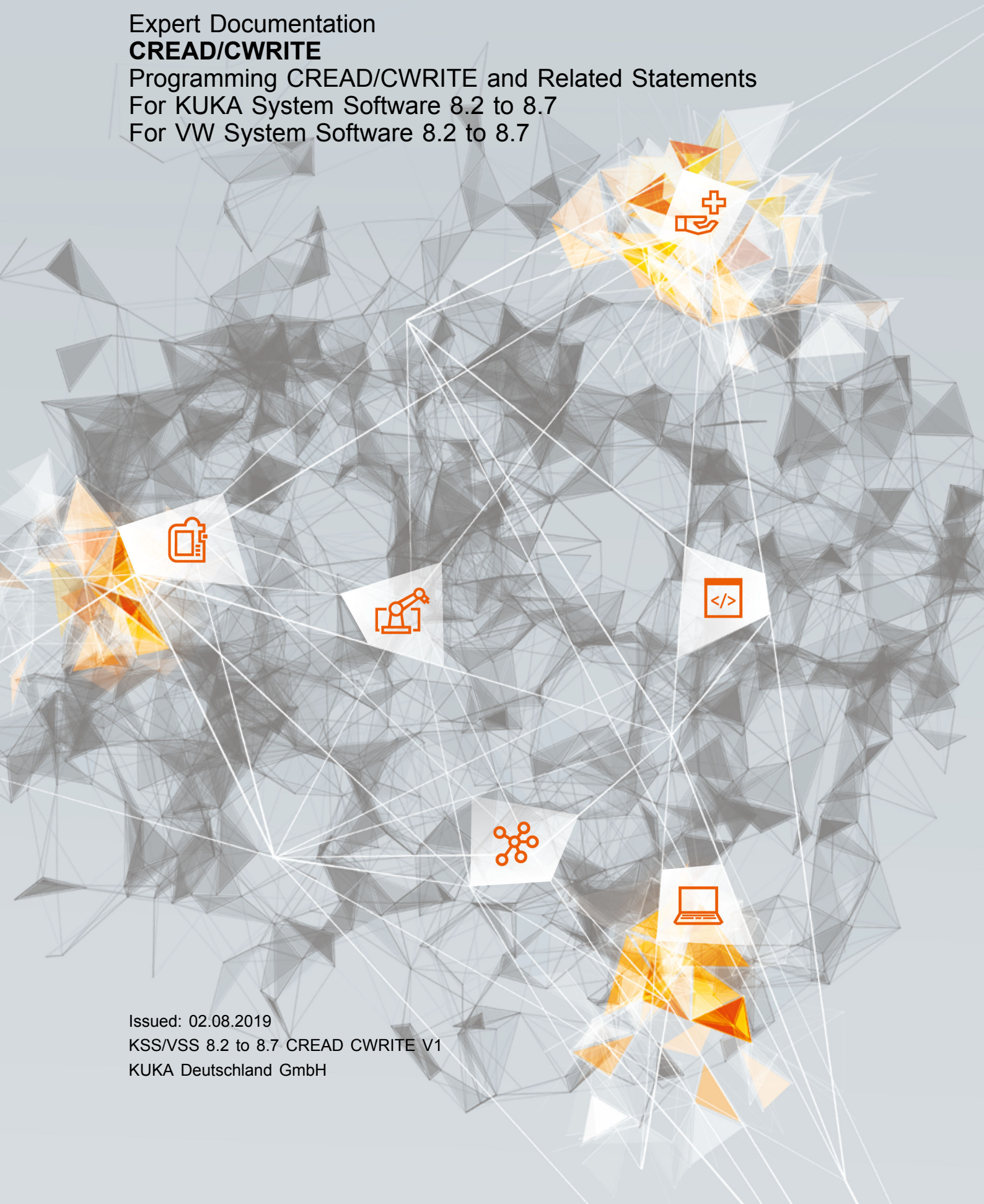
KUKA

Expert Documentation
CREAD/CWRITE
Programming CREAD/CWRITE and Related Statements
For KUKA System Software 8.2 to 8.7
For VW System Software 8.2 to 8.7



Issued: 02.08.2019
KSS/VSS 8.2 to 8.7 CREAD CWRITE V1
KUKA Deutschland GmbH

KUKA Deutschland GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

KIM-PS5-DOC

Translation of the original documentation

Publication:	Pub KSS/VSS 8.2 bis 8.7 CREAD/CWRITE (PDF) en PB3031
Book structure:	KSS/VSS 8.2 bis 8.7 CREAD/CWRITE V1.1 BS1551
Version:	KSS/VSS 8.2 to 8.7 CREAD CWRITE V1

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 robot controller communicates

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

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

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

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

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

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

SAFETY INSTRUCTION
The following procedure must be followed exactly!

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

Notices

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

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

1.4 Terms used

Term	Description
CIFS	Common Internet File System File system for file, print and other server services in networks
Ethernet	Data network technology for local data networks Ethernet allows data to be exchanged between the connected devices in the form of data frames.
Header file	Text file containing declarations and other components of the source text
IP	Internet Protocol The Internet Protocol is used to define subnetworks by means of physical MAC addresses.
KR C	KUKA Robot Control Robot controller
KRL	KUKA Robot Language KUKA robot programming language
Little Endian	Byte sequence for the coding of simple numeric values In Little Endian format, the least significant bit is saved in the first position.
stdio.h	Header file of the programming languages C and C++
TCP	Transmission Control Protocol Protocol of the data exchange between devices of a network. TCP constitutes a virtual channel between 2 sockets in a network connection. Data can be transmitted on this channel in both directions.

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:

- External modules
  (>>> *4.1 "Communication via external modules" Page 11*)
- Command channel $CMD (CWRITE only)
  (>>> *4.2 "Communication via the command channel $CMD" Page 11*)
- Command channel $FCT_CALL (CWRITE only)
  (>>> *4.3 "Communication via the command channel $FCT_CALL" Page 12*)

Example

The robot controller receives position data from another controller (e.g. from a camera system) via a loaded external module. The robot controller uses CREAD to read these position data from the external module.

