KUKA

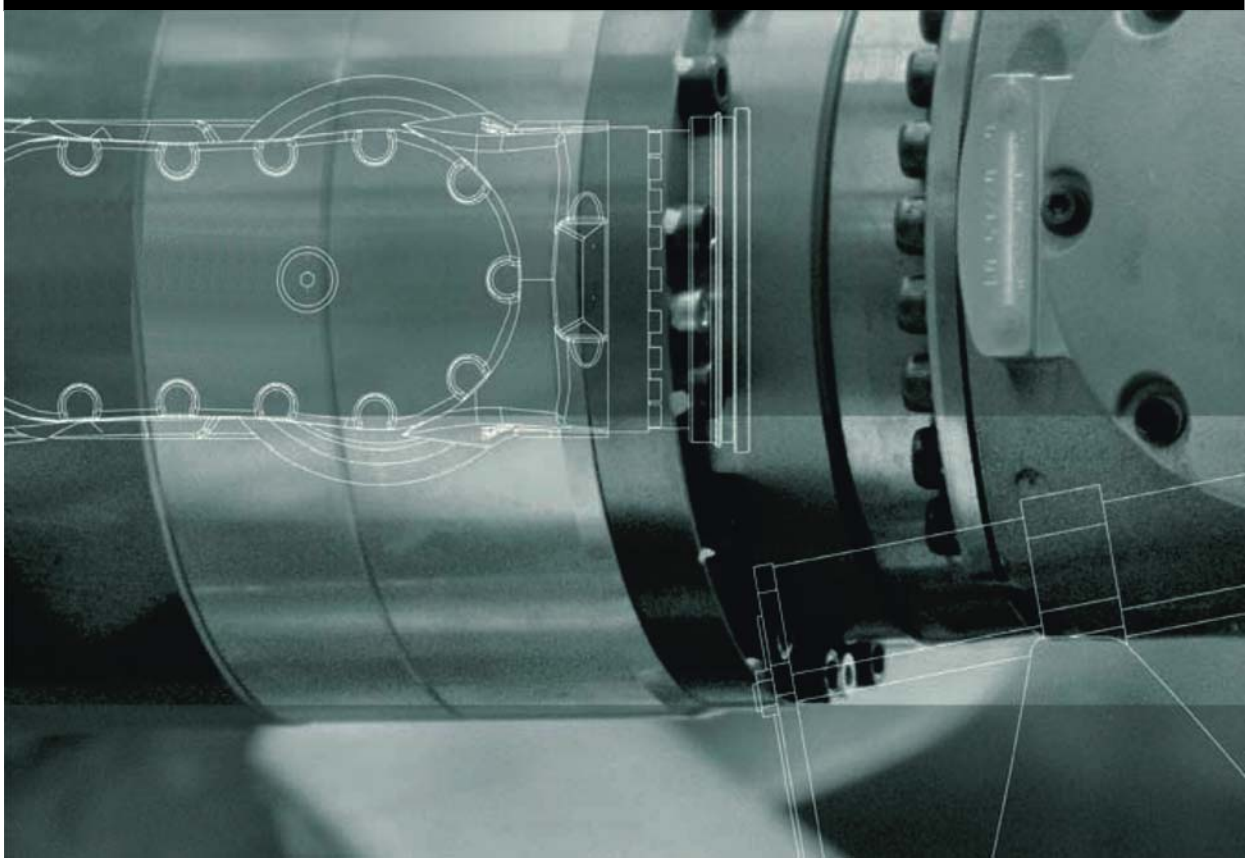
Expert Documentation

KUKA Roboter GmbH

Programming Messages

For KUKA System Software 8.2

For VW System Software 8.2



Issued: 26.05.2011

Version: KSS 8.2 Meldungen V1 en

KUKA Roboter GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

Translation of the original documentation

KIM-PS5-DOC

Publication:	Pub KSS 8.2 Meldungen en
Bookstructure:	KSS 8.2 Meldungen V1.2
Label:	KSS 8.2 Meldungen V1 en

Contents

1 Introduction

1.1 Target group

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

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

For optimal use of our products, we recommend that our customers take part in a course of training at KUKA College. Information about the training program can be found at www.kuka.com or can be obtained directly from our subsidiaries.

