostic package KRCDiag

    Additionally for KUKA Sunrise: Existing projects including applications

    For versions of KUKA System Software older than V8: Archive of the software (KRCDiag is not yet available here.)
  - Application used
  - External axes used

## 8.2 KUKA Customer Support

**Availability**

KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

**Argentina**

Ruben Costantini S.A. (Agency)  
Luis Angel Huergo 13 20  
Parque Industrial  
2400 San Francisco (CBA)  
Argentina  
Tel. +54 3564 421033  
Fax +54 3564 428877  
ventas@costantini-sa.com

**Australia**

KUKA Robotics Australia Pty Ltd  
45 Fennell Street  
Port Melbourne VIC 3207  
Australia  
Tel. +61 3 9939 9656  
info@kuka-robotics.com.au  
www.kuka-robotics.com.au

**Belgium**

KUKA Automatisering + Robots N.V.
Centrum Zuid 1031
3530 Houthalen
Belgium
Tel. +32 11 516160
Fax +32 11 526794
info@kuka.be
www.kuka.be

**Brazil**

KUKA Roboter do Brasil Ltda.
Travessa Claudio Armando, nº 171
Bloco 5 - Galpões 51/52
Bairro Assunção
CEP 09861-7630 São Bernardo do Campo - SP
Brazil
Tel. +55 11 4942-8299
Fax +55 11 2201-7883
info@kuka-roboter.com.br
www.kuka-roboter.com.br

**Chile**

Robotec S.A. (Agency)
Santiago de Chile
Chile
Tel. +56 2 331-5951
Fax +56 2 331-5952
robotec@robotec.cl
www.robotec.cl

**China**

KUKA Robotics China Co., Ltd.
No. 889 Kungang Road
Xiaokunshan Town
Songjiang District
201614 Shanghai
P. R. China
Tel. +86 21 5707 2688
Fax +86 21 5707 2603
info@kuka-robotics.cn
www.kuka-robotics.com

**Germany**

KUKA Roboter GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-1926
Fax +49 821 797-41 1926
Hotline.robotics.de@kuka.com
www.kuka-roboter.de

**France**

KUKA Automatisme + Robotique SAS
Techvallée
6, Avenue du Parc
91140 Villebon S/Yvette
France
Tel. +33 1 6931660-0
Fax +33 1 6931660-1
commercial@kuka.fr
www.kuka.fr

**India**

KUKA Robotics India Pvt. Ltd.
Office Number-7, German Centre,
Level 12, Building No. - 9B
DLF Cyber City Phase III
122 002 Gurgaon
Haryana
India
Tel. +91 124 4635774
Fax +91 124 4635773
info@kuka.in
www.kuka.in

**Italy**

KUKA Roboter Italia S.p.A.
Via Pavia 9/a - int.6
10098 Rivoli (TO)
Italy
Tel. +39 011 959-5013
Fax +39 011 959-5141
kuka@kuka.it
www.kuka.it

**Japan**

KUKA Robotics Japan K.K.
YBP Technical Center
134 Godo-cho, Hodogaya-ku
Yokohama, Kanagawa
240 0005
Japan
Tel. +81 45 744 7691
Fax +81 45 744 7696
info@kuka.co.jp

**Canada**

KUKA Robotics Canada Ltd.
6710 Maritz Drive - Unit 4
Mississauga
L5W 0A1
Ontario
Canada
Tel. +1 905 670-8600
Fax +1 905 670-8604
info@kukarobotics.com
www.kuka-robotics.com/canada

**Korea**

KUKA Robotics Korea Co. Ltd.
RIT Center 306, Gyeonggi Technopark
1271-11 Sa 3-dong, Sangnok-gu
Ansan City, Gyeonggi Do
426-901
Korea
Tel. +82 31 501-1451
Fax +82 31 501-1461
info@kukakorea.com

**Malaysia**

KUKA Robot Automation (M) Sdn Bhd
South East Asia Regional Office
No. 7, Jalan TPP 6/6
Taman Perindustrian Puchong
47100 Puchong
Selangor
Malaysia
Tel. +60 (03) 8063-1792
Fax +60 (03) 8060-7386
info@kuka.com.my

