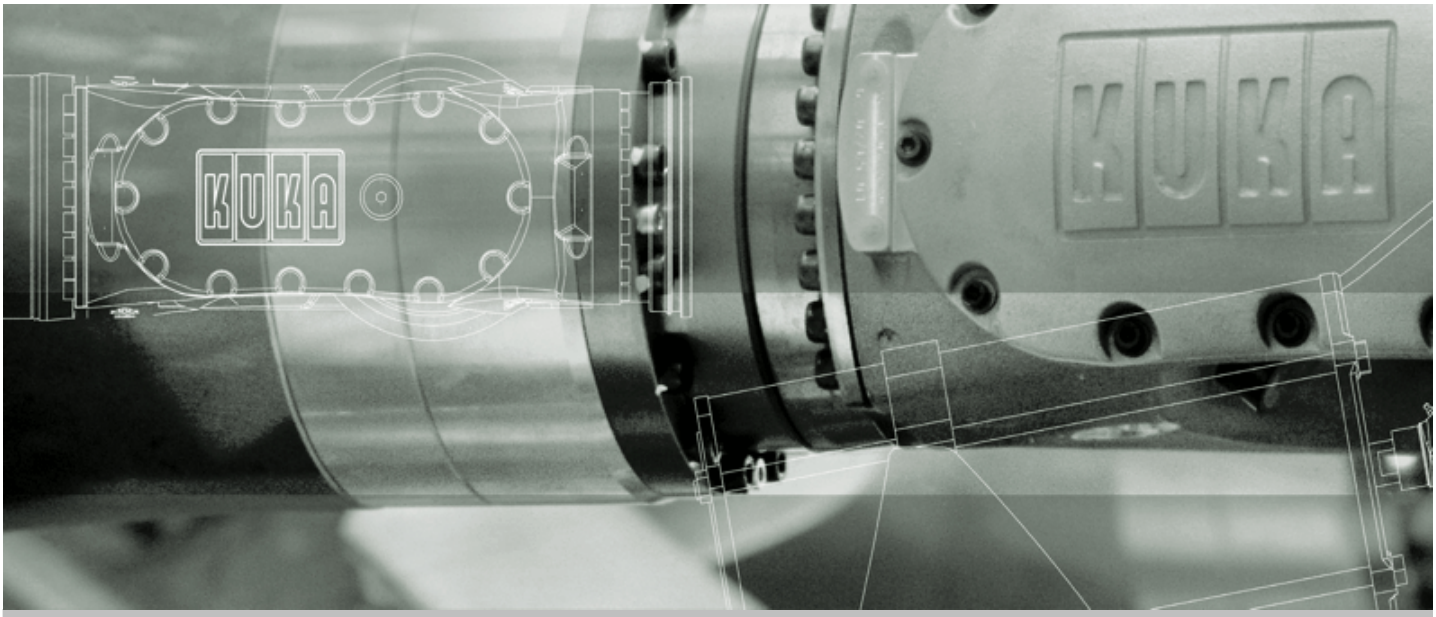

KUKA Robot Group

Communication

CREAD/CWRITE

Programming CREAD/CWRITE and related statements

For KUKA System Software (KSS) 5.4, 5.5, 7.0

Issued: 19.06.2007 Version: 1.3

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.

KIM-PS4-DOC

Contents

1 Introduction

1.1 Target group

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

- Advanced KRL programming skills
- Advanced knowledge of the robot controller system
- Advanced system knowledge of the controllers with which the KR C controller communicates

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 Robot system documentation

The robot system documentation consists of the following parts:

- Operating instructions for the robot
- Operating instructions for the robot controller
- Operating and programming instructions for the KUKA System Software
- Documentation relating to options and accessories

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

Warnings marked with this pictogram are relevant to safety and **must** be observed.

Danger!
This warning means that death, severe physical injury or substantial material damage **will** occur, if no precautions are taken.

Warning!
This warning means that death, severe physical injury or substantial material damage **may** occur, if no precautions are taken.

Caution!
This warning means that minor physical injuries or minor material damage **may** occur, if no precautions are taken.

Notes

Notes marked with this pictogram contain tips to make your work easier or references to further information.

Tips to make your work easier or references to further information.

2 Description of functions

Functions

CREAD and CWRITE are flexible statements which can be used to communicate between the robot controller and another controller. They can also be used for communication within the robot controller.

CREAD reads data from a channel. CWRITE writes data to a channel.

CREAD/CWRITE can be used for communication via the following channels:

- Serial interfaces
  (>>> 3.1 "Communication via a serial interface" page 9)
- External modules
  (>>> 3.2 "Communication via external modules" page 9)
  Communication via external modules is not possible in KSS 7.0.
- Command channel (CWRITE only)
  (>>> 3.3 "Communication via the command channel" page 10)

Example

The robot controller receives position data from another controller (e.g. from a camera system) at the serial interface. The robot controller uses CREAD to read these position data from the serial interface.

3 Communication channels

3.1 Communication via a serial interface

Description

The robot controller can use CREAD/CWRITE to communicate with another controller via a serial interface. The data are transferred in real time.

The controllers with which communication is carried out are generally intelligent systems. Examples:

- Camera systems, e.g. Perceptron
- Other robot controllers
- Intelligent sensor systems, e.g. force/torque sensors

Configuration

The serial interface must be configured for communication with CREAD/CWRITE.

(>>> 4.1 "Overview of configuration of the serial interface" page 11)

Overview

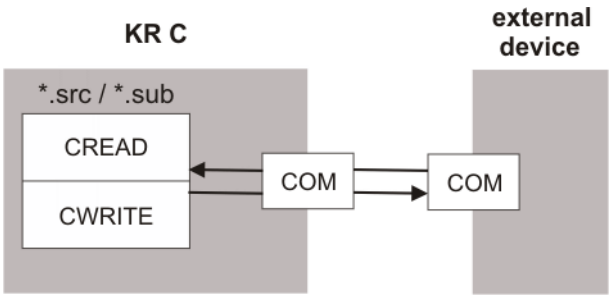

Fig. 3-1: Communication via a serial interface

3.2 Communication via external modules

Communication via external modules is not possible in KSS 7.0.

Description

External modules are drivers for interfaces, e.g. for TCP/IP, for Ethernet interfaces or for serial interfaces. An external module is always implemented outside the robot controller as an O file and then integrated into the robot controller.

The implementation and integration of external modules is not covered by this documentation. This documentation deals with the use of CREAD/CWRITE to communicate with integrated external modules.

External modules can be used both for communication within the robot controller and for communication with other controllers.

There are 2 types of external modules:

- **LD_EXT_OBJ**

  This type can be used to exchange data by means of CREAD and CWRITE.
- **LD_EXT_FCT**

  This type contains functions. The functions are called via CWRITE. LD_EXT_FCT can return function parameters to CWRITE. (CREAD is not possible with this type.)

The robot controller can communicate with a maximum of 4 external modules (2 per type) simultaneously.

Configuration

The external modules must be configured for communication with CREAD/CWRITE.

(>>> 5 "Configuring the external modules" page 19)

Overview

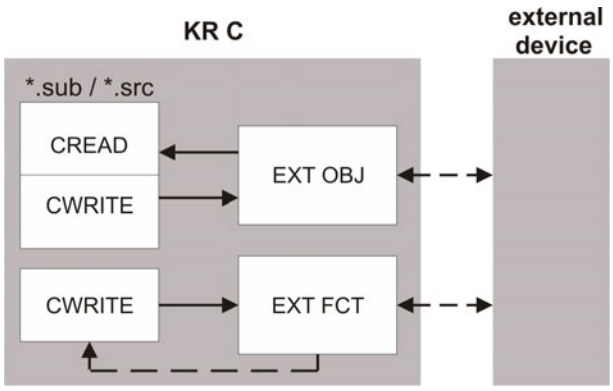

Fig. 3-2: Communication via external modules

3.3 Communication via the command channel

Description

CWRITE can transfer statements to a program interpreter via the command channel. Example: start a program via the command channel with RUN and stop it with STOP.

CREAD is not relevant for the command channel.

Configuration

The command channel does not need to be configured for communication with CWRITE.

Overview

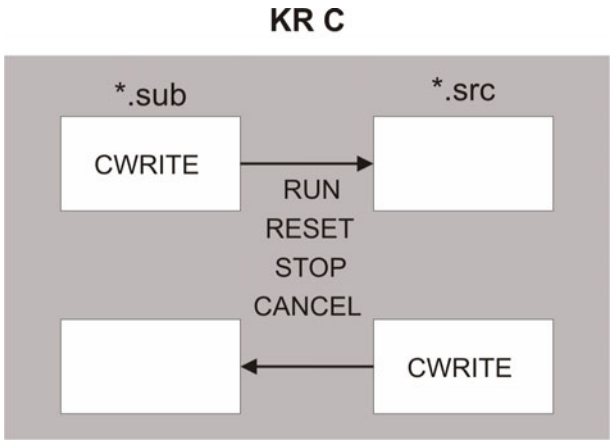

Fig. 3-3: Communication via the command channel

4 Configuring the serial interface

4.1 Overview of configuration of the serial interface

Step	Description
1	Check the assignment of the serial interface. **Note:** This step is not necessary in KSS 7.0. (>>> 4.2 "Assigning the serial interface to the operating system" page 11)
2	Configure the serial interface. A transmission procedure must be selected. (>>> 4.3 "Configuring the serial interface" page 11)
3	Configure the transmission procedure selected in step 2. Either (>>> 4.4 "Configuring the 3964R procedure" page 12) Or (>>> 4.5 "Configuring the Xon/Xoff protocol" page 13)
4	If the transmitted data are to be displayed using Telnet: activate Telnet. (>>> 4.6 "Displaying sent/received data with Telnet" page 14)
5	Reboot the robot controller with a cold start.