3 Safety

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

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

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

4 Communication channels

4.1 Communication via external modules

Description

External modules are drivers for interfaces, e.g. for TCP/IP or for Ethernet 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 15*)

Overview

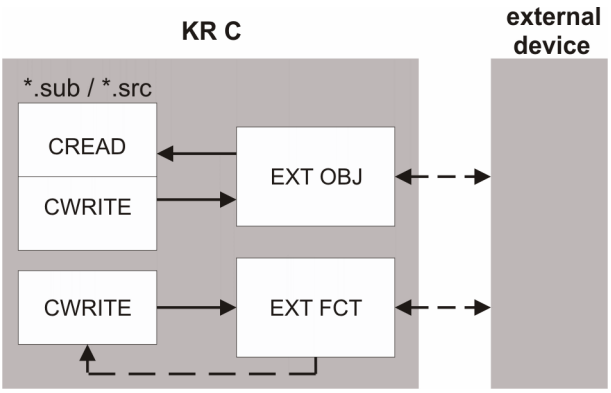


Fig. 4-1: Communication via external modules

4.2 Communication via the command channel $CMD

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

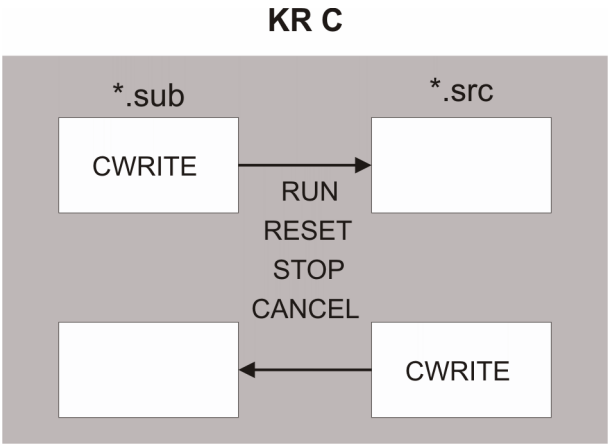


Fig. 4-2: Communication via the command channel $CMD

4.3 Communication via the command channel $FCT_CALL

Description

CWRITE can be used to perform operations on the file system via the command channel. Example: Open a file via the command channel and write a character or a character string into the file.

The file system functions can be called from a robot interpreter program or from a submit interpreter program. It is not permissible to open a file from one interpreter and then access it from the other interpreter, e.g. to write to it or to close the file again.

When a file system function is called, a specific number of parameters are transferred to this function. Each of these functions supplies a return value via CWRITE. The return value indicates whether the function was executed successfully or aborted with an error. In the case of an error, the returned error number can be used to localize the cause.

(>>> *8.1 "Error writing to the command channel $FCT_CALL" Page 65*)

CREAD is not relevant for the command channel.

Configuration

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

Overview

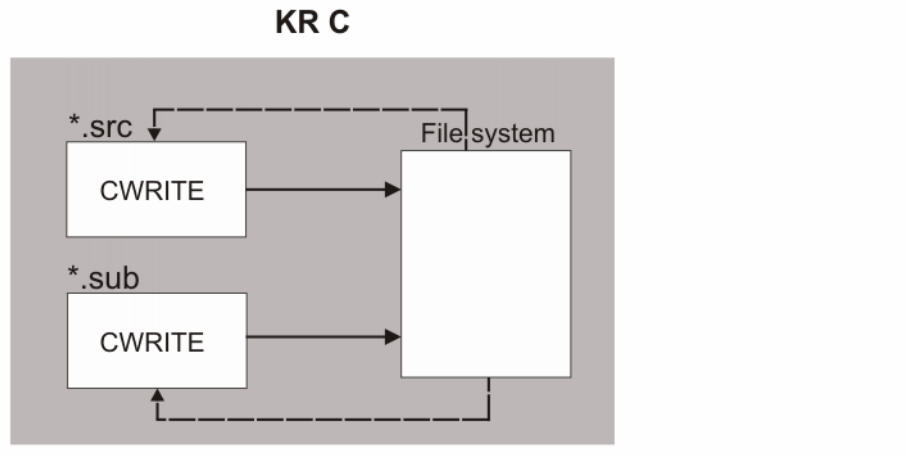


Fig. 4-3: Communication via the command channel $FCT_CALL

5 Configuring the external modules

The complete configuration of an external module is not described in this documentation. 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}
```

Parameters

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	External module Type LD_EXT_OBJ	External module Type LD_EXT_FCT	Command channel $CMD	Command channel $FCT_CALL
CHANNEL	+	+	-	-
COPEN	+	+	-	-
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			

6.2 Symbols and fonts

The following symbols and fonts are used in the syntax descriptions:

Syntax element	Representation
KRL code	• Courier font • Upper-case letters Examples: `GLOBAL`; `ANIN ON`; `OFFSET`
Elements that must be replaced by program-specific entries	• Italics • Upper/lower-case letters Examples: *Distance*; *Time*; *Format*
Optional elements	• In angle brackets Example: <`STEP` *Step_width*>
Elements that are mutually exclusive	• Separated by the “|” symbol Example: `IN`|`OUT`

6.3 CHANNEL

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

Description

CHANNEL links a predefined structure variable to a name.

This is necessary if an external module is to be addressed, as the predefined structure variables cannot be addressed directly. For example, the external module LD_EXT_OBJ1 cannot be addressed directly by means of $EXT_MOD_1.

Syntax

`CHANNEL` `:`*Channel name* `:`*Interface name Structure variable*

Explanation of the syntax

Element	Description
Channel name	Name for the external module
Interface name	Predefined signal variable
Structure variable	Structure variable for the external module ($EXT_MOD_x)

Example

CHANNEL statement for external module 1:

```
CHANNEL :LD_EXT_OBJ1 :LD_EXT_OBJ1 $EXT_MOD_1
```

6.4 COPEN

Description

Before an external module can be used for communication with CREAD/CWRITE, the channel must first be opened with COPEN.

A 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*`)`

Explanation of the syntax

Element	Description
Channel name	Channel name declared using the CHANNEL statement
Handle	Type: INT Handle 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 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

Description

CREAD reads data 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 a 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*>)

Explanation of the syntax

Element	Description
Handle	Type: INT Handle 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 30*)

Element	Description
Mode	Type: MODUS_T The read mode must be initialized. (>>> *6.5.1 "Read mode for CREAD" Page 20*)
TIMEOUT	Type: REAL The timeout is only relevant for read mode ABS. The timeout defines how long to wait for data. • **0.0 … 60.0 s** Example: • TIMEOUT=10.0: If no data have arrived after 10 seconds, the CREAD statement is terminated.
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 array The received data must be formatted in order for them to be able to be written to the variables *Var1* … *Var10*. A format must be specified for every variable. (>>> *6.15 ""Format" variable" Page 33*)
Var1 … *Var10*	Variables to which the received data are written. A maximum of 10 variables per statement are possible.

6.5.1 Read mode for CREAD

Description

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 and COND are relevant:

Value	Description
ABS	CREAD waits until the channel makes data available for reading or until waiting is aborted by timeout.
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. (>>> *"Reading with interrupts" Page 21*)

Reading with interrupts

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

- $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. If the option “Leave data” is activated, the variable is not reset.

Example with interrupt

Main program with interrupt declaration:

```
INTERRUPT DECL 10 WHEN $DATA_LD_EXT_OBJ2<>0 DO OBJ_INT ()
INTERRUPT ON 10
...
```

Interrupt program:

```
DEF OBJ_INT ()
DECL MODUS_T MODE
...
INTERRUPT OFF 10
WHILE ($DATA_LD_EXT_OBJ2<>0)
 ...
 MODE=#COND
 OFFSET=0
 CREAD (HANDLE,..., MODE,...)
 ...
ENDWHILE
INTERRUPT ON 10
END
```

6.6 CWRITE

A CWRITE call in the trigger is not possible. Instead, CWRITE can be used in a trigger subprogram.

Description

CWRITE writes data 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 $CMD.

CWRITE calls a function via the command channel $FCT_CALL.

CWRITE triggers an advance run stop.

Syntax

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

Explanation of the syntax

Element	Description
Handle	Type: INT Handle variable transferred by COPEN to CWRITE identifying the channel to the external module
$CMD	Predefined handle variable for writing to the command channel $CMD
$FCT_CALL	Predefined handle variable for writing to the command channel $FCT_CALL
State	Type: STATE_T State that is automatically returned to CWRITE (>>> *6.14 ""State" variable" Page 30*)
Mode	Type: MODUS_T The write mode must be initialized. (>>> *6.6.1 "Write mode for CWRITE" Page 23*)
Format	Type: CHAR array The variables *Var1* … *Var10* 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 variable. (>>> *6.15 ""Format" variable" Page 33*) In the case of external modules of type LD_EXT_FCT and command channel $FCT_CALL: Instead of a format, the name of the function to be called is specified at this point. (>>> *6.16 "Functions for the command channel $FCT_CALL" Page 39*)
Var1 … *Var10*	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 and command channel $FCT_CALL: The variables *Var1* … *Var10* contain the transfer parameters for the function called with *Format*.

6.6.1 Write mode for CWRITE

Description

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 CWRITE statement is deemed to have been executed once the partner controller has fetched the transferred data from the receive buffer.
ASYNC	When writing to external modules of type LD_EXT_FCT and to the command channel $FCT_CALL, ASYNC mode is not allowed. The following applies for the other channels: The CWRITE 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.
ABS, COND, SEQ	If the mode has a value other than SYNC or ASYNC, writing is carried out by default in SYNC mode.

6.7 CCLOSE

Description

CCLOSE closes the channel to an 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 15*).

A command channel is always open and does not need to be opened or closed.

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*`)`

Explanation of the syntax

Element	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 30*)

6.8 CIOCTL

Description

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 instead 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. The return value can be queried and evaluated in the KRL program.

Syntax

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

Explanation of the syntax

Element	Description
Handle	Type: INT Handle 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 Data transferred to the external module.
String	Type: CHAR array Array transferred to the external module. Maximum 128 array elements.

Element	Description
Retval	Type: INT Return value transferred by reference to the external module. The external module can modify the value. (>>> *"Return values" Page 25*)

Return values

Value	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.

6.9 SREAD

Description

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>)
```

Explanation of the syntax

Element	Description
String	Type: CHAR array This string is read, formatted and written to the variables *Var1* … *Var10*.
State	Type: STATE_T State that is automatically returned to SREAD (>>> *6.14 ""State" variable" Page 30*)
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 array The formats into which the string is converted so that it can be written to the variables *Var1* … *Var10*. A format must be specified for every variable. (>>> *6.15 ""Format" variable" Page 33*)
Var1 … *Var10*	Variables to which the disassembled and formatted string is written. A maximum of 10 variables per statement are possible.

6.10 SWRITE

Description

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.

Syntax

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

Explanation of the syntax

Element	Description
String	Type: CHAR array The formatted contents of the variables *Var1* … *Var10* are written to the string.
State	Type: STATE_T State that is automatically returned to SWRITE (>>> *6.14 ""State" variable" Page 30*)
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 array Converts the variables *Var1* … *Var10* before they are written to the string. A format must be specified for every variable. (>>> *6.15 ""Format" variable" Page 33*)
Var1 … *Var10*	Variables whose data are written to the string. A maximum of 10 variables per statement are possible.

6.11 CAST_TO

Description

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 (= 4,096 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 characters