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

- Documentation for the manipulator
- Documentation for the robot controller
- Operating and programming instructions for the KUKA System Software
- Documentation relating to options and accessories
- Parts catalog on storage medium

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

These warnings are relevant to safety and **must** be observed.

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

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

CAUTION These warnings mean that minor physical 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.

Hints

These hints 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 Trademarks

Windows is a trademark of Microsoft Corporation.

1.5 Terms used

Term	Description
Dialog	Dialog message
KCP	The KCP (KUKA Control Panel) teach pendant has all the operator control and display functions required for operating and programming the industrial robot. The KCP variant for the (V)KR C4 is called KUKA smartPAD. The general term “KCP” is used in this documentation, however.
KRL	KUKA Robot Language
Generating a message	Transferring a message to the message buffer and displaying it in the message window. Notification messages only: displaying the message in the message window. (Notification messages are not managed in the message buffer.)
Deleting a message	Removing a message from the message window and the message buffer. Notification messages only: removing a message from the message window. (Notification messages are not managed in the message buffer.)

2 Description of functions

2.1 Message types

An icon is displayed in the message window alongside every message. The icons are permanently assigned to the message types and cannot be altered by the programmer.

No predefined reactions of the industrial robot are linked to the different message types (e.g. robot brakes or program is stopped). The desired reactions must be programmed.

The following types of message can be programmed:

Icon	Type
	**Notification message** Notification messages are suitable for displaying general information. Notification messages can only be deleted using the buttons **OK** and **Confirm all**.
	**Acknowledgement message** Acknowledgement messages are suitable for displaying information of which the user must be made aware. Acknowledgement messages can be deleted (acknowledged) using the buttons **OK** and **Confirm all**. In the case of an acknowledgement message (unlike a notification message), it is possible to check whether or not the user has acknowledged it. It is possible, for example, to stop the program until the message has been acknowledged.
	**Wait message** Wait messages can be deleted from the program by means of a function. The user can also delete the message using the **Simulate** button.
	**Status message** Status messages are suitable for indicating a change of status (e.g. variable X changes from TRUE to FALSE). Status messages are deleted from the program by means of a function. The message is deleted when the status that triggered it (e.g. variable X is FALSE) is no longer applicable.
	**Dialog message** Dialog messages can be deleted using a button that can be labeled by the programmer. Up to 7 buttons can be defined. How program execution continues can be made dependent on which button the user selects. Dialog messages are suitable for displaying questions that must be answered by the user.

2.2 Message programming properties

The programmer can use KRL (KUKA Robot Language) to program his own messages.

Message programming properties:

- The message mechanism is reentrant, i.e. it can be started more than once at the same time.
- Up to 3 parameters can be integrated into a message.
- Generated messages are stored in a message buffer until they are deleted.

  Exception: Notification messages are not managed in the message buffer (“fire and forget” principle).
- The messages can be easily checked or deleted. It is of no importance what order the messages were generated in or whether there are other messages present.

  Exception: Notification messages cannot be checked and cannot be deleted by means of a KRL instruction.

3 Programming

3.1 Symbols and fonts

The following symbols and fonts are used in the descriptions of KRL statements and functions:

Element	Representation
KRL code	■ Courier font ■ Upper/lower-case letters Examples: `KrlMsg_t`; `log_to_DB`; `struc`
Elements that must be replaced by program-specific entries	■ Italics ■ Lower-case letters Examples: *number*; *handle*
Elements that are mutually exclusive	■ Separated by the "|" symbol Example: `true` |`false`

3.2 Basic principle of message programming

Description This example is the simplest possible form of a message. It illustrates the basic principle of message programming.