4.2 Assigning the serial interface to the operating system

Description

The serial interface must be assigned to the VxWorks operating system so that it can be used for data transmission with CREAD/CWRITE.

- KSS 5.4 and 5.5:
  Only COM3 is available for data transmission with CREAD/CWRITE. By default, COM3 is assigned to VxWorks.
- KSS 7.0:
  Only COM2 is available for data transmission with CREAD/CWRITE. By default, COM2 is assigned to VxWorks.

File

- KSS 5.4 and 5.5:
  The assignment of the serial interface to the operating system is defined in the file HW_INF.INI.

```
[SERIAL]
;ENABLE:  COM is accessible by robot (vxWorks)
;DISABLE: COM is accessible by WinXP
;CONSOLE: for developer only
COM3=ENABLE ;[ENABLE, DISABLE, CONSOLE]
```

- KSS 7.0:
  The assignment of the serial interface to the operating system is defined, by default, in KRC.XML. The user does not need to make any changes.

4.3 Configuring the serial interface

File

The serial interface is configured in the file SERIAL.INI.

```
[COM3]
BAUD=9600
CHAR_LEN=8 ; 7,8
STOP_BIT=1 ; 1,2 at time not changeable
PARITY=2   ; EVEN=2, ODD=1, NONE=0
PROC=1     ; 3964R=1, SRVT=2, WTC=3, XONXOFF=4
```

Parameter

Parameter	Description
BAUD	Data transmission speed in baud (= bits per second). Default: 9 600 baud. For the robot controller to be able to communicate with another controller via the serial interface, the baud rate must be the same for both controllers. Permissible baud rates in the KR C: ■ 57 600 ■ 38 400 ■ 19 200 ■ 9 600 ■ 4 800 ■ 2 400 ■ 1 200 ■ 600 ■ 300 ■ 150 ■ 110
CHAR_LEN	Number of data bits. Communication is generally in bytes, so the number is usually 8.
STOP_BIT	Number of stop bits. Currently invariably set to 1.
PARITY	Parity. This parameter is only relevant if the protocol 3964R is selected for the parameter PROC. ■ 0 = the parity is not taken into consideration. ■ 1 = odd parity, i.e. the number of bits including the parity bit is odd. ■ 2 = even parity, i.e. the number of bits including the parity bit is even.
PROC	Transmission procedure. The protocol 3964R is generally used for the transmission procedure. If the controller with which the robot controller is communicating is unable to cope with the 3964R procedure, the simpler Xon/Xoff protocol can be used. SRVT and WTC protocols: only for internal use by KUKA.

4.4 Configuring the 3964R procedure

File

The 3964R procedure is configured in the file SERIAL.INI.

```
[3964R]
CHAR_TIMEOUT=500   ; msec
QUITT_TIMEOUT=500  ; msec
TRANS_TIMEOUT=500  ; msec
MAX_TX_BUFFER=2    ; 1...5
MAX_RX_BUFFER=10   ; 1...20
SIZE_RX_BUFFER=100 ; 1...2048
PROTOCOL_PRIOR=1   ; HIGH=1, LOW=0
```

Parameter

Parameter	Description
CHAR_TIMEOUT	The maximum interval between two characters
QUITT_TIMEOUT	The maximum wait time of the robot controller for the character DLE
TRANS_TIMEOUT	This is the maximum length of time the robot controller waits, after receiving an incorrect BCC, for the other controller to establish a new connection.
MAX_TX_BUFFER	Maximum number of output buffers
MAX_RX_BUFFER	Maximum number of receive buffers
SIZE_RX_BUFFER	The size of a receive buffer in bytes (It is not necessary to define a size for the output buffers, as the size automatically adapts to the contents.)
PROTOCOL_PRIOR	Priority. The priority serves to avoid initialization conflicts.

4.5 Configuring the Xon/Xoff protocol

Description

The Xon/Xoff protocol is a common handshake procedure. Handshake procedures are also known as low-level protocols. They stop the transmitter of the partner controller if the receive buffer is threatening to overflow.

File

The Xon/Xoff protocol is configured in the file SERIAL.INI.

```
[XONXOFF]
CHAR_TIMEOUT=50     ;msec Timout after last received character
CHAR_TIMEOUT=50     ;to recognize the end of telegram
MAX_TX_BUFFER=2     ;1...5
MAX_RX_BUFFER=2     ;1...20
SIZE_RX_BUFFER=100 ;1...2048 longest expected telegram length + 15
characters
XON_VAL=17          ;0...255 XON character (decimal)
XOFF_VAL=19         ;0...255 XOFF character (decimal)
                   ;if NON_VAL=0 and XOFF_VAL=0 then XON/XOFF-protocol
                    ;is disabled (pure serial communication)
DSR_LINE=0          ;0=DSR line not connected, 1=DSR line must be high
```

Parameter

Parameter	Description
CHAR_TIMEOUT	The maximum interval between two characters. If this interval is exceeded, the Xon/Xoff protocol interprets this as the end of the data block.
MAX_TX_BUFFER	Maximum number of output buffers
MAX_RX_BUFFER	Maximum number of receive buffers
SIZE_RX_BUFFER	The size of a receive buffer in bytes. The size should be defined as the length of the longest expected data block plus 15 characters. The additional 15 characters are to account for the delay that arises before the transmitter reacts to the signal from the receiver warning that its receive buffer is threatening to overflow. (It is not necessary to define a size for the output buffers, as the size automatically adapts to the contents.)

Parameter	Description
XON_VAL XOFF_VAL	XOFF_VAL: This character is used by the transmitter to signal that it must interrupt the transmission because the receive buffer is threatening to overflow. XON_VAL: This character is used by the transmitter to signal that it can resume transmission once again. **Note:** The Xon/Xoff procedure can only be used if the XON_VAL and XOFF_VAL characters are not contained in the data to be transmitted! If XON_VAL and XOFF_VAL are both set to 0, the Xon/Xoff protocol is deactivated. The data are then transmitted without being checked by the receive buffer.
DSR_LINE	The DSR line is a hardware line that can be used in the communication between the controllers. This line is not always present – this depends on the specific case. If the DSR line is present, data can only be exchanged if its state is "high". ■ 0 = The robot controller does not monitor the state of the DSR line. This option must be selected if the DSR line is not present. ■ 1 = The robot controller monitors the state of the DSR line.

4.6 Displaying sent/received data with Telnet

Data that have been sent and received via a serial interface can be displayed using Telnet.

Precondition

TESTPRINT must be set to 1 in the file SERIAL.INI:

```
[TEST]
...
TESTPRINT=1
```

Procedure

1. Click on the Windows **Start** button.
2. Select the menu option **Run...**.
3. In the **Open** box, enter "telnet 192.0.1.1".
4. Click on **OK**.

   The Telnet window is opened. It displays the values that are sent and received.

Example

The Telnet window indicates that the COM3 port has been used to send the integer value 54321 and receive the integer value 12321 (TX = output buffer, RX = receive buffer).

```
-> SER OPEN
SER [3]-TX: 54321
SER [3]-RX: 12321
```

4.7 3964R procedure

The 3964R procedure is an asynchronous, bit-serial transmission procedure.

When the procedure receives data from the partner controller, it deposits them in receive buffers (RX_BUFFER) and passes the buffers on to the interpreter for further processing. Data to be sent to the partner controller lie ready in output buffers (TX_BUFFER). The interpreter forwards the data to the procedure which sends them to the partner controller.

The procedure repeats transmissions, if required, and signals errors to the interpreter.

The control characters for the 3964R procedure are taken from the DIN 66003 standard for 7-bit code. The transmission itself, however, uses an 8-bit character length with bit 7=0. For the purposes of data protection, a block check character (BCC) is sent at the end of each data block.

The block check character, BCC, is the block parity of the information bits of all data bytes in a transmitted or received block (EXCLUSIVE OR operation). This block parity is defined in the PARITY section of the file SERIAL.INI during configuration of the serial interface (>>> 4.3 "Configuring the serial interface" page 11). It begins with the first user data byte after the connection has been made and ends after the DLE ETX character on termination of the connection.

4.7.1 3964R procedure sends data

Overview

Robot controller		Partner controller
STX	----->	
	<-----	DLE
Character 1	----->	
Character 2	----->	
…	----->	
…	----->	
Character n	----->	
DLE	----->	
ETX	----->	
BCC	----->	
	<-----	DLE

Description

Error-free process:

1. The 3964R procedure sends the character STX in order to make a connection.
2. The partner controller responds within the acknowledgement delay time QUITT_TIMEOUT with the character DLE. The connection is established and the procedure commences transmission.
3. The data from the output buffer are sent at the defined transmission speed to the partner controller.