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>)
```

Explanation of the syntax

Element	Description
Buffer	Type: CHAR array Buffer to which the variables *Var1* … *Var10* 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.
Var1 … *Var10*	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 30*) 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

Description

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 (= 4,096 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*`>)`

Explanation of the syntax

Element	Description
Buffer	Type: CHAR array Buffer whose data are used to fill the variables *Var1* … *Var10*
OFFSET	Type: INT The position within the buffer (in bytes) after which the data are used to write them to the variables *Var1* … *Var10*. 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.
Var1 … *Var10*	Variables that are written to 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 30*) 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

Overview

	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
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. Random values are written to non-initialized elements.

Structure types

Only structure types 7 to 13 may 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 30*)

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 31*)

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 read 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 read mode: Reading has been aborted because the wait time has been exceeded.

Value	Description
DATA_END	Only relevant when reading from an external module of type LD_EXT_OBJ: All data have been read.
CMD_ABORT	Reading has been aborted. Possible causes: • Error message from channel • Error reading the data • A read mode other than ABS or COND has been initialized. • The read mode has not been initialized.
FMT_ERR	The specified format does not match the variable type of the variable *Var1* to *Var10*.

CWRITE

RET1 can have the following values for CWRITE:

Value	Description
DATA_OK	Only relevant for writing to an external module and the command channel $FCT_CALL: The statement was executed successfully.
CMD_ABORT	The statement was not executed successfully.
CMD_SYN	Only relevant for writing to the command channel $CMD: The syntax of the statement is incorrect and the statement cannot be executed.
FMT_ERR	Only relevant for writing to an external module and the command channel $CMD: The specified format does not match the variable type of the variable *Var1* to *Var10*.

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 variable *Var1* to *Var10*.

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 characters (>>> *6.15.3 "Conversion characters" Page 35*)

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.

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.

Character	Description
#	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 variables *Var1* … *Var10*, 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
Other formats	Number of characters to be represented. If the source value contains more characters, it is truncated or rounded.

Element *U*

Conversion characters

(>>> *6.15.3 "Conversion characters" Page 35*)

6.15.3 Conversion characters

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

The conversion characters listed in the tables are permissible. They correspond to the formatting characters of the fprintf() function in the programming language C. The characters o, p, n, u and [list] from fprintf() are not supported.

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.

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.

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**.

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	+		+	+	+
%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

6.16 Functions for the command channel $FCT_CALL

The file system functions described here must be written in lower case. Capitals are not allowed.

Overview

Files system functions for the command channel $FCT_CALL are called using CWRITE. When the call is made, the function parameters must be transferred to CWRITE as *Var1* … *Var10*. Furthermore, the write mode transferred by CWRITE must be #SYNC.

By default, the files accessed by the functions are stored locally in the directory C:\KRC\ROBOTER\UserFiles. The maximum total size of the files stored there is limited to 10 MB, i.e. the total size of all files stored there must not exceed this limit. It is also possible to use an enabled network drive by means of the function krl_mount().

The file system functions can be called from a robot interpreter program or from a submit interpreter program.

It is recommended not to keep files open for an extended period of time. Wherever possible, files should not be opened until they are to be accessed and then closed again immediately afterwards.

When a file is opened, a file-specific handle variable is returned. During programming, it must be ensured that the file handle cannot be lost, e.g. by overwriting the handle variable or by selecting the wrong scope of application for the handle variable. It is then no longer possible to explicitly close this file.
If this problem recurs in continuous operation, this can lead to a situation in which there are no more free file handles available and therefore file operations can no longer be carried out.

Avoid using file system functions in time-critical KRL sections. It is not possible to determine the execution time for file operations since a file system must be operated in the background (in extreme cases as a mounted network drive).
If the implementation of such logging is desired, the time-critical states must first be buffered in KRL variables and then transferred to the file system in a phase that is not time-critical.

Error treatment

Every file system function supplies a return value, via CWRITE, that can be queried and evaluated in the KRL program. The return value indicates whether the function was executed successfully or aborted with an error.

The variable *State*.RET1, which is transferred when CWRITE is called, is used to monitor whether an error has occurred:

- *State*.RET1= #DATA_OK: Function executed successfully
- *State*.RET1= #CMD_ABORT: Function canceled with error

(>>> *6.14 ""State" variable" Page 30*)

In the case of an error, the returned error number can be used to localize the cause. The error number is polled using the variable *State*.MSG_NO. The possible error numbers are described for every function.

6.16.1 krl_mount()

Description

Creation and connection of a mount point

In order to connect to an enabled network drive from KRL, a name must be assigned to the mount point and the complete path to the enabled directory must be known. The user name and encrypted password of the user are also required for connection to the network drive.

The program **khash.exe** in the directory C:\KRC\UTIL\Hash of the robot controller is available for encrypting the password. The program **khash.exe** must be executed on the Windows shell **cmdk.exe** and entry of the password is required.

If, for example, the password for the user is “kuka”, the encryption program on the Windows shell is called as follows: `khash.exe kuka`

The encrypted password that must be used when calling krl_mount() in the KRL program is then displayed on the Windows shell.

Syntax

`krl_mount(CHAR[] Name, CHAR[] Path, CHAR[] User, CHAR[] Password)`

Explanation of the syntax

Element	Description
Name	Name of the mount point to be created; this name is used in the KRL program for access to the network drive. The name must begin with the character “/” and must not contain another “/” character.
Path	Path to the enabled directory to which the mount point should refer. The path must begin with the characters “//” and the individual directories must be separated by the character “/”. The IP address of the host on which the directory has been enabled must be used in the path, e.g. “//160.160.113.23/transfer”. The resolution of host names, e.g. “//pcrc40763/transfer”, is not supported.
User	User name to be used for establishing the connection to the network drive.
Password	Encrypted password of the user.

Example

A mount point to network drive \\160.160.113.23\transfer is established with the name TransferNet. The connection is established for the user “Administrator” with his encrypted password:

```
CWRITE($FCT_CALL, State, Mode, "krl_mount", "/TransferNet",
"//160.160.113.23/transfer", "Administrator",
"04FF94D4B99A1153C8CF3D479089A77AFE")
```

Error numbers

MSG_NO	Description
-2	Operation failed: mount point not (or no longer) available.
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.
-16	At least one function parameter string contains invalid characters, e.g. **\ * ? " < > : , = ;] [ ) (} {**, for paths, etc.

6.16.2 krl_unmount()

Description

Removal of a mount point

This function can be used to terminate the connection to a network drive established with krl_mount().

Syntax

`krl_unmount(CHAR[] Name)`

Explanation of the syntax

Element	Description
Name	Name of the created mount point The name must begin with the character “/” and must not contain another “/” character.

Example

The connection to the mount point TransferNet is terminated:

```
CWRITE($FCT_CALL, State, Mode, "krl_unmount", "/
TransferNet")
```

Error numbers

MSG_NO	Description
-2	Operation failed: invalid user, incorrect password, mount point not (or no longer) available.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.
-16	At least one function parameter string contains invalid characters, e.g. **\ * ? " < > : , = ;] [ ) (} {**, for paths, etc.

6.16.3 krl_fopen()

Description

Opens a file

Depending on the selected mode, a file is opened as a text file or binary file for read access and/or write access. The available modes correspond to those of the fopen() function of the header file stdio.h.

A maximum of 11 files can be open simultaneously with this function in KRL programs.

It is recommended not to keep files open for an extended period of time. Wherever possible, files should not be opened until they are to be accessed and then closed again immediately afterwards.

When a file is opened, a file-specific handle variable is returned. During programming, it must be ensured that the file handle cannot be lost, e.g. by overwriting the handle variable or by selecting the wrong scope of application for the handle variable. It is then no longer possible to explicitly close this file.
If this problem recurs in continuous operation, this can lead to a situation in which there are no more free file handles available and therefore file operations can no longer be carried out.

Syntax

`krl_fopen(CHAR[] Name, CHAR[] Mode, INT Handle <, BUFF_MODE_T Buffering>)`

Explanation of the syntax

Element	Description
Name	File name without path specification or with relative path specification. • File without path specification, e.g. “MyFile.txt”: The file is searched for or created in the directory ROBOTER\UserFiles. • File with path specification, e.g. “TestDir/MyFile.txt”: The file is searched for or created in the directory ROBOTER\UserFiles\TestDir. • File with network path, e.g. “/MyMount/MyFile.txt”: The file is searched for or created in an enabled network directory. The mount point to the network drive with the name MyMount must first have been created by means of krl_mount().
Mode	Mode for opening the file. (>>> *"Modes for opening" Page 43*)
Handle	Handle variable for the open file. When the file is opened, this variable is used to return a file handle. All function calls that refer to the open file must use this handle variable.

Element	Description
Buffering	Buffer mode for the open file. Optional. If no buffer mode is specified, the default value applies. • **#FULL_BUFF**: When writing to the file, data are written to a buffer until the buffer is full or the function krl_fflush() is called. The contents of the buffer are then written to the file. The size of the buffer depends on the CIFS used. • **#LINE_BUFF**: When writing to the file, data are written to a buffer until a line end character is detected, the buffer is full or the function krl_fflush() is called. The contents of the buffer are then written to the file. • **#NO_BUFF**: Without buffering, data are written directly to the file. Default: **#FULL_BUFF**

Modes for opening

Mode	Description
“r”	Open as text file for read access In this mode a file can be opened several times in succession (including by both the robot and submit interpreters simultaneously) without being closed first. A different handle is returned in each case. Each of these handles must be closed at some point, as the file cannot otherwise be opened in a different mode.
“w”	Open as text file for write access In this mode, the contents of the file are deleted on opening. If the file does not exist, it is created.
“a”	Open as text file for write access In this mode, the contents of the file are not deleted on opening; instead, the written values are added at the end of the file. If the file does not exist, it is created.
“rb”	Open as binary file for read access (for behavior, see mode **“r”**)
“wb”	Open as binary file for write access (for behavior, see mode **“w”**)
“ab”	Open as binary file for write access (for behavior, see mode **“a”**)
“r+”	Open as text file for read and write access (for behavior, see mode **“r”**) **Note**: Once the file has been opened, the file pointer points to the start of the file. To ensure that no contents are overwritten when data are written to the file, the file pointer must be placed at the end of the file by means of the function krl_fseek().
“w+”	Open as text file for write and read access (for behavior, see mode **“w”**)
“a+”	Open as text file for write and read access (for behavior, see mode **“a”**)

Mode	Description
“rb+”	Open as binary file for read and write access (for behavior, see mode **“r”**) **Note**: Once the file has been opened, the file pointer points to the start of the file. To ensure that no contents are overwritten when data are written to the file, the file pointer must be placed at the end of the file by means of the function krl_fseek().
“wb+”	Open as binary file for write and read access (for behavior, see mode **“w”**)
“ab+”	Open as binary file for write and read access (for behavior, see mode **“a”**)

Example 1

The file ROBOTER\UserFiles\MyFile.txt is opened as a text file for read access:

```
CWRITE($FCT_CALL, State, Mode, "krl_fopen", "MyFile.txt",
"r", FileHandle)
```

Example 2

The file ROBOTER\UserFiles\test\MyFile.txt is opened as a binary file for write and read access. If the file already exists, the contents of the file are deleted. If the file does not exist, it is created:

```
CWRITE($FCT_CALL, State, Mode, "krl_fopen", "Test/
MyFile.txt", "wb+", FileHandle)
```

Example 3

The file MyFile.txt is opened as a text file for read access without buffering on an enabled network drive. The mount point to the network drive with the name Net1 must first have been created by means of krl_mount():

```
CWRITE($FCT_CALL, State, Mode, "krl_fopen", "/Net1/
MyFile.txt", "r", FileHandle, #NO_BUFF)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed, e.g. because the file was opened outside a KRL program or the mode is “r”, “r+”, “rb” or “rb+”, but the file does not exist.
-6	File already open.
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.
-14	At least one function parameter string is blank.

MSG_NO	Description
-15	At least one function parameter string is too long.
-16	At least one function parameter string contains invalid characters, e.g. **\ * ? " < > : , = ;] [ ) (} {**, for paths, etc.
-30	Direct access to the ROBOTER directory is not permissible.
-31	Invalid absolute path
-32	Mount point not found
-40	File cannot be opened because 11 files are already open.

6.16.4 krl_fclose()

Description

Closes a file

The file is unambiguously determined by the transferred handle. Once the file has been closed, the handle is invalid and cannot be used for further file system operations.

When the file is closed, the contents written to a buffer are written to the file.

Syntax

```
krl_fclose(INT Handle)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file

Example

A file is closed by means of a file handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fclose", FileHandle)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed
-3	There is no file open.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.

6.16.5 krl_fclose_all()

This function is only available in KUKA System Software 8.3.

Description

Closes all files

This function can simultaneously close all open files or all files that have been opened by a specific interpreter using krl_open().

Syntax

```
krl_fclose_all(INT Interpreter)
```

Explanation of the syntax

Element	Description
Interpreter	All files that have been opened by the interpreter specified here using krl_open() are closed. • **0**: Closes all files that have been opened by the submit interpreter. • **1**: Closes all files that have been opened by the robot interpreter. • **-1**: Closes all files that have been opened by the interpreter currently calling krl_fclose_all(). • **-99**: Closes all files that have been opened, regardless of the interpreter.

Example

All files that have been opened by the robot interpreter using krl_open() are closed.

```
CWRITE($FCT_CALL, State, Mode, "krl_fclose_all", 1)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.

6.16.6 krl_feof()

Description

Checks whether file pointer reached the end of the file

This function can be used when reading files. If the file pointer has reached the end of the file, no more characters can be read from the file. The function does not return correct values until at least one read attempt has been made.