Message The following notification message is to be generated:

Fig. 3-1: Example of a notification message

Program

```
    ...
 1  decl KrlMsg_t mymessage
 2  decl KrlMsgPar_t mypar[3]
 3  decl KrlMsgOpt_t myoptions
 4  int nhandle
 5  ...
 6  INI
 7  ...
 8  mymessage = {modul[] "User", Nr 123, msg_txt[] "My new
message."}
 9  nhandle = Set_KrlMsg (#notify, mymessage, mypar[], myoptions)
    ...
```

Line	Description
1 … 4	Declaration of the required variables
2	Variables of type KrlMsgPar_t must always be declared with 3 array elements.

Line	Description
8	**Definition of the message text, number and originator:** The variable of type KrlMsg_t contains the parts of the message that are displayed in the message window (originator, number, text). A variable of type KrlMsg_t is an integral part of the programming of every message.
9	**Definition of the message type and generation of the message:** The function Set_KrlMsg generates the message `mymessage`. The message is of type `#notify`, i.e. a notification message. The function Set_KrlMsg also has additional parameters: an array of type KrlMsgPar_t and a variable of type KrlMsgOpt_t. These are not used in the example. They must always be specified, however. (Dialog messages are not generated with Set_KrlMsg, but with the similar function Set_KrlDlg.)

3.3 Message properties

3.3.1 Defining the originator, number and message text

Statement

name = {`modul[]` "*originator*", `nr` *number*, `msg_txt[]` "*text*"}

Description

A variable of type KrlMsg_T is used to define the message components that are displayed in the message window: originator, message number and message text.

Element	Description
name	Type: KrlMsg_T Variable name for the message
originator	Type: CHAR Originator displayed in the message window ■ Maximum length: 24 characters ■ The originator must not consist of blanks. The system puts the name of the originator in angle brackets. In this way, it is possible to distinguish between system messages and user-defined messages in the message window. Example: The originator “myTech” appears in the message window as <myTech>.

Element	Description
number	Type: INT Message number. Message numbers may be used more than once. ■ > 0
text	Type: CHAR Message text (or key for a message database) ■ Maximum length: 80 characters ■ Permissible characters: letters, numbers, underscores. The text must not start with a number, however. ■ The text must not consist of blanks. The message text (or database key) can contain “%1”, “%2” and/or “%3” as placeholders. If this is the case, the placeholders must be assigned parameters. (>>> 3.3.2 "Assigning parameters to placeholders" Page 11)

3.3.2 Assigning parameters to placeholders

Statement

param[n] = {par_type *type*, par_txt[] "*text*" | par_int *int* | par_real *real* | par_bool *bool*}

Description

The message text can contain “%1”, “%2” and/or “%3” as placeholders. If this is the case, the placeholders must be assigned parameters.

Element	Description
param[n]	Type: KrlMsgPar_T Variable name for the parameter Array index[*n*]: ■ **1**: parameter for placeholder “%1” ■ **2**: parameter for placeholder “%2” ■ **3**: parameter for placeholder “%3”
Type	Type: KrlMsgParType_T Type of parameter ■ **#value**: The parameter is inserted into the message text as specified in *text*, *int*, *real* or *bool*. ■ **#key**: The parameter is a key that must be searched for in the message database. ■ **#empty**: The parameter is empty.
text	Type: CHAR Text of the parameter (or key for a message database) ■ The text can have a maximum length of 26 characters and must not consist of blank spaces.
int	Type: INT This can be used to fill the placeholder with an integer value. It can only be used in conjunction with *type*=#value.

Element	Description
real	Type: REAL This can be used to fill the placeholder with a real value. It can only be used in conjunction with *type*=#value.
bool	Type: BOOL This can be used to fill the placeholder with a Boolean value. It can only be used in conjunction with *type*=#value.

Par_int *int*, par_real *real* and par_bool *bool* enable the programmer to use an integer, real or Boolean value as a parameter without first having to convert the value to a string (e.g. with SWRITE). Conversion is carried out automatically. Par_Bool is converted to the string “True” or “False”.