   The partner controller monitors the interval between two characters. This interval must not exceed the character delay time CHAR_TIMEOUT configured in the robot controller.

   Each DLE character contained in the output buffer is sent twice. Reason: DLE is actually a control character and in this way it is recognized by the receiving controller as an information character.

4. When the output buffer is empty, the procedure sends the characters DLE, ETX and BCC (in this order) as an end identifier and waits for an acknowledgement character.
5. The partner controller sends the character DLE within the acknowledgement delay time QUITT_TIMEOUT. The data block has been received without errors and the transmission is terminated.

Error establishing connection:

In the following cases, no connection is made after the 3964R procedure has sent the character STX:

- The partner controller responds with the character NAK.
- The partner controller responds with any character or character string other than DLE.
- QUITT_TIMEOUT elapses without a reaction from the partner controller.

After 6 failed attempts to make a connection, the procedure cancels the process, signals the error to the interpreter and sends the character NAK to the partner controller.

Error receiving:

In the following cases, the data block from the partner controller has not been received correctly:

- Once the procedure has sent the end identifier DLE, ETX, BCC, the partner controller responds with the character NAK.
- Once the procedure has sent the end identifier DLE, ETX, BCC, the partner controller responds with any character or character string other than DLE.
- Once the procedure has sent the end identifier DLE, ETX, BCC, the acknowledgement delay time QUITT_TIMEOUT elapses without a reaction from the partner controller.

After 6 failed attempts to send the data block, the procedure cancels the process, signals the error to the interpreter and sends the character NAK to the partner controller.

Partner controller sends NAK:

If the partner controller sends the character NAK while a transmission is in progress, the procedure terminates the block and recommences connection. In the case of a character other than NAK, the procedure waits first for the character delay time CHAR_TIMEOUT to elapse. It then sends the character NAK to bring the partner controller to the rest state. The procedure then recommences connection with STX.

4.7.2 3964R procedure receives data

Error-free process:

When there is no transmission request to process, the 3964R procedure waits in the rest state for a connection to be established by the partner controller. The establishment of the connection is initiated when the procedure receives the character STX from the partner controller.

1. The procedure receives the character STX from the partner controller.
2. If there is an empty receive buffer available to the procedure, it responds with the character DLE.
3. The procedure receives characters and stores them in the receive buffer.

   If the procedure receives the character DLE twice in succession, it only writes it to the receive buffer once. Reason: DLE is actually a control char-

acter. The partner controller sends DLE twice so that it is can be identified by the procedure as a character.

When the receive buffer is full, the procedure transfers the full buffer to the interpreter before the partner controller terminates the connection. Any further characters it receives are deposited in a further receive buffer.

4. If the procedure identifies the characters DLE, ETX and BCC (in this order), it terminates the reception. It compares the received block check character BCC with the internally generated block parity.
5. If the block check character is correct and no receiving errors have occurred, the procedure sends the character DLE. It then transfers the contents of the receive buffer to the interpreter and returns to the rest state.

Error:

If the procedure in the rest state receives any character (except STX), it waits for the character delay time CHAR_TIMEOUT to elapse and then sends the character NAK. This error is signaled to the interpreter.

If, during the transmission phase, the character delay time CHAR_TIMEOUT elapses without the procedure receiving another character, it sends the character NAK to the partner controller and signals the error to the interpreter.

If errors occur during the transmission phase (e.g. lost characters, parity errors), the procedure continues receiving until the partner controller terminates the connection. The procedure then sends the character NAK to the partner controller. The procedure then waits for a repetition.

If the block check character BCC of the end label is incorrect, the procedure sends the character NAK to the partner controller and waits for it to re-establish a connection. If the block still cannot be received without errors after 6 attempts, or if the partner controller does not transmit within the wait time TRANS_TIMEOUT, the procedure cancels the reception and signals the error to the interpreter.

4.7.3 Initialization conflict

Initialization conflict means: both controllers are simultaneously attempting to establish a connection. Both send the character STX. Initialization conflicts are avoided by means of the priority. The priority determines which controller can transmit first:

1. The controller with the lower priority withdraws its transmission request and sends the character DLE.
2. The controller with the higher priority then transmits its data.

The priority is defined in the file SERIAL.INI. The possible values are:

- 1 = higher priority
- 0 = lower priority

Different priorities must be assigned to each of the controllers. If, for example, the partner controller has priority 0, the robot controller must have priority 1.

5 Configuring the external modules

Communication via external modules is not possible in KSS 7.0.

The complete configuration of an external module is not described below. This is module-specific. Only settings relevant to CREAD/CWRITE are described.

File

The external modules are declared in the file $CUSTOM.DAT.

```
DECL EXT_MOD_T $EXT_MOD_x={O_FILE[] " ",OPTION 0}
```

Parameter

O_FILE[]

The path and file name of the O file must be entered in the square brackets, e.g. DRIVERS\tcpdrv.o.

O files are always situated in the directory C:\KRC:\ROBOTER. This part of the path does not need to be specified.

OPTION (bit 0)

Bit 0 of OPTION defines what happens to an external module in the case of a CCLOSE statement. The setting is called "Force unload".

Value	Description
Bit 0 = 1	"Force unload" is active. CCLOSE closes the channel to the module, the module is unloaded and the module environment is destroyed.
Bit 0 = 0	"Force unload" is not active. CCLOSE closes the channel to the module. The module remains loaded and the module environment is not destroyed. In the case of a COPEN statement, the module does not need to be reloaded.

OPTION (bit 1)

Bit 1 of OPTION is only relevant for external modules of type LD_EXT_OBJ with "Force unload" deactivated. The setting is called "Leave data".

Value	Description
Bit 1 = 1	"Leave data" is active. In the case of a CCLOSE statement, all data that have been received, but not yet read, are retained. When the channel is reopened, these data can be read by CREAD.
Bit 1 = 0	"Leave data" is not active. CCLOSE deletes all data that have been received, but not yet read.

6 Programming

6.1 Programming overview

All statements (except CHANNEL) can be used in both SRC and SUB files.

Statements can be interrupted by interrupt programs. If attempts are made to access channels within the interrupt program, this access can only be interrupted by other interrupt programs.

The table shows which statements are relevant for which channels:

Statement	Serial interface	External module Type LD_EXT_OBJ	External module Type LD_EXT_FCT	Command channel
CHANNEL	+	+	+	-
CCOPEN	+	+	+	-
CREAD	+	+	-	-
CWRITE	+	+	+	+
CCLOSE	+	+	+	-
CIOCTL	-	+	+	-
SREAD	Statement does not refer to a channel			
SWRITE	Statement does not refer to a channel			
CAST_TO	Statement does not refer to a channel			
CAST_FROM	Statement does not refer to a channel			

Communication via external modules is not possible in KSS 7.0.

6.2 Symbols and fonts

The following symbols and fonts are used in syntax descriptions:

Description	Example
KRL code: ■ Courier font ■ Upper-case characters	`GLOBAL; ANIN ON; OFFSET`
Elements that must be replaced by program-specific entries: ■ Upper- and lower-case characters ■ Italics	*Distance*; *Time*; *Format*
Optional elements: ■ In angle brackets	< ... >
Elements that are mutually exclusive: ■ Separated by the "|" symbol	`IN |OUT`

6.3 CHANNEL

It is not necessary to program the CHANNEL statement. It is already predefined for all serial interfaces and external modules in the file $CUSTOM.DAT. The statement is nonetheless explained here for the sake of understanding.

Function

CHANNEL links a predefined structure variable to a name.

This is necessary if an interface or external module is to be addressed, as the predefined structure variables cannot be addressed directly. For example, the serial interface COM3 cannot be addressed directly by means of $PSER3.

Syntax

`CHANNEL` *:Channel_Name :Interface_Name Structure_Variable*

Description

Parameter	Description
Channel_Name	Name for the serial interface or external module
Interface_Name	Predefined signal variable
Structure_Variable	Structure variable for the serial interface or external module $PSER_x contains the parameters of the serial interface (e.g. baud rate). $PSER_x is not analyzed, however, as the parameters of the serial interface are also contained in the file SERIAL.INI and are analyzed there. The structure variable for the external module, $EXT_MOD_x, is analyzed (unlike $SPSER_x).

Example

CHANNEL statement for serial interface COM3:

```
CHANNEL :SER_3 :SER_3 $PSER3
```

6.4 COPEN

Function

Before a channel can be used for communication with CREAD/CWRITE, it must first be opened with COPEN.

Exception: the command channel is always open and does not need to be opened or closed.

If a channel has been opened in an SRC program, it must be closed again before it can be opened in a SUB program. Likewise, a channel that has been opened in a SUB program must be closed again before it can be opened in an SRC program.

If a channel that is already open is opened again by the same interpreter, the same handle is returned again.

Syntax

`COPEN` (*:Channel_Name, Handle*)

Description

Parameter	Description
Channel_Name	Channel name declared using the CHANNEL statement
Handle	Type: INT Variable to which feedback signal is sent about whether the channel has been opened. ■ *Handle* > 0 The channel has been opened. The management number of the open channel has been returned. ■ *Handle* = 0 Error. The channel could not be opened. ■ *Handle* < 0 Only possible with external modules. The module was successfully loaded, but could not be initialized. The number specifies the cause of the initialization error which is described in the driver documentation.