Syntax

```
krl_feof(INT Handle, BOOL State)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
State	Variable for the position of the file pointer • **TRUE**: End of file reached. • **FALSE**: End of file not reached.

Example

The result of the check whether the end of the file has been reached is written to the variable “PointerPos”:

```
CWRITE($FCT_CALL, State, Mode, "krl_feof", FileHandle,
PointerPos)
```

Error numbers

MSG_NO	Description
-3	There is no file open.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.

6.16.7 krl_fflush()

Description

Writes all data in the buffer to the file

Syntax

`krl_fflush(INT` *Handle*`)`

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file

Example

All data in the buffer are written to the file transferred by the handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fflush", FileHandle)
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-10	Invalid number of function parameters transferred.

MSG_NO	Description
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.

6.16.8 krl_fgetc()

Description

Reads the next character from a file

Syntax

```
krl_fgetc(INT Handle, CHAR Character)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Character	Variable for the character read from the file

Example

A single character is read from the file transferred by the handle and written to the variable “SingleChar”:

```
CWRITE($FCT_CALL, State, Mode, "krl_fgetc", FileHandle,
SingleChar)
```

Error numbers

MSG_NO	Description
-3	There is no file open.
-4	End of file has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.

6.16.9 krl_fgets()

Description

Reads a character string from a file

This function can be used to read whole lines or up to a defined separator.

- If no separator is specified, characters are read into a buffer until either the end of a line is reached or the maximum possible number of characters has been read.
- If a separator is specified, characters are read into a buffer until either the separator is reached or the maximum possible number of characters has been read.

Syntax

```
krl_fgets(INT Handle, CHAR[] Buffer name, INT Buffer size ,
INT Read<, CHAR Separator>)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Buffer name	Variable for the buffer to which the read character string is written
Buffer size	Maximum possible number of characters (including end of line character) that can be read from the file and written into the buffer
Read	Variable for the number of characters that were actually read from the file
Separator	Separator marking the point up to which characters are to be read. Optional.

Example 1

A whole line is read from the file transferred by the handle, but with a maximum length of 256 characters:

```
CWRITE($FCT_CALL, State, Mode, "krl_fgets", FileHandle,
Buff[], 256, Read)
```

Example 2

Characters are read from the file transferred by the handle up to the character ":", but no more than 100 characters:

```
CWRITE($FCT_CALL, State, Mode, "krl_fgets", FileHandle,
Buff[], 100, Read, ":")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-4	End of file has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.
-41	One value could not be read. The values before this value were read correctly.

6.16.10 krl_fscanf()

Description

Formatted reading of a character string from a file (not a CSV file)

The format to be read is transferred to the function as a character string and written to variables.

The available formats are limited as compared with the standard implementation in the programming language C.

The following formats are permissible:

- “%d”, “%i”, “%u”, “%x”, “%e”, “%f”, “%g”, “%s” and “%c”

The following formats are not permissible:

- “%o”, “%p”, “%n” and lists “[List]”

The following are also not allowed:

- More detailed format definitions for length specifications, signs or fill characters and conversion symbols, e.g. “%6.2.f”, “%-d”, “%+d”, “%#f”, etc.

The krl_fscanf() method cannot be used to read CSV files, as the separator for the scanf formats is a space. In CSV files, however, the separator is a semicolon.

Syntax

`krl_fscanf(INT` *Handle*`,` `CHAR[]` *Format*`,` *Var1* `<,` ... `,` *Var8*`>)`

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Format	Character string with the formats to be read (maximum 8) For each format, a variable *Var1* … *Var8* must be specified, into which the read data can be written. The specified format also defines the data type of the associated variables.
Var1 … *Var8*	Variables to which the read data are written. The number of variables must correspond to the number of formats to be read (maximum 8). Permissible data types: INT, REAL, BOOL, ENUM, CHAR and CHAR array

Example

3 values separated by a space are read from the file transferred by the handle and formatted:

```
CWRITE($FCT_CALL, State, Mode, "krl_fscanf", FileHandle,
"%s %i %c", StringVar[], IntVar, CharVar)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed
-4	End of file has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.

MSG_NO	Description
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.
-42	The format string contains an invalid format statement.
-43	The number of values to be read in the format string does not correspond to the number of transferred parameters.
-44	An invalid format was specified for one parameter.

6.16.11 krl_fputc()

Description

Writes a character to a file

The point in the file at which the character is written depends on the mode used to open the file and the position of the file pointer.

Syntax

```
krl_fputc(INT Handle, CHAR Character)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Character	Character that is written to the file.

Example

The character “x” is written to the file transferred by the handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fputc", FileHandle, "x")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-5	Maximum size of directory C:\KRC\ROBOTER\UserFiles has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.

6.16.12 krl_fputs()

Description

Writes a character string to a file

This function can be used to write a constant character string or variable character string to a file. If a variable is used, each element of the CHAR array must be initialized and contain a valid value.

The point in the file at which the character string is written depends on the mode used to open the file and the position of the file pointer.

Syntax

```
krl_fputs(INT Handle, CHAR[] String)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
String	Character string that is written to the file.

Example 1

A constant character string is written to the file transferred by the handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fputs", FileHandle,
"write this!")
```

Example 2

The character string contained in the variable “StringVar[]” is written to the file transferred by the handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fputs", FileHandle,
StringVar[])
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-5	Maximum size of directory C:\KRC\ROBOTER\UserFiles has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.

6.16.13 krl_fwriteln()

Description

Writes a character string to a file line by line

This function can be used to write a constant character string or variable character string to a file. An end-of-line character is appended to the character string to be written. If a variable is used, each element of the CHAR array must be initialized and contain a valid value.

The line in the file to which the character string is written depends on the mode used to open the file and the position of the file pointer.

Syntax

```
krl_fwriteln(INT Handle, CHAR[] String)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
String	Character string that is written to the file. If the string is blank, a blank line is written to the file.

Example

A blank line is written to the file transferred by the handle:

```
CWRITE($FCT_CALL, State, Mode, "krl_fwriteln", FileHandle,
" ")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-5	Maximum size of directory C:\KRC\ROBOTER\UserFiles has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.

6.16.14 krl_fprintf()

Description

Formatted writing of a character string to a file

The format to be written is transferred to the function as a character string. The values to be written are also transferred.

The available formats are limited as compared with the standard implementation in the programming language C.

The following formats are permissible:

- “%d”, “%i”, “%u”, “%x”, “%e”, “%f”, “%g”, “%s” and “%c”

The following formats are not permissible:

- “%o”, “%p”, “%n” and lists “[List]”

The following are also not allowed:

- More detailed format definitions for length specifications, signs or fill characters and conversion symbols, e.g. “%6.2.f”, “%-d”, “%+d”, “%#f”, etc.