Example

Programmed message text: “Value of analog output 5: %1”

```
decl KrlMsg_t mymessage
...
mymessage = {modul[] "My module", Nr 987, msg_txt[] "Value of analog
output 5: %1"}
```

Placeholder “%1” is to be replaced by the value of analog output $ANOUT[5]. KRL parameter to be programmed:

```
decl KrlMsgPar_t par[3]
...
par[1] = {par_type #value, par_real 0.0}
par[1].par_real = $anout[5]
```

3.3.3 Labeling buttons for dialog messages

Statement

softkey[n] = {sk_type *keytype*, sk_txt[] "*keyname*"}

Description

For button assignment in the case of dialog messages.

The buttons were implemented as softkeys in earlier versions of the KCP. This is the origin of the abbreviation “SK” or “sk” in data types.

Element	Description
softkey[n]	Type: KrlMsgDlgSK_T Variable name for the button. A maximum of 7 buttons are available. Array index[*n*] = **1 …7**: ■ **1**: label for first button on left ■ **2**: label for second button on left ■ etc.

Element	Description
keytype	Type: KrlMsgParType_T Type of button label ■ **#value**: sk_txt[ ] corresponds to the button label. ■ **#key**: sk_txt[ ] is the database key containing the label of the button. ■ **#empty**: The button is not assigned.
keyname	Type: CHAR Label of the button (or key for a message database) ■ The text can have a maximum length of 10 characters and must not consist of blank spaces.

3.3.4 Defining the reaction to a message

Statement

```
options = {vl_stop TRUE|FALSE, clear_p_reset TRUE|FALSE,
clear_p_SAW TRUE|FALSE, log_to_DB TRUE|FALSE}
```

Description

Element	Description
options	Type: KrlMsgOpt_T Name of the variable for the message response
vl_stop	Type: BOOL ■ **TRUE:** Set_KrlMsg/Set_KrlDlg triggers an advance run stop. ■ **FALSE:** Set_KrlMsg/Set_KrlDlg does not trigger an advance run stop. Default: TRUE
clear_p_reset	Type: BOOL Delete message when the program is reset or deselected? ■ **TRUE:** All status messages, acknowledgement messages and wait messages generated by Set_KrlMsg() with the variable *options* are deleted. ■ **FALSE:** The messages are not deleted. Default: TRUE Notification messages can only be deleted using the buttons **OK** and **Confirm all**. The following always applies for dialog messages: clear_p_reset = TRUE.

Element	Description
clear_p_SAW	Type: BOOL Delete message when a block selection is carried out using the button **Line Sel.**? ■ **TRUE:** All status messages, acknowledgement messages and wait messages generated by Set_KrlMsg() with the variable *options* are deleted. ■ **FALSE:** The messages are not deleted. Default: FALSE Notification messages can only be deleted using the buttons **OK** and **Confirm all**. Dialog messages: No block selection is possible while a dialog is present on the user interface, as all operator control elements are deactivated.
log_to_DB	Type: BOOL ■ **TRUE:** The message is logged. ■ **FALSE:** The message is not logged. Default: FALSE

3.4 Generating, checking and deleting messages

3.4.1 Generating a message (Set_KrlMsg)

Description

The function Set_KrlMsg() generates a message. This means that the message is transferred to the message buffer and displayed from there in the message window.