The variable *Handle* must be declared. It is useful to declare and initialize it in the file $CONFIG.DAT as follows:

```
INT HANDLE = 0
```

Declaring the variable in $CONFIG.DAT makes it available in all programs.

6.5 CREAD

Function

CREAD reads data from an opened serial interface or from a loaded external module of type LD_EXT_OBJ.

It is not possible to read from external modules of type LD_EXT_FCT or from the command channel.

- Data of type INT must be in little endian format and be preceded by a sign.
- Data of type REAL must be in 32-bit representation in IEEE 754 standard format.

Syntax

`CREAD (`*Handle, State, Mode,* `TIMEOUT, OFFSET,` *Format, Var1 <, ...,Var10>*`)`

Description

Parameter	Description
Handle	Type: INT Variable transferred by COPEN to CREAD identifying the channel
State	Type: STATE_T State that is automatically returned to CREAD (>>> 6.14 ""State" variable" page 33)
Mode	Type: MODUS_T Read mode (see section "Read mode" below). The mode must be initialized.

Parameter	Description
TIMEOUT	Type: REAL The timeout is only relevant for read mode ABS. The timeout defines how many seconds to wait for data. Once this time has elapsed, waiting is terminated. Timeout values less than 0.0 or greater than 60.0 will be rejected.
OFFSET	Type: INT The position in the received text string at which CREAD commences reading. If reading is to start from the beginning, the offset must be set to 0. Examples: ■ OFFSET=0: CREAD commences reading at the first position. ■ OFFSET=2: CREAD commences reading at the third position. Positions 1 and 2 are ignored. The offset is incremented during reading. If, in the case of another CREAD statement, reading is to start again at the first position, then the offset must be set to 0 before this statement. Otherwise, the incremented offset of the previous statement will be accepted.
Format	Type: CHAR[] The received data must be formatted in order for them to be able to be written to the variable *Var*. A format must be specified for every *Var* variable. (>>> 6.15 ""Format" variable" page 35)
Var	Variables that are filled with the received data. A maximum of 10 variables per statement are possible.

Read mode

The read mode is determined by a variable of type MODUS_T. MODUS_T is a predefined enumeration type:

ENUM MODUS_T SYNC, ASYNC, ABS, COND, SEQ

For CREAD, only ABS, COND and SEQ are relevant:

Value	Description
ABS	CREAD waits until the channel makes data available for reading or until waiting is aborted by timeout.

Value	Description
COND	CREAD checks whether data are present: ■ If data are present, then they are read. ■ If no data are present, then the system does not wait. The CREAD statement is deemed to have been completed. COND is useful if the CREAD statement is triggered by an interrupt when data are available for reading (see section "Reading with interrupts" below).
SEQ	SEQ is only possible when reading from a serial interface. If a CREAD statement cannot completely read a data set because it is too long, the reading can be completed using SEQ. For this purpose, the first CREAD statement in the program (with ABS or COND) must be followed by another (with SEQ). If required, several SEQ statements can be used in succession.

Reading with interrupts

A system variable is monitored to determine whether data are available for reading:

- $DATA_SERx for the serial interface COMx
- $DATA_LD_EXT_OBJx for the external module LD_EXT_OBJx

When data are received, the system variable is incremented by the channel driver. The data can then be read with an interrupt program.

The variables are initialized with 0 when a warm restart is carried out or when a channel is opened or closed.

In the case of external modules of type LD_EXT_OBJ: if the option "Leave data" is activated, the variable is not reset.

Example interrupt procedure:

Main program with interrupt declaration:

```
INTERRUPT DECL 10 WHEN $DATA_SER3<>0 DO SER_INT ()
INTERRUPT ON 10
...
```

Interrupt program:

```
DEF SER_INT ()
DECL MODUS_T MODE
...
INTERRUPT OFF 10
WHILE ($DATA_SER<>0)
 ...
 MODE=#COND
 OFFSET=0
 CREAD (HANDLE,..., MODE,...)
 ...
ENDWHILE
INTERRUPT ON 10
END
```

6.6 CWRITE

Function

CWRITE writes data to an opened serial interface or to a loaded external module of type LD_EXT_OBJ.

In a loaded external module of type LD_EXT_FCT, CWRITE calls a function.

CWRITE writes commands to the command channel.

CWRITE triggers an advance run stop.

Syntax

`CWRITE (`*Handle* or `$CMD`, *State, Mode, Format, Var1 <, ..., Var10>*`)`

Description

Parameter	Description
Handle/`$CMD`	Type: INT *Handle*: Variable transferred by COPEN to CWRITE identifying the channel `$CMD`: Predefined variable for writing to the command channel
State	Type: STATE_T State that is automatically returned to CWRITE (>>> 6.14 ""State" variable" page 33)
Mode	Type: MODUS_T Write mode (see section "Write mode" below). The mode must be initialized.
Format	Type: CHAR[] The *Var* variables must be converted into a text string before they can be written to the channel. *Format* defines the format of the text that is to be generated. A format must be specified for every *Var* variable. (>>> 6.15 ""Format" variable" page 35) In the case of external modules of type LD_EXT_FCT: Instead of a format, the name of the function to be called is specified at this point.
Var	Variables whose data are written to the channel. A maximum of 10 variables per statement are possible. In the case of external modules of type LD_EXT_FCT: The *Var* variables contain the transfer parameters for the function called with *Format*.

Write mode

The write mode is determined by a variable of type MODUS_T. MODUS_T is a predefined enumeration type:

ENUM MODUS_T SYNC, ASYNC, ABS, COND, SEQ

For CWRITE, only SYNC and ASYNC are relevant:

Value	Description
SYNC	The statement is deemed to have been executed once the partner controller has fetched the transferred data from the receive buffer.

Value	Description
ASYNC	When writing to external modules of type LD_EXT_FCT, ASYNC mode is not allowed! The following applies for all other channels: The statement is deemed to have been executed once the data have arrived in the receive buffer of the partner controller. ■ Advantage over SYNC: The program is executed more quickly. ■ Disadvantage compared with SYNC: Data can be lost.
Other value	If the mode has a value other than SYNC or ASYNC, writing is carried out by default in SYNC mode.

6.7 CCLOSE

Function

CCLOSE closes the serial interface and deletes all of the data that are waiting to be read. Once the interface has been closed by means of CCLOSE, the variable *Handle* can no longer be used for CREAD, CWRITE or CCLOSE. The value of the variable is not changed, however.

An open serial interface is automatically closed if a program is deselected or closed.

CCLOSE closes the channel to the external module. Whether the module is unloaded and whether the data waiting to be read are deleted depends on the configuration (>>> 5 "Configuring the external modules" page 19).

CCLOSE is not relevant for the command channel, as this channel is always open.

CCLOSE triggers an advance run stop.

If an attempt is made, using CCLOSE, to close a channel that has already been closed, the state #CMD_ABORT is returned.

Syntax

`CCLOSE` *(Handle, State)*

Description

Parameter	Description
Handle	Type: INT Handle variable transferred by COPEN to CCLOSE identifying the channel
State	Type: STATE_T State that is automatically returned to CCLOSE (>>> 6.14 ""State" variable" page 33)

6.8 CIOCTL

Function

CIOCTL is only relevant for external objects.

- CIOCTL can be used to transfer any data to an external object, e.g. configuration data to change a file name.
- CIOCTL can request any data of an external object.

CIOCTL is used to transfer data in addition to the data communicated using CREAD/CWRITE, e.g. to request a detailed error message following a failed CREAD or CWRITE statement. The CIOCTL statement can not be used in-

stead of CREAD/CWRITE because, although it can transfer the same data, it cannot format them.

CIOCTL can also be called by the command interpreter.

CIOCTL always has a return value:

Return value (Type INT)	Description
0	CIOCTL was executed successfully.
1	CIOCTL was not executed successfully. Cause: The channel is closed.
2	CIOCTL was not executed successfully. Cause: CIOCTL was called by a different interpreter than COPEN. Example: COPEN was called by S_INT; CIOCTL was called by R_INT. If CIOCTL is called by the command interpreter, the statement is always executed, irrespective of which interpreter called COPEN.
3	CIOCTL was not executed successfully. Cause: Invalid request number
>0	Error number returned by the external module.

Syntax

`CIOCTL` *(Handle, Request, Argument, String, Retval)*

Description

Parameter	Description
Handle	Type: INT Variable transferred by COPEN to CIOCTL identifying the channel
Request	Type: INT Request number transferred by value to the external module. Only request numbers greater than 0 are permissible. Request numbers can have a wide range of different functions; for example, a request number can start a specific program. The meaning of the request number is module-specific.
Argument	Type: INT Integer data transferred to the external module.
String	Type: CHAR[] Character array transferred to the external module. Maximum 128 array elements.
Retval	Type: INT Integer value transferred by reference to the external module. The external module can modify the value.

6.9 SREAD

Function

SREAD has a similar function and syntax to CREAD. Unlike CREAD, however, SREAD does not read data from a channel, but from a CHAR array.