**Mexico**

KUKA de México S. de R.L. de C.V.
Progreso #8
Col. Centro Industrial Puente de Vigas
Tlalnepantla de Baz
54020 Estado de México
Mexico
Tel. +52 55 5203-8407
Fax +52 55 5203-8148
info@kuka.com.mx
www.kuka-robotics.com/mexico

**Norway**

KUKA Sveiseanlegg + Roboter
Sentrumsvegen 5
2867 Hov
Norway
Tel. +47 61 18 91 30
Fax +47 61 18 62 00
info@kuka.no

**Austria**

KUKA Roboter CEE GmbH
Gruberstraße 2-4
4020 Linz
Austria
Tel. +43 7 32 78 47 52
Fax +43 7 32 79 38 80
office@kuka-roboter.at
www.kuka.at

**Poland**

KUKA Roboter Austria GmbH
Spółka z ograniczoną odpowiedzialnością
Oddział w Polsce
Ul. Porcelanowa 10
40-246 Katowice
Poland
Tel. +48 327 30 32 13 or -14
Fax +48 327 30 32 26
ServicePL@kuka-roboter.de

**Portugal**

KUKA Robots IBÉRICA, S.A.
Rua do Alto da Guerra n° 50
Armazém 04
2910 011 Setúbal
Portugal
Tel. +351 265 729 780
Fax +351 265 729 782
info.portugal@kukapt.com
www.kuka.com

**Russia**

KUKA Robotics RUS
Werbnaja ul. 8A
107143 Moskau
Russia
Tel. +7 495 781-31-20
Fax +7 495 781-31-19
info@kuka-robotics.ru
www.kuka-robotics.ru

**Sweden**

KUKA Svetsanläggningar + Robotar AB
A. Odhners gata 15
421 30 Västra Frölunda
Sweden
Tel. +46 31 7266-200
Fax +46 31 7266-201
info@kuka.se

**Switzerland**

KUKA Roboter Schweiz AG
Industriestr. 9
5432 Neuenhof
Switzerland
Tel. +41 44 74490-90
Fax +41 44 74490-91
info@kuka-roboter.ch
www.kuka-roboter.ch

**Spain**

KUKA Robots IBÉRICA, S.A.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
Fax +34 93 8142-950
comercial@kukarob.es
www.kuka.es

**South Africa**

Jendamark Automation LTD (Agency)
76a York Road
North End
6000 Port Elizabeth
South Africa
Tel. +27 41 391 4700
Fax +27 41 373 3869
www.jendamark.co.za

**Taiwan**

KUKA Robot Automation Taiwan Co., Ltd.
No. 249 Pujong Road
Jungli City, Taoyuan County 320
Taiwan, R. O. C.
Tel. +886 3 4331988
Fax +886 3 4331948
info@kuka.com.tw
www.kuka.com.tw

**Thailand**

KUKA Robot Automation (M)SdnBhd
Thailand Office
c/o Maccall System Co. Ltd.
49/9-10 Soi Kingkaew 30 Kingkaew Road
Tt. Rachatheva, A. Bangpli
Samutprakarn
10540 Thailand
Tel. +66 2 7502737
Fax +66 2 6612355
atika@ji-net.com
www.kuka-roboter.de

**Czech Republic**

KUKA Roboter Austria GmbH
Organisation Tschechien und Slowakei
Sezemická 2757/2
193 00 Praha
Horní Počernice
Czech Republic
Tel. +420 22 62 12 27 2
Fax +420 22 62 12 27 0
support@kuka.cz

**Hungary**

KUKA Robotics Hungaria Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

**USA**

KUKA Robotics Corporation
51870 Shelby Parkway
Shelby Township
48315-1787
Michigan
USA
Tel. +1 866 873-5852
Fax +1 866 329-5852
info@kukarobotics.com
www.kukarobotics.com

**UK**

KUKA Robotics UK Ltd
Great Western Street
Wednesbury West Midlands
WS10 7LL
UK
Tel. +44 121 505 9970
Fax +44 121 505 6589
service@kuka-robotics.co.uk
www.kuka-robotics.co.uk

# Index