Exceptions:

- Notification messages are displayed in the message window by means of Set_KrlMsg(). They are not managed in the message buffer, however (“fire and forget” principle).
- To generate dialog messages, the function Set_KrlDlg() must be used.
  (>>> 3.4.4 "Generating a dialog message (Set_KrlDlg)" Page 16)

Syntax

```
handle = Set_KrlMsg(type, name, params[], options)
```

Explanation of the syntax

Element	Description
handle	Type: INT Variable for the return value ■ **-1**: The message could not be generated (e.g. because the message buffer is too full). ■ **> 0**: The message was generated successfully. The return value is a valid handle that can be used for further operations for this message, e.g. for deleting the message with Clear_KrlMsg(). **Note:** For notification messages, the handle is always **0**.
Type	Type: EKrlMsgType; IN parameter Defines the type of the message to be generated. ■ **#notify**: Notification message ■ **#state**: Status message ■ **#quit**: Acknowledgement message ■ **#waiting**: Wait message
name	Type: KrlMsg_T; OUT parameter Structure defining the name, originator and message text
params	Type: KrlMsgPar_T; OUT parameter Structure containing the message parameters
options	Type: KrlMsgOpt_T; OUT parameter Structure containing the message reaction

3.4.2 Checking a message (Exists_KrlMsg)

Description

The function Exists_KrlMsg() can be used to check whether a specific message still exists. It also checks whether this message is still present in the message buffer.

The function does not wait until the message has been deleted, but merely searches the buffer for the message with this handle. The KRL program must therefore be polled cyclically until the message has been deleted.

Notification messages cannot be checked, as they are not managed in the message buffer.

Syntax

result = `Exists_KrlMsg(`*handle*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Return value ■ **TRUE**: This message still exists in the message buffer. ■ **FALSE**: This message no longer exists in the message buffer (because it has been deleted).
handle	IN parameter The handle provided for the message by the function Set_KrlMsg()

3.4.3 Deleting a message (Clear_KrlMsg)

Description

The function Clear_KrlMsg() can be used to delete a message. This means that the message is removed from the message buffer and the message window.

Notification messages cannot be deleted in this way, as they are not managed in the message buffer. Notification messages can only be deleted using the buttons **OK** and **Confirm all**.

Syntax

result = `Clear_KrlMsg(`*clear*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Return value ■ **TRUE**: The message has been deleted. ■ **FALSE**: The message could not be deleted.
clear	IN parameter ■ The handle provided by the function Set_KrlMsg(): the message to which the handle refers is deleted. ■ **-1**: All messages initiated by this process are deleted. ■ **-99**: All user-defined messages are deleted. (All processes: robot interpreter, Submit interpreter, command interpreter)

3.4.4 Generating a dialog message (Set_KrlDlg)

Description

The function Set_KrlDlg() generates a dialog message. This means that the message is transferred to the message buffer and displayed from there in the message window.

The function merely generates the dialog. It does not wait until the dialog has been answered.

A dialog cannot be generated until no other dialog is active.

Syntax

handle = `Set_KrlDlg(`*name*`, `*params*`[], `*softkey*`[], `*options*`)`

The buttons were implemented as softkeys in earlier versions of the KCP. This is the origin of the abbreviation “SK” or “sk” in data types.

Explanation of the syntax

Element	Description
handle	Type: INT Variable for the return value ■ **-1**: The dialog could not be generated (e.g. because another dialog has not yet been answered or the message buffer is too full). ■ **> 0**: The dialog was generated successfully. The return value is a valid handle that can be used for further operations for this dialog.
name	Type: KrlMsg_T; OUT parameter Structure defining the name, originator and message text

Element	Description
params	Type: KrlMsgPar_T; OUT parameter Structure containing the message parameters
softkey	Type: KrlMsgDlgSK_T; OUT parameter Structure containing the button assignment
options	Type: KrlMsgOpt_T; OUT parameter Structure containing the message reaction

3.4.5 Checking a dialog message (Exists_KrlDlg)

Description

The function Exists_KrlDlg() can be used to check whether a specific dialog still exists. It also checks whether this dialog is still present in the message buffer.

The function does not wait until the dialog has been deleted, but merely searches the buffer for the dialog with this handle. The KRL program must therefore be polled cyclically until the dialog has been answered or deleted.