SREAD can be combined in programs with CREAD. Advantages:

- CREAD can be restricted to reading data from the channel. More complex formatting tasks can be carried out by SREAD. This makes programs more flexible.
- CREAD can process a maximum of 10 variables. Combination with several SREAD statements makes it possible to read the data of more than 10 variables.

Syntax

`SREAD` *(String, State,* `OFFSET`*, Format, Var1 <, ..., Var10>)*

Description

Parameter	Description
String	Type: CHAR[] This string is read, formatted and written to the *Var* variables.
State	Type: STATE_T State that is automatically returned to SREAD (>>> 6.14 ""State" variable" page 33)
OFFSET	Type: INT The position in the string at which SREAD commences reading. If reading is to start from the beginning, the offset must be set to 0. Examples: `OFFSET=0`: SREAD commences reading at the first position. `OFFSET=2`: SREAD commences reading at the third position. Positions 1 and 2 are ignored. The offset is incremented during reading. If, in the case of another SREAD statement, reading is to start again at the first position, then the offset must be set to 0 before this statement. Otherwise, the incremented offset of the previous statement will be accepted.
Format	Type: CHAR[] The formats into which the string is converted so that it can be written to the *Var* variables. A format must be specified for every *Var* variable. (>>> 6.15 ""Format" variable" page 35)
Var	Variables to which the disassembled and formatted string is written. A maximum of 10 variables per statement are possible.

6.10 SWRITE

Function

SWRITE has a similar function and syntax to CWRITE. Unlike CWRITE, however, SWRITE does not write data to a channel, but to a CHAR array.

SWRITE can be combined in programs with CWRITE. Advantages:

- CWRITE can be restricted to writing data to the channel. More complex formatting tasks can be carried out by SWRITE. This makes programs more flexible.

- CWRITE can process a maximum of 10 variables. Combination with several SWRITE statements makes it possible to write the data of more than 10 variables.

SWRITE triggers an advance run stop.

This document contains an example of a program combining CWRITE and SWRITE.
(>>> 7.2 "Serial interface: outputting the date" page 42)

Syntax

`SWRITE` *(String, State,* `OFFSET`*, Format, Var1 <, ..., Var10>)*

Description

Parameter	Description
String	Type: CHAR[] The formatted contents of the *Var* variables are written to the string.
State	Type: STATE_T State that is automatically returned to SWRITE (>>> 6.14 ""State" variable" page 33)
`OFFSET`	Type: INT The position in the string at which SWRITE commences writing. If writing is to start from the beginning, the offset must be set to 0. Examples: `OFFSET=0`: SWRITE commences writing at the first position. `OFFSET=2`: SWRITE commences writing at the third position. Positions 1 and 2 are ignored. The offset is incremented during writing. If, in the case of another SWRITE statement, writing is to start again at the first position, then the offset must be set to 0 before this statement. Otherwise, the incremented offset of the previous statement will be accepted.
Format	Type: CHAR[] Converts the *Var* variables before they are written to the string. A format must be specified for every *Var* variable (>>> 6.15 ""Format" variable" page 35).
Var	Variables whose data are written to the string. A maximum of 10 variables per statement are possible.

6.11 CAST_TO

Function

CAST_TO makes it possible to process up to 4 KB of data with a single CWRITE statement. CAST_TO groups individual variables together as a single buffer. CWRITE then writes this buffer to the channel.

Maximum buffer size: 4 KB (= 4096 bytes). If the quantity of data is so great that the maximum buffer size is insufficient, several successive CWRITE statements must be used.

If the buffer is declared in the data list, no initial value may be set! Reason: The initial value is overwritten by the current value. The current value can be up to 4 KB and thus exceeds the maximum permissible length of a KRL line.
CORRECT: `DECL CHAR mybuffer[4096]`
INCORRECT: `DECL CHAR mybuffer[4096]=" "`

CAST_TO does not trigger an advance run stop If, however, variables are processed that do trigger an advance run stop, then an advance run stop is triggered indirectly.

Conversion character

If a buffer that has been generated with CAST_TO is transferred using CWRITE, only the following conversion characters are permissible in the CWRITE statement:

- r (= raw data format)
- s (= string format)

r has the following advantages over s:

- If the character 0 is transferred, s interprets this as the end of the string. This problem does not occur with r.
- The offset counts in bytes. If CREAD reads the data with r, i.e. binary, the number of values that have already been transferred can easily be calculated using the offset.

Syntax

`CAST_TO` *(Buffer,* `OFFSET`*, Var1 <, ..., Var10>)*

Description

Parameter	Description
Buffer	Type: CHAR[] Buffer to which the *Var* variables are written
`OFFSET`	Type: INT The position within the buffer (in bytes) after which data are to be written to the buffer. The offset starts with 0. Examples: `OFFSET=0`: Writing commences at the first position. `OFFSET=2`: Writing commences at the third position. Positions 1 and 2 are ignored.
Var	Variables that are written to the buffer. A maximum of 10 variables per statement are possible. In the case of non-initialized variables or array elements, random values are written to the buffer. Since random values can cause problems for the buffer receiver, it is recommended that all variables and array elements should be initialized. The number of bytes written to the buffer by each variable is determined by the data type of the variable. (>>> 6.13 "Permissible data types in CAST statements" page 32) Example: INT Var1, BOOL Var2, REAL Var3 Var1 writes 4 bytes to the buffer; Var2 writes 1 byte; Var3 writes 4 bytes.

6.12 CAST_FROM

Function

CAST_FROM makes it possible to process up to 4 KB of data with a single CREAD statement. If CREAD has read a buffer from a channel, CAST_FROM can break the buffer down into individual variables.

Maximum buffer size: 4 KB (= 4096 bytes). If the quantity of data is so great that the maximum buffer size is insufficient, several successive CREAD statements must be used.

If the buffer is declared in the data list, no initial value may be set! Reason: The initial value is overwritten by the current value. The current value can be up to 4 KB and thus exceeds the maximum permissible length of a KRL line.
CORRECT: `DECL CHAR mybuffer[4096]`
INCORRECT: `DECL CHAR mybuffer[4096]=" "`

CAST_FROM does not trigger an advance run stop If, however, variables are processed that do trigger an advance run stop, then an advance run stop is triggered indirectly.

Syntax

`CAST_FROM` *(Buffer,* `OFFSET`*, Var1 <, ..., Var10>)*

Description

Parameter	Description
Buffer	Type: CHAR[] Buffer whose data are used to fill the *Var* variables
`OFFSET`	Type: INT The position within the buffer (in bytes) after which the data are used to fill *Var*. The offset starts with 0. Examples: `OFFSET=0`: The buffer is used from the first position. `OFFSET=2`: The buffer is used from the third position. Positions 1 and 2 are ignored.
Var	Variables that are filled with the data from the buffer. A maximum of 10 variables per statement are possible. The number of bytes each variable receives from the buffer is determined by its data type. (>>> 6.13 "Permissible data types in CAST statements" page 32) Example: INT Var1, BOOL Var2, REAL Var3 Var1 receives 4 bytes; Var2 receives 1 byte; Var3 receives 4 bytes.

6.13 Permissible data types in CAST statements

	Permissible data type	Size
1.	INT	4 bytes
2.	REAL	4 bytes
3.	BOOL	1 byte
4.	CHAR	1 byte
5.	ENUM	4 bytes
6.	SIGNAL	1 byte

	Permissible data type	Size
7.	FRAME	6*REAL
8.	POS	6*REAL + 2*INT
9.	AXIS	6*REAL
10.	E3POS	6*REAL + 2*INT + 3*REAL
11.	E3AXIS	6*REAL + 3*REAL
12.	E6POS	6*REAL + 2*INT * 6*REAL
13	E6AXIS	6*REAL + 6*REAL

Arrays

The CAST statements can process arrays of the simple data types 1 to 5. CAST statements do not check whether all array elements have been initialized. Non-initialized elements are filled with random values.

Structure types

Structure types other than 7 to 13 may not be used in CAST statements. If other structure types are to be processed, they must be processed one component at a time.

6.14 "State" variable

The state of a statement is automatically returned to the *State* variable. *State* is a variable of type STATE_T and must be declared.

(>>> 6.14.1 "Structure type STATE_T" page 33)

The *State* variable is a component of the following statements:

- CREAD
- CWRITE
- CCLOSE
- SREAD
- SWRITE

6.14.1 Structure type STATE_T

Description

STATE_T is a predefined structure type:

STRUC STATE_T CMD_STAT RET1, INT MSG_NO, INT HITS, INT LENGTH

CMD_STAT RET1

The variable "RET1" is used to signal whether a statement has been executed successfully.

(>>> 6.14.2 "Return values for the variable "RET1"" page 34)

INT MSG_NO

If an error occurs during execution of a statement, the variable MSG_NO contains the error number. MSG_NO is relevant for the following statements:

- CREAD
- CWRITE
- SREAD
- SWRITE

INT HITS

The number of correctly read or written formats. INT HITS is relevant for the following statements:

- CREAD
- CWRITE
- SREAD
- SWRITE

INT LENGTH

The length of the bytes correctly converted to variables in accordance with the format specification. INT LENGTH is relevant for the following statements:

- CREAD
- SREAD

6.14.2 Return values for the variable "RET1"

Description

The variable "RET1" is used to signal whether a statement has been executed successfully. The variable "RET1" is of type CMD_STAT and is a component of the structure type STATE_T.