Syntax

```
krl_fprintf(INT Handle, CHAR[] Format, Par1, ... , Par8)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Format	Character string with the formats to be written (maximum 8) For each format, a parameter *Par1* … *Par8* with the values to be written must be specified. The specified format also defines the data type of the associated parameter.
Par1 … *Par8*	Parameter with the values to be written. The number of parameters must correspond to the number of formats to be written (maximum 8). Permissible data types: INT, REAL, BOOL, ENUM, CHAR and CHAR array

Example

3 values separated by the character “;” are written to the file transferred by the handle and formatted:

```
CWRITE($FCT_CALL, State, Mode, "krl_fprintf", FileHandle,
"%s;%x;%c", "Item1", 'HA0', CharVar)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed, e.g. because the file was opened outside a KRL program or the mode is “r”, “r+”, “rb” or “rb+”, but the file does not exist.
-3	There is no file open.
-5	Maximum size of directory C:\KRC\ROBOTER\UserFiles has been reached.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.
-42	The format string contains an invalid format statement.
-43	The number of values to be read in the format string does not correspond to the number of transferred parameters.
-44	An invalid format was specified for one parameter.

6.16.15 krl_fseek()

Description

Positions the file pointer

This function can be used to position the file pointer at the start or end of a file or at a position relative to the current position.

Syntax

```
krl_fseek(INT Handle, SEEK_MODE_T Mode , INT Offset)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Mode	Mode for positioning the file pointer • **#SEEK_BEGIN**: Sets the file pointer to the start of the file. • **#SEEK_END**: Sets the file pointer to the end of the file. • **#SEEK_CUR**: Sets the pointer to a position relative to the current position.
Offset	Offset to the current position of the file pointer (only relevant if mode = #SEEK_CUR) The offset is specified in bytes and can be either positive or negative.

Example

The file pointer in the file transferred by the handle is set back by 5 bytes:

```
CWRITE($FCT_CALL, State, Mode, "krl_fseek", FileHandle,
#SEEK_CUR,
-5)
```

Error numbers

MSG_NO	Description
-1	Internal error
-2	Operation failed
-3	There is no file open.
-7	Positioning of the file pointer outside the limits of the file (only relevant with positioning mode SEEK_CUR)
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.

6.16.16 krl_fsizeget()

Description

Determines the file size

The file size is specified in bytes. This function can only be used to determine the size of files that have first been opened with krl_fopen().

Syntax

```
krl_fsizeget(INT Handle, INT Size)
```

Explanation of the syntax

Element	Description
Handle	Handle variable that is returned when krl_fopen() is called, unambiguously identifying the file
Size	Variable for the determined file size in bytes

Example

The size of the file transferred by the handle is determined and written to the variable “IntVar”:

```
CWRITE($FCT_CALL, State, Mode, "krl_fsizeget", FileHandle,
IntVar)
```

Error numbers

MSG_NO	Description
-2	Operation failed
-3	There is no file open.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.

6.16.17 krl_mkdir()

Description

Creates a directory

For this operation, the user must possess the required read- and write-access rights.

Syntax

`krl_mkdir(CHAR[]` *Name*`)`

Explanation of the syntax

Element	Description
Name	Directory name without path specification or with relative path specification. • Directory without path specification, e.g. “NewDir”: the directory is created under ROBOTER\UserFiles. • Directory with path specification, e.g. “TestDir/NewDir”: the directory is created under ROBOTER\UserFiles\TestDir. • Directory with network path, e.g. “/MyMount/NewDir”: the directory is created in an enabled network directory. The mount point to the network drive with the name MyMount must first have been created by means of krl_mount().

Example 1

A new directory is created under ROBOTER\UserFiles:

```
CWRITE($FCT_CALL, State, Mode, "krl_mkdir", "NewDir")
```

Example 2

A new directory is created on the mount “MyMount”:

```
CWRITE($FCT_CALL, State, Mode, "krl_mkdir", "/MyMount/
NewDir")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.

6.16.18 krl_rename()

Description

Renames a file or directory

For this operation, the user must possess the required read- and write-access rights. The file to be renamed must not be open, nor may any file already exist with the new name.

Syntax

`krl_rename(CHAR[] OldName, CHAR[] NewName)`

Explanation of the syntax

Element	Description
OldName	Old file or directory name without path specification or with relative path specification. (>>> *6.16.3 "krl_fopen()" Page 42*) (>>> *6.16.17 "krl_mkdir()" Page 56*)
NewName	New file or directory name without path specification or with relative path specification. The path specified here must match the path specified under *OldName*.

Example 1

The file ROBOTER\UserFiles\MyFile.txt is renamed:

```
CWRITE($FCT_CALL, State, Mode, "krl_rename", "MyFile.txt",
"YourFile.txt")
```

Example 2

The directory “NewDir” on the mount “MyMount” is renamed:

```
CWRITE($FCT_CALL, State, Mode, "krl_rename", "/MyMount/
NewDir", "/MyMount/TestDir")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-6	File already open.
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.

6.16.19 krl_remove()

Description

Deletes a file or directory

For this operation, the user must possess the required read- and write-access rights. The file to be deleted must not be open.

Syntax

`krl_remove(CHAR[]` *Name*`)`

Explanation of the syntax

Element	Description
Name	File or directory name without path specification or with relative path specification. (>>> *6.16.3 "krl_fopen()" Page 42*) (>>> *6.16.17 "krl_mkdir()" Page 56*)

Example 1

The file ROBOTER\UserFiles\MyFile.txt is deleted:

```
CWRITE($FCT_CALL, State, Mode, "krl_remove", "MyFile.txt")
```

Example 2

The directory “NewDir” on the mount “MyMount” is deleted:

```
CWRITE($FCT_CALL, State, Mode, "krl_remove", "/MyMount/
NewDir")
```

Error numbers

MSG_NO	Description
-2	Operation failed
-6	File already open.
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-12	At least one function parameter has an incorrect data type.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.

7 Example programs

7.1 External module: calling a function by means of LD_EXT_FCT

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
```

7.2 Command channel $CMD: starting, stopping and deselecting a program

The program A6.SRC is to be started, stopped and deselected via the command channel $CMD. 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
```

7.3 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",BIGSTRIN
G_2[])
...
OFFSET=0
FOR N=1 to 1024
    CAST_FROM (BIGSTRING_2[],OFFSET,VAR[n])
ENDFOR
...
```

7.4 Command channel $FCT_CALL: writing characters to a file

The character “a” is to be written to the file MyTest.TXT via the command channel $FCT_CALL. If the file already exists, the contents of the file are deleted when the file is opened.