Syntax

result = `Exists_KrlDlg(`*handle*`,` *answer*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Return value ■ **TRUE**: This dialog still exists in the message buffer. ■ **FALSE**: This dialog no longer exists in the message buffer. It has therefore been answered.
handle	IN parameter The handle provided for the dialog by the function Set_KrlDlg()
answer	OUT parameter Number of the button used to answer the dialog. This parameter does not need to be initialized. It is written by the system. ■ **1 … 7**: Answer with the corresponding button ■ **0**: If the dialog has not been answered, but deleted (e.g. the dialog has been deleted by means of Clear_KrlMsg() by an interrupt or by a different process).

3.5 Reading the message buffer (Get_MsgBuffer)

Description

The function Get_MsgBuffer() reads the message buffer and writes the messages in the buffer to the OUT parameter MsgBuf[].

The size of the buffer is 150.

Get_MsgBuffer() can read the following message types from the buffer:

- Status messages from the kernel system (#sys_state)
- Acknowledgement messages from the kernel system (#sys_quit)
- User-defined status messages (#usr_state)
- User-defined acknowledgement messages (#usr_quit)
- User-defined dialog messages (#usr_dlg)

- User-defined wait messages (#usr_wait)

Syntax

count = Get_MsgBuffer(*MsgBuf*[])

Explanation of the syntax

Element	Description
count	Type: INT Return value: Number of messages in the message buffer This value can be used with a FOR loop for the array MsgBuf[] to read the details about the messages.
MsgBuf	Type: MsgBuf_T; OUT parameter Array of buffer elements containing all the messages in the buffer. The array is filled in ascending order, without gaps, starting with Index=1. If there are fewer messages in the buffer than represented by the size of the array, these array elements are not initialized. Contents of the structure MsgBuf_T: (>>> 5.1 "Data types" Page 27)

Example

```
Def ReadMsgBuffer ( )
  decl MsgBuf_T buffer[100]
  int message_count
  message_count = Get_MsgBuffer (buffer[])
  FOR n=1 TO message_count
   if (buffer[n].type==#sys_quit) then
        ...
   endif
  ENDFOR
End
```

4 Examples

4.1 Notification message

Message

The following notification message is to be generated:

Fig. 4-1: Message on the user interface

The word “gripper” is to be inserted by a placeholder.

Program

```
    ...
 1  decl KrlMsg_T msg
 2  decl KrlMsgPar_T par[3]
 3  decl KrlMsgOpt_T opt
 4  int nHandle
 5  ...
 6  msg = {modul[] "MyTech", nr 123, msg_txt[] "Check %1."}
 7  par[1] = {par_type #value, par_txt[] "gripper"}
 8  opt = {vl_stop true, clear_p_reset true, clear_p_SAW false,
log_to_DB false}
 9  ...
10  nHandle = Set_KrlMsg (#notify, msg, par[], opt)
    ...
```

Description

Line	Description
1 … 4	Declaration of the required variables
2	Variables of type KrlMsgPar_t must always be declared with 3 array elements.
6	**Definition of the message text, number and originator:** The variable `msg` of type KrlMsg_t defines the parts of the message that are displayed in the message window (originator, number, text). The text contains the placeholder `%1`. The placeholder must be filled.
7	**Filling the placeholder:** `par[1]` defines the type and contents of the placeholder `%1`. `#value` defines that the contents of `par_txt[]` are a text. (Not a key to be searched for in a database.) As there are no placeholders %2 and %3, Par[2] and Par[3] may be left not initialized. The parameters are then automatically empty.
8	**Definition of message response:** Each parameter has its own default value here. For this reason, the line could also be omitted. (Set_KrlMsg must always contain `opt`, however.)
10	**Definition of the message type and generation of the message:** The function Set_KrlMsg generates the message `msg`. `#notify` defines that the message is generated as a notification message.

4.2 Acknowledgement message

Message

The following acknowledgement message is to be generated:

Fig. 4-2: Message on the user interface

The words “gripper” and “weld gun” are to be inserted by placeholders. The program is not to continue until the message has been acknowledged.