CMD_STAT is a predefined enumeration type:

ENUM CMD_STAT CMD_OK, CMD_TIMEOUT, DATA_OK, DATA_BLK, DATA_END, CMD_ABORT, CMD_SYN, FMT_ERR

CREAD

RET1 can have the following values for CREAD:

Value	Description
CMD_OK	Only relevant in COND mode: A check has been made to see whether data are present for reading. No data are present for reading, however.
CMD_TIMEOUT	Only relevant in ABS mode: Reading has been aborted because the wait time has been exceeded
DATA_OK	Only relevant when reading from a serial interface: All data have been read.
DATA_BLK	Only relevant when reading from a serial interface: CREAD has received a data set and read the start of the data. The data set was too long, however, to be read through to the end.
DATA_END	When reading from an external module of type LD_EXT_OBJ: All data have been read. When reading from a serial interface: Only relevant in SEQ mode: CREAD has read the data set through to the end.
CMD_ABORT	Reading has been aborted. Possible causes: ■ Error message from channel ■ Error reading the data ■ A mode other than ABS, COND or SEQ has been initialized. ■ The mode has not been initialized.
FMT_ERR	The specified format does not match the variable type of the *Var* variable.

CWRITE

RET1 can have the following values for CWRITE:

Value	Description
CMD_OK	Only relevant when writing to a serial interface: The statement was executed successfully. Unlike with DATA_OK, there are no data ready to be read in the receive buffer.
DATA_OK	When writing to a serial interface: The statement was executed successfully. Furthermore, there are data in the receive buffer than could be read. (This does not necessarily mean that the data have arrived recently in the receive buffer and are up to date.) When writing to an external module: The statement was executed successfully.
CMD_ABORT	The statement was not executed successfully.
CMD_SYN	Only relevant for writing to the command channel: The syntax of the statement is incorrect and the statement cannot be executed.
FMT_ERR	The specified format does not match the variable type of the *Var* variable.

SREAD/SWRITE

RET1 can have the following values for SREAD and SWRITE:

Value	Description
CMD_OK	The statement was executed successfully.
CMD_ABORT	The statement was not executed successfully.
FMT_ERR	The specified format does not match the variable type of the *Var* variable.

CCLOSE

RET1 can have the following values for CCLOSE:

Value	Description
CMD_OK	The statement was executed successfully.
CMD_ABORT	The statement was not executed successfully. Possible causes: ■ The channel is already closed ■ The handle is invalid. ■ The channel has been opened by another process.

6.15 "Format" variable

The *Format* variable is a component of the following statements:

- CREAD
- CWRITE
- SREAD
- SWRITE

6.15.1 "Format" variable for CREAD/SREAD

A format specification for CREAD or SREAD has the following structure:
"*%<W>U*"

Format specifications for arrays have the following structure: "*%<W<.Z>>U*"

Element	Description
W	Maximum number of characters to be read. Optional.
Z	Number of array elements to be written. Optional.
U	Conversion character (>>> 6.15.3 "Conversion characters" page 37)

Format specifications for CREAD/SREAD may not contain formatting characters (e.g. #)!

6.15.2 "Format" variable for CWRITE/SWRITE

A format specification for CWRITE or SWRITE has the following structure: "*%<FW.G>U*"

Element *F*

Formatting characters. Optional.

Multiple formatting characters can be applied to a format.

Formatting character	Description
+	The converted value is always preceded by a sign: positive values with +, negative values with -. If this formatting character is not used, positive values are represented without a sign and negative values are represented with -.
-	The converted value is left-aligned.
#	In format x, every value that is not equal to zero is preceded by 0. In formats e, f and g, a decimal point is always inserted.
0	The converted value is preceded by zeros to make up the minimum width W.
[Space]	In format d, e, f, g or i, the converted argument is preceded by a space.
*	Formats c and r do not always correctly interpret a space in the data string. To avoid misinterpretations, special format specifications with an asterisk (*) can be used for spaces. A format preceded by this character thus no longer corresponds to one of the *Var* variables, but to a space in the data string.

Element *W*

Minimum number of positions to be output. Optional.

Decimal points are counted as helping to make up the minimum number, preceding signs are not. To reach the minimum number, zero bytes (in little endian format) are added at the end. The minimum number may be exceeded where necessary.

Examples:

- VAR=1.56

  ```
  "%+8.4d", VAR
  ```

  Result: _ _ + 1 . 5 6 0 0

- VAR=125.568

  `"%+8.4d", VAR`

  Result: + 1 2 5 . 5 6 8 0

If the width is specified with 0x, this means that the positions to be output are filled with zeros.

Example:

- VAR=1

  `"%+04d", VAR`

  Result: + 0 0 0 1

Compared with the specification without 0:

- `"%+4d", VAR`

  Result: _ _ _ + 1

If no width is specified, the following default widths are used:

- INT, REAL, ENUM: 4 bytes
- BOOL, CHAR: 1 byte

Element *G* Accuracy specification

Format	Description
r in the case of an array	Number of array elements to be represented
e, f	Number of positions to the right of the decimal point
g	Number of significant figures
s	Maximum number of characters represented
All other formats	Number of characters to be represented. If the source value contains more characters, it is truncated or rounded.

Element *U* Conversion character

(>>> 6.15.3 "Conversion characters" page 37)

6.15.3 Conversion characters

The conversion character is a component of the *Format* variable.

The conversion characters listed in the following table are permissible. They correspond to FPRINTF in the programming language C. The characters o, p, n, u and [list] from FPRINTF are not supported. In addition to the characters from FPRINTF, the character r has been introduced:

Format with r	Description
"%x.xr"	Reads or writes a byte sequence of the specified length (e.g. "%2.5r").
"%r"	Reads or writes all available bytes.
"%1r	Reads or writes one byte. Unlike the other conversion characters, the reading of an individual byte must be explicitly specified here with 1.

No distinction is made between upper-case and lower-case letters with conversion characters. Boolean values are output as 0 or 1, ENUM constants as numbers.

Conversion character	Description
c	A single-character argument is expected; this is then processed as an ASCII character. No width W can be specified for formats with the conversion character c.
d	Integer number represented as a decimal.
e	Exponential notation. The argument is converted into the format [-]m.nnnnnnE[+-]xx. The second character string in Format specifies the number of digits to the right of the decimal point.
f	Decimal point representation. An argument is represented in the format [-]mm.nnnnnn. The second character string in Format specifies the number of digits to the right of the decimal point.
g	Formatting is carried out using %e or %f, depending on which format allows the shorter representation.
i	Integer number represented as a decimal.
r	Converts the value of its variable not into ASCII, but into binary notation. With the format %r, the system does not check whether the variable or the array element is initialized.
s	Represents a character string.
x	Hexadecimal notation. Represents the argument in base 16.

6.15.4 Which format for which variable?

Use of the formats is identical for CREAD and SREAD on the one hand and for CWRITE and SWRITE on the other.

Procedure

1. Select the required table below.
2. In the header of the table, search for the data type of the *Var* variable.
   All the permissible formats are indicated by "+" in the column of this data type.

Description

For most data types, there are several permissible formats, e.g. "%s" and "%1.<Z>r" for CHAR arrays. Which format needs to be selected depends on the manner in which the partner controller can send or receive the data.

In the case of arrays, the specification "Z" can be used to define the number of array elements to be taken into consideration. If no value is specified for "Z", all array elements are taken into consideration. The process is aborted, however, at the first non-initialized value. An exception is the format r. In this case, the process is not aborted. Instead, random values are output for variables or array elements that have not been initialized.

CREAD/SREAD

Format	INT	REAL	BOOL	ENUM	CHAR
%d, %i, %x	+	+	+	+	+
%f, %e, %g		+			
%c	+		+	+	+
%1r	+		+	+	+

Format	INT	REAL	BOOL	ENUM	CHAR
%2r	+		+	+	
%4r	+	+	+	+	
%r	+	+	+	+	+

Remarks:

- Data type BOOL

  Every value that is not equal to zero is converted to TRUE
- Data type ENUM

  The system checks whether the value is a permissible ENUM value. If it is not, reading is aborted. The value of the first ENUM constant is 1.
- Format specifications for arrays

  If there are not enough data available to satisfy the format specifications (e.g. "%2.5r", but only 7 bytes are present), nothing is read for this format and the CREAD statement is aborted. The ignored data are still available for reading.
- Format %r

  Only as many bytes as can fit into the variable are read. The rest are still available for reading. If the array is big enough but the number of bytes is not a multiple of the size of an array element, the redundant bytes remain available for reading (for the following format or for the next CREAD statement).