For every function call with CWRITE, a check is to be carried out to see whether the function was executed successfully or aborted with an error. If an error has occurred, the corresponding error treatment can be programmed in accordance with the cause of the error. If no error has occurred, the character is to be written and the file closed again.

```
DECL INT HANDLE
DECL STATE_T STAT
DECL MODUS_T MODE
...
CHAR MYCHAR
...
MODE = #SYNC
HANDLE = 0
MYCHAR = "a"
...
; create a file under C:\KRC\ROBOTER\UserFiles
```

```
CWRITE($FCT_CALL, STAT, MODE, "krl_fopen", "MyTest.TXT",
"w", HANDLE)
; check whether a error occured
IF (STAT.RET1 == #CMD_ABORT) THEN
    IF (STAT.MSG_NO == -5) THEN
        ; error treatment if to many files are already open
        ...
    ELSE
        ; check other error codes if required
        ...
    ENDIF

    ELSE
        ; no error occured
        ; write a character to the file
        CWrite($FCT_CALL, STAT, MODE, "krl_fputc", HANDLE,
MYCHAR)
    IF (STAT.RET1 == #CMD_ABORT) THEN
        ; error treatment
        ...
    ENDIF
    ; close the file
CWRITE($FCT_CALL, STAT, MODE, "krl_fclose", HANDLE)
    IF (STAT.RET1 == #CMD_ABORT) THEN
    ; error treatment
    ...
    ENDIF
ENDIF
```

8 Appendix

8.1 Error writing to the command channel $FCT_CALL

Overview

MSG_NO	Description
-1	Internal error
-2	Operation failed: invalid user, incorrect password, mount point not available, file not available, etc.
-3	There is no file open.
-4	End of file has been reached (relevant for read functions).
-5	Maximum size of directory C:\KRC\ROBOTER\UserFiles has been reached (relevant for write functions).
-6	File is open (relevant for opening, renaming or deleting a file).
-7	Positioning of the file pointer outside the limits of the file (only relevant with positioning mode SEEK_CUR)
-8	File system not available: error in specification of target directory.
-10	Invalid number of function parameters transferred.
-11	At least one function parameter has an invalid value.
-12	At least one function parameter has an incorrect data type.
-13	At least one function parameter is not a variable although a variable is expected.
-14	At least one function parameter string is blank.
-15	At least one function parameter string is too long.
-16	At least one function parameter string contains invalid characters, e.g. **\ * ? " < > : , = ;] [ ) (} {**, for paths, etc.
-30	Direct access to the ROBOTER directory is not permissible when opening a file.
-31	Invalid absolute path when opening a file
-32	Mount point not found when opening a file
-40	File cannot be opened because 11 files are already open.
-41	One value could not be read. The values before this value were read correctly.
-42	The format string contains an invalid format statement (relevant for formatted reading and writing).
-43	The number of values to be read in the format string does not correspond to the number of transferred parameters (relevant for formatted reading and writing).
-44	An invalid format was specified for one parameter (relevant for formatted reading and writing).

9 KUKA Service

9.1 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

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. (Agentur)
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
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. +86 400 820 8865
Fax +86 21 5707 2607
864008208865@kuka.com oder CN-ROB-CS-Hotline@kuka.com
www.kuka.com

Germany

KUKA Deutschland GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-1926
Fax +49 821 797-41 1926
CustomerService@kuka.com
www.kuka.com

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 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 Japan K.K.
YBP Technical Center
134 Godo-cho, Hodogaya-ku
Yokohama, Kanagawa
240 0005
Japan
Tel. +81 45 744 7531
Fax +81 45 744 7541
info@kuka.co.jp

Canada

KUKA Robotics Canada Ltd.
2865 Argentia Road, Unit 4-5
Mississauga
Ontario L5N 8G6
Canada
Tel. +1 905 858-5852
Fax +1 905 858-8581
KUKAFocusCenter@KUKARobotics.com
www.kukarobotics.ca

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 CEE GmbH
Gruberstraße 2-4
4020 Linz
Austria
Tel. +43 732 784 752 0
Fax +43 732 793 880
KUKAAustriaOffice@kuka.com
www.kuka.at

Poland

KUKA CEE GmbH Poland
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.com

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 Russia OOO
1-y Nagatinskiy pr-d, 2
117105 Moskau
Russia
Tel. +7 495 665-6241
support.robotics.ru@kuka.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 CEE GmbH
Steyregg, Zweigniederlassung Buchs
Heinrich Wehrli-Strasse 27
5033 Buchs
Switzerland
Tel. +41 62 837 43 20
info.robotics.ch@kuka.com

Slovakia

KUKA CEE GmbH
organizačná zložka
Bojnická 3
831 04 Bratislava
Slovakia
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Spain

KUKA Iberia, S.A.U.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
comercial@kukarob.es

South Africa

Jendamark Automation LTD (Agentur)
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 Automation Taiwan Co. Ltd.
1F, No. 298 Yangguang ST.,
Nei Hu Dist., Taipei City, Taiwan 114
Taiwan
Tel. +886 2 8978 1188
Fax +886 2 8797 5118
info@kuka.com.tw

Thailand

KUKA (Thailand) Co. Ltd.
No 22/11-12 H-Cape Biz Sector Onnut
Sukhaphiban 2 road, Prawet
Bangkok 10250
Thailand
Tel. +66 (0) 90-940-8950
HelpdeskTH@kuka.com

Czech Republic

KUKA Roboter CEE GmbH
organizační složka
Pražská 239
25066 Zdiby
Czech Republic
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Turkey

KUKA CEE GmbH
Şerifali Mah.
Bayraktar Bulv. Beyit Sok. No:9-11
34775 Ümraniye/Istanbul
Turkey
Tel. +90 216 508 14 04
hotline-tr@kuka.com; servis-tr@kuka.com
www.kuka.com

Hungary

KUKA HUNGÁRIA 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
CustomerService@kuka.com
www.kuka.com

United Kingdom

KUKA Robotics UK Ltd
Great Western Street
Wednesbury West Midlands
WS10 7LL
United Kingdom
Tel. +44 121 505 9970
Fax +44 121 505 6589
service@kuka-robotics.co.uk
www.kuka-robotics.co.uk

Index

A

C

D

E

F

H

I

K

L

M

O

P

R

S

T

W