Program

```
    ...
 1  decl KrlMsg_T msg
 2  decl KrlMsgPar_T par[3]
 3  decl KrlMsgOpt_T opt
 4  int nHandle
 5  ...
 6  msg = {modul[] "MyTech", nr 231, msg_txt[] "Check %1 and %2."}
 7  par[1] = {par_type #value, par_txt[] "gripper"}
 8  par[2] = {par_type #value, par_txt[] "weld gun"}
 9  opt = {vl_stop true, clear_p_reset true, clear_p_SAW false,
log_to_DB true}
10  ...
11  nHandle = Set_KrlMsg (#quit, msg, par[], opt)
12
13  while (Exists_KrlMsg(nHandle))
14    wait sec 0.1
15  endwhile
    ...
```

Description

Line	Description
1 … 4	Declaration of the required variables
2	Variables of type KrlMsgPar_t must always be declared with 3 array elements.
6	**Definition of the message text, number and originator:** The variable `msg` of type KrlMsg_t defines the parts of the message that are displayed in the message window (originator, number, text). The text contains placeholders %1 and %2. The placeholders must be filled.
7, 8	**Filling the placeholder:** `par[1]` and `par[2]` define the type and contents of placeholders `%1` and `%2`. `#value` defines that the contents of `par_txt[]` are a text. (Not a key to be searched for in a database.) As there is no placeholder %3, Par[3] may be left not initialized. The parameter is then automatically empty.
9	**Definition of message response:** Contrary to the default settings, it has been defined that the message is logged.

Line	Description
11	**Definition of the message type and generation of the message:** The function Set_KrlMsg generates the message `msg`. `#quit` defines that the message is generated as an acknowledgement message.
13 … 15	Program execution remains in the wait loop until the user has acknowledged the message.

4.3 Wait message

Message

The following wait message is to be generated:

Fig. 4-3: Message on the user interface

The wait message is to be removed from the message window when a valid program number is received from the PLC (valid program numbers in this example are ≥ 1.)

In the case of wait messages, the user can also hide the message at any time using the **Simulate** button. In this example, program number 0 is simulated by means of **Simulate**.

Program

```
...
 1  decl KrlMsg_T msg
 2  decl KrlMsgPar_T par[3]
 3  decl KrlMsgOpt_T opt
 4  int  nHandle, validPgNo
 5  bool retVal
 6  ...
 7  msg = {modul[] "MsgTech", Nr 1, msg_txt[] "This is a test."
 8  validPgNo = -1
 9  ...
10  nHandle = Set_KrlMsg (#waiting, msg, par[], opt)
11  IF (nHandle > 0) THEN
12    repeat
13       validPgNo = getPgNoFromPLC()
14       if (Exists_KrlMsg (nHandle) == false) then
15           validPgNo = 0
16       endif
17    until (validPgNo > -1)
18
19    if (Exists_KrlMsg (nHandle) == true) then
20       retVal = Clear_KrlMsg (nHandle)
21    endif
22  ENDIF
...
```

Description

Line	Description
1 … 5	Declaration of the required variables
2	Variables of type KrlMsgPar_t must always be declared with 3 array elements.
7	**Definition of the message text, number and originator:** The variable `msg` of type KrlMsg_t defines the parts of the message that are displayed in the message window (originator, number, text).

Line	Description
8	The variable for the program number is initialized with a value that must not be from the PLC.
10	**Definition of the message type and generation of the message:** The function Set_KrlMsg generates the message `msg`. The message is of type `#waiting`. Set_KrlMsg must always also contain a variable for parameters (type KrlMsgPar_T) and a variable for the message response (type KrlMsgOpt_T), even if the message contains no parameters or the default message response is retained.
11 … 22	This statement block is to be executed once the message has been generated successfully.
12 … 17	This statement block is repeated until a valid program number is present or the user has deleted the message by means of **Simulate**.
13	The robot controller requests a program number from the PLC.
14 … 16	Check whether the user has pressed **Simulate**. (In this case, the message is already hidden, i.e. Exists_KrlMsg (nHandle) == FALSE.) If so, set the variable for the program number to 0.
19 … 21	If the user has not pressed **Simulate**, the message is still active. It is then deleted by means of Clear_KrlMsg.

4.4 Status message

Message

The following status message is to be generated:

Fig. 4-4: Message on the user interface

The container number is to be taken from a database. The message is to be triggered by `state_OK = FALSE`. Once this status has been eliminated, the message is to be reset.

Program

```
    ...
 1  decl KrlMsg_T msg
 2  decl KrlMsgPar_T par[3]
 3  decl KrlMsgOpt_T opt
 4  int nHandle
 5  bool state_OK, result
 6  ...
 7  IF state_OK == false THEN
 8    msg = {modul[] "MyTech", Nr 5, msg_txt[] "Container nr %1 is
empty."}
 9    par[1] = {par_type #key, par_txt[] "mytech_container_nr"}
10    opt = {vl_stop true, clear_p_reset true, clear_p_SAW false,
log_to_DB false}
11
12    nHandle = Set_KrlMsg (#state, msg, par[], opt)
13  ENDIF
14  ...
15  REPEAT
16    IF (nHandle > 0) then
17      if state_OK == true then
18        result = Clear_KrlMsg (nHandle)
19      endif
20    ENDIF
21    wait sec 0.5
22  UNTIL state_OK == true
    ...
```

Description

Line	Description
1 … 5	Declaration of the required variables
2	Variables of type KrlMsgPar_t must always be declared with 3 array elements.
7 … 13	The status message is triggered if state_OK == FALSE.
8	**Definition of the message text, number and originator:** The variable `msg` of type KrlMsg_t defines the parts of the message that are displayed in the message window (originator, number, text). The text contains the placeholder `%1`. The placeholder must be filled.
9	**Filling the placeholder:** `par[1]` defines the type and contents of the placeholder `%1`. `#key` defines that the contents of `par_txt[]` are a key that is to be searched for in a database. As there are no placeholders %2 and %3, Par[2] and Par[3] may be left not initialized. The parameters are then automatically empty.
10	**Definition of message response:** Each parameter has its own default value here. For this reason, the line could also be omitted. (Set_KrlMsg must always contain `opt`, however.)
12	**Definition of the message type and generation of the message:** The function set_KrlMsg generates the message `msg`. The message is of type `#state`.
15 … 22	In the case of a REPEAT loop, a check is carried out every 0.5 s to see whether the message has been generated and whether the state that triggers it is still active. If this is no longer the case, the message is deleted and the REPEAT loop is exited.

Line	Description
16	`(nHandle > 0)` = the message has been generated successfully.
17 … 19	Monitor whether the status that triggered the message (state_OK == FALSE) is still applicable. If not, the status message is to be withdrawn.
18	The status message is reset.

4.5 Dialog message

Message

The following dialog message is to be generated:

Fig. 4-5: Message on the user interface

Program

```
    ...
 1  decl KrlMsg_T msg
 2  decl KrlMsgPar_T par[3]
 3  decl KrlMsgOpt_T opt
 4  decl KrlMsgDlgSK_T SK[7]
 5  int nHandle, keynumber
 6  ...
 7  msg = {modul[] "MyTech", Nr 7, msg_txt[] "Continue?"}
 8  SK[1] = {sk_Type #value, sk_txt[] "yes"}
 9  SK[2] = {sk_Type #value, sk_txt[] "no"}
10  ...
11  nHandle = Set_KrlDlg (msg, par[], SK[], opt)
12  IF (nHandle > 0) THEN
13     while (Exists_KrlDlg(nHandle, keynumber))
14         wait Sec 0.1
15     endwhile
16     switch keynumber
17         case 1
18          ...
19         case 2
20          ...
21         case 0
22          ...
23     endswitch
24  ENDIF
    ...
```

Description

Line	Description
1 … 5	