Format	INT array	REAL array	BOOL array	ENUM array	CHAR array
%s					+
%1.<Z>r	+		+	+	+
%2.<Z>r	+		+	+	
%4.<Z>r	+	+	+	+	
%r	+	+	+	+	+
%.<Z>r	+	+	+	+	+

CWRITE/SWRITE

Format	INT	REAL	BOOL	ENUM	CHAR
%d, %i, %x	+		+	+	+
%f, %e, %g	+	+			
%c					+
%1r	+		+	+	+
%2r	+		+	+	
%4r	+	+	+	+	
%r	+	+	+	+	+

Format	INT array	REAL array	BOOL array	ENUM array	CHAR array
%s					+
%1.<Z>r	+		+	+	+
%2.<Z>r	+		+	+	
%4.<Z>r	+	+	+	+	
%r	+	+	+	+	+
%.<Z>r	+	+	+	+	+

6.15.5 Conversion examples

Example 1

The value of the integer variable VI is transferred in decimal and hexadecimal ASCII notation. The first CWRITE statement transfers the characters 123. The second CWRITE statement transfers the characters 7B.

```
INT VI
VI=123
CWRITE(HANDLE,SW_T,MW_T,"%d",VI)
CWRITE(HANDLE,SW_T,MW_T,"%x",VI)
```

Example 2

The value of the integer variable VI is transferred in binary notation:

```
INT VI
VI=123
CWRITE(HANDLE,SW_T,MW_T,"%r",VI)
```

Example 3

All array elements of an array are transferred:

```
REAL VR[10]
CWRITE(HANDLE,SW_T,MW_T,"%r",VR[])
```

With the format "%r", the system does not check whether the variable or the array element is initialized. Random values are transferred for array elements that have not been initialized.

Example 4

The first five array elements of an array are transferred in binary notation:

```
REAL VR[10]
CWRITE(HANDLE,SW_T,MW_T,"%.5r",VR[])
```

20 bytes are transferred in binary notation.

Example 5

All array elements up to the first non-initialized element are transferred:

```
CHAR VS[100]
CWRITE(HANDLE,SW_T,MW_T,"%s",VS[])
```

Example 6

The first 50 array elements are transferred:

```
CHAR VS[100]
CWRITE(HANDLE,SW_T,MW_T,"%s",VS[])
```

Example 7

The internal value of the ENUM constant is transferred in ASCII notation. The corresponding number is transferred:

```
DECL ENUM_TYP E
CWRITE(HANDLE,SW_T,MW_T,"%d",E)
```

Example 8

Two REAL values are transferred with additional text:

```
REAL V1,V2
V1=3.97
V2=-27.3
CWRITE(...,...,...,"value1=%+#07.3f value2=+#06.2f",V1,V2)
```

The following data are transferred:

- value1=+03.970
- value2=-27.30

7 Example programs

7.1 Serial interface: sending and receiving position data

7.1.1 Sending position data

Robot station 1 sends its current position data to robot station 2.

Main program

Synchronous transfer is initialized with MW_T=#SYNC:

```
DEFDAT SEND
;----Declarations----
INT HANDLE
DECL STATE_T SW_T, SC_T
DECL MODUS_T MW_T
ENDDAT
```

```
DEF SEND()
;----Initializations----
MW_T=#SYNC
;----Instructions----
OPEN_P()
WRITE()
CLOSE_P()
END
```

Subprograms

Subprogram for opening the channel:

```
DEF OPEN_P()
COPEN(:SER_3, HANDLE)
IF (HANDLE==0) THEN
  HALT
ENDIF
END
```

Subprogram for sending. Successive formats must be separated by spaces, otherwise misinterpretations are possible:

```
DEF WRITE()
CWRITE (HANDLE, SW_T, MW_T, "%+#10.4f %+#10.4f %+#10.4f %+#10.4f
%+#10.4f %+#10.4f", $POS_ACT.X, $POS_ACT.Y, $POS_ACT.Z, $POS_ACT.A,
$POS_ACT.B, $POS_ACT.C)
IF  (SW_T.RET1<>#CMD_OK) THEN
  HALT
ENDIF
END
```

Subprogram for closing the channel:

```
DEF CLOSE_P()
CCLOSE (HANDLE,SC_T)
IF (SC_T.RET1<>#CMD_OK) THEN
HALT
ENDIF
END
```

7.1.2 Receiving position data

Robot station 2 receives position data from robot station 1 and applies them as its own current position.

Main program

The absolute wait is initialized with MW_T=#ABS. The wait time is limited to 3 seconds with TIMEOUT=3.0:

```
DEFDAT RECEIVE
;----Declarations----
INT HANDLE, OFFSET
REAL TIMEOUT
DECL STATE_T SR_T, SC_T
DECL MODUS_T MR_T
ENDDAT
```

```
DEF RECEIVE()
;----Initializations----
MW_T=#ABS
TIMEOUT=3.0
;----Instructions----
OPEN_P()
READ()
CLOSE_P()
END
```

Subprograms

Subprogram for opening the channel:

```
DEF OPEN_P()
COPEN(:SER_3, HANDLE)
IF (HANDLE==0) THEN
   HALT
ENDIF
END
```

Subprogram for reading. Successive formats must be separated by spaces, otherwise misinterpretations are possible:

```
DEF READ()
OFFSET=0
CREAD (HANDLE, SR_T, MR_T, TIMEOUT, OFFSET, "%f %f %f %f %f %f",
$POS_ACT.X, $POS_ACT.Y, $POS_ACT.Z, $POS_ACT.A, $POS_ACT.B,
$POS_ACT.C)
IF  (SR_T.RET1<>#DATA_OK) THEN
   HALT
ENDIF
END
```

Subprogram for closing the channel:

```
DEF CLOSE_P()
CCLOSE (HANDLE,SC_T)
IF (SC_T.RET1<>#CMD_OK) THEN
   HALT
ENDIF
END
```

7.2 Serial interface: outputting the date

This program is an example of how CWRITE can be combined with SWRITE.

The program can be expanded so that in addition to the year, month and day, the hours, minutes and seconds can also be written to the TEXT string by SWRITE. In this way, CWRITE can transfer a string in which a total of 12 formats are processed (2 per SWRITE statement). Combination of CWRITE with SWRITE thus makes it possible to bypass the the limitation of CWRITE to 10 formats.

$DATE contains the system time and date. DATE is a predefined structure type: DATE INT CSEC, SEC, MIN, HOUR, DAY, MONTH, YEAR.

Data list

```
DEFDAT DATE_PRG
INT OFFSET=0
DECL MODUS_T MOD
DECL CHAR TEXT[30]
TEXT[1]=" "
TEXT[2]=" "
TEXT[3]=" "
TEXT[4]=" "
TEXT[5]=" "
TEXT[6]=" "
TEXT[7]=" "
TEXT[8]=" "
TEXT[9]=" "
TEXT[10]=" "
TEXT[11]=" "
TEXT[12]=" "
TEXT[13]=" "
TEXT[14]=" "
TEXT[15]=" "
TEXT[16]=" "
TEXT[17]=" "
TEXT[18]=" "
TEXT[19]=" "
TEXT[20]=" "
TEXT[21]=" "
TEXT[22]=" "
TEXT[23]=" "
TEXT[24]=" "
TEXT[25]=" "
TEXT[26]=" "
TEXT[27]=" "
TEXT[28]=" "
TEXT[29]=" "
TEXT[30]=" "
ENDDAT
```

The array elements of TEXT have been initialized individually in the data list. In this way, the result can be read after program execution.

SRC file

The first "OFFSET=0" in the statements section fills the TEXT string with spaces. In this way, data from previous program executions can be deleted.

The second "OFFSET=0" prevents the incremented offset from the data list from being applied.

CWRITE writes the TEXT string generated with SWRITE to the channel.

```
DEF DATE_PRG()
;----Initializations----
INT I
DECL DATE DATE_VAR
DATE_VAR=$DATE
MOD=#SYNC
;----Instructions----
OFFSET=0
FOR I= 1 TO 30
TEXT[I]=" "
ENDFOR
OFFSET=0
SWRITE(TEXT[],STAT,OFFSET,"%s%d","YEAR:",DATE_VAR.YEAR)
 OFFSET=10
SWRITE(TEXT[],STAT,OFFSET,"%s%02d","MONTH:",DATE_VAR.MONTH)
OFFSET=20
SWRITE(TEXT[],STAT,OFFSET,"%s%02d","DAY:",DATE_VAR.DAY)
C_OPEN()
CWRITE(HANDLE,STAT,MOD,"%s",TEXT[])
C_CLOSE()
END
```

7.3 External module: calling a function by means of LD_EXT_FCT

Communication via external modules is not possible in KSS 7.0.

COPEN loads the external module. CWRITE calls the function. CLOSE unloads the external module.

```
DEF FUNCTION()
INT HANDLE
DECL CHAR STRING[30]
DECL STATE_T STAT
DECL MODUS_T MOD
COPEN(:LD_EXT_FCT1, HANDLE)
IF HANDLE <=0 THEN
   ERRMSG ("Cannot open ld_ext_fct1")
ENDIF
MOD=#SYNC
STRING[]="test data for ext. mod."
CWRITE(HANDLE,STAT,MOD,"MyOwnFunction",STRING[])
IF STAT.RET<>#DATA_OK THEN
   ERRMSG("Cannot send data to ld_ext_fct1")
ENDIF
CCLOSE(HANDLE,STAT)
IF STAT.RET<>#CMD_OK THEN
   ERRMSG("Cannot close ld_ext_fct1")
ENDIF
END
```

7.4 Command channel: starting, stopping and deselecting a program

The program A6.SRC is to be started, stopped and deselected via the command channel. This is done by means of the following program lines in a SUB file.

```
DECL STATE_T STAT
DECL MODUS_T MODE
MODE=#SYNC
...
;select program A6()
;to start the program the START-button or
;an external start-signal is needed
IF $FLAG[1]==TRUE THEN
  CWRITE($CMD,STAT,MODE,"RUN/R1/A6()")
  $FLAG[1]=FALSE
ENDIF
;stop program A6()
IF $FLAG[2]==TRUE THEN
  CWRITE($CMD,STAT,MODE,"STOP 1")
  $FLAG[2]=FALSE
ENDIF
;cancel program A6()
IF $FLAG[3]==TRUE THEN
  CWRITE($CMD,STAT,MODE,"CANCEL 1")
  $FLAG[3]=FALSE
ENDIF
```

7.5 Combining CREAD/CWRITE with CAST statements

Example 1

The integer values 1 to 1024 are written to the channel using a single CWRITE statement. For this purpose, the values are first written to the buffer BIGSTRING[] using CAST_TO. CWRITE then writes the buffer to the channel.

```
DECL CHAR BIGSTRING[4096]
...
OFFSET=0
FOR n=1 TO 1024
   CAST_TO (BIGSTRING[],OFFSET,N)
ENDFOR
CWRITE(HANDLE,STAT,MODEWRITE,"%1.4096r",BIGSTRING[])
...
```

Example 2

A second robot station receives the data from example 1. It reads the data from the channel and writes them to the buffer BIGSTRING_2[]. The buffer is then written to the VAR variable.

```
DECL CHAR BIGSTRING_2[4096]
...
INT VAR[1024]
...
OFFSET=0
CREAD(HANDLE,STAT,MODEREAD,TIMEOUT,OFFSET,"%1.4096r",BIGSTRING_2[])
...
OFFSET=0
FOR N=1 to 1024
   CAST_FROM (BIGSTRING_2[],OFFSET,VAR[n])
ENDFOR
...
```

8 Appendix

8.1 File paths

The following table specifies the default paths of all the files referred to in this documentation.

File	Directory
$CUSTOM.DAT	C:\KRC\ROBOTER\KRC\STEU\Mada
$CONFIG.DAT	C:\KRC\ROBOTER\KRC\R1\System
HW_INF.INI This file is not available in KSS 7.0.	C:\KRC\ROBOTER\INIT
SERIAL.INI	C:\KRC\ROBOTER\INIT

8.2 Hardware requirements

Cables

- Connectors: RS232 SUB-D
- Standard cables, maximum capacitance 2 500 pF/m. Maximum length 15 m.
- Cables with lower capacitance, e.g. UTP CAT-5 with 55pF/m. Maximum length 45 m.
- Or KR C with network connection (MFC or network card)

Connector pin allocation

The serial interfaces of the KR C2 are designed as 9-contact Sub-D connectors.

Pin	Direction	Name	Description
1	<-----	CD	Carrier Detect (telephone picked up)
2	<-----	RDX	Receive Data (data cable - receive)
3	----->	TDX	Transmit Data (data cable - send)
4	----->	DTR	Data Terminal Ready (handshake output for DSR)
5	------	GND	System Ground (zero potential)
6	<-----	DSR	Data Set Ready (handshake input for DSR)
7	----->	RTS	Request to Send (handshake request for sending)
8	<-----	CTS	Clear to Send (handshake input for RTS from partner)
9	<-----	RI	Ring Indicator (telephone ring tone)

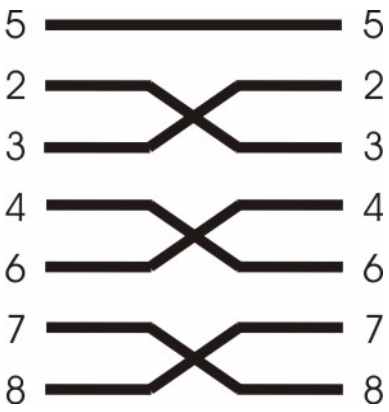

Fig. 8-1: Connector pin assignment of 9-contact Sub-D connector

9 KUKA Service

9.1 Requesting support

Introduction

The KUKA Robot Group documentation offers information on operation and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Faults leading to production downtime are to be reported to the local KUKA subsidiary within one hour of their occurrence.

Information

The following information is required for processing a support request:

- Model and serial number of the robot
- Model and serial number of the controller
- Model and serial number of the linear unit (if applicable)
- Version of the KUKA System Software
- Optional software or modifications
- Archive of the software
- Application used
- Any external axes used
- Description of the problem, duration and frequency of the fault

9.2 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

Marand Precision Engineering Pty. Ltd. (Agency)
153 Keys Road
Moorabbin
Victoria 31 89
Australia
Tel. +61 3 8552-0600
Fax +61 3 8552-0605
robotics@marand.com.au

Austria

KUKA Roboter GmbH
Vertriebsbüro Österreich
Regensburger Strasse 9/1
4020 Linz
Austria
Tel. +43 732 784752
Fax +43 732 793880
office@kuka-roboter.at
www.kuka-roboter.at

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.
Avenida Franz Liszt, 80
Parque Novo Mundo
Jd. Guançã
CEP 02151 900 São Paulo
SP Brazil
Tel. +55 11 69844900
Fax +55 11 62017883
info@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 Flexible Manufacturing Equipment (Shanghai) Co., Ltd.
Shanghai Qingpu Industrial Zone
No. 502 Tianying Rd.
201712 Shanghai
P.R. China
Tel. +86 21 5922-8652
Fax +86 21 5922-8538
Franz.Poeckl@kuka-sha.com.cn
www.kuka.cn

France

KUKA Automatisme + Robotique SAS
Techvallée
6 Avenue du Parc
91140 Villebon s/Yvette
France
Tel. +33 1 6931-6600
Fax +33 1 6931-6601
commercial@kuka.fr
www.kuka.fr

Germany

KUKA Roboter GmbH
Blücherstr. 144
86165 Augsburg
Germany
Tel. +49 821 797-4000
Fax +49 821 797-1616
info@kuka-roboter.de
www.kuka-roboter.de

Hungary

KUKA Robotics Hungaria Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

India

KUKA Robotics, Private Limited
621 Galleria Towers
DLF Phase IV
122 002 Gurgaon
Haryana
India
Tel. +91 124 4148574
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

Korea

KUKA Robot Automation Korea Co. Ltd.
4 Ba 806 Sihwa Ind. Complex
Sung-Gok Dong, Ansan City
Kyunggi Do
425-110
Korea
Tel. +82 31 496-9937 or -9938
Fax +82 31 496-9939
info@kukakorea.com

Malaysia

KUKA Robot Automation Sdn Bhd
South East Asia Regional Office
No. 24, Jalan TPP 1/10
Taman Industri Puchong
47100 Puchong
Selangor
Malaysia
Tel. +60 3 8061-0613 or -0614
Fax +60 3 8061-7386
info@kuka.com.my

Mexico

KUKA de Mexico S. de R.L. de C.V.
Rio San Joaquin #339, Local 5
Colonia Pensil Sur
C.P. 11490 Mexico D.F.
Mexico
Tel. +52 55 5203-8407
Fax +52 55 5203-8148
info@kuka.com.mx

Norway

KUKA Sveiseanlegg + Roboter
Bryggeveien 9
2821 Gjövik
Norway
Tel. +47 61 133422
Fax +47 61 186200
geir.ulsrud@kuka.no

Portugal

KUKA Sistemas de Automatización S.A.
Rua do Alto da Guerra n° 50
Armazém 04
2910 011 Setúbal
Portugal
Tel. +351 265 729780
Fax +351 265 729782
kuka@mail.telepac.pt

Russia

KUKA-VAZ Engineering
Jushnoje Chaussee, 36 VAZ, PTO
445633 Togliatti
Russia
Tel. +7 8482 391249 or 370564
Fax +7 8482 736730
Y.Klychkov@VAZ.RU

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

Spain

KUKA Sistemas de Automatización S.A.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 814-2353
Fax +34 93 814-2950
Comercial@kuka-e.com
www.kuka-e.com

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
Riedstr. 7
8953 Dietikon
Switzerland
Tel. +41 44 74490-90
Fax +41 44 74490-91
info@kuka-roboter.ch
www.kuka-roboter.ch

Taiwan

KUKA Robot Automation Taiwan Co. Ltd.
136, Section 2, Huanjung E. Road
Jungli City, Taoyuan
Taiwan 320
Tel. +886 3 4371902
Fax +886 3 2830023
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

UK

KUKA Automation + Robotics
Hereward Rise
Halesowen
B62 8AN
UK
Tel. +44 121 585-0800
Fax +44 121 585-0900
sales@kuka.co.uk

USA

KUKA Robotics Corp.
22500 Key Drive
Clinton Township
48036 Michigan
USA
Tel. +1 866 8735852
Fax +1 586 5692087
info@kukarobotics.com
www.kukarobotics.com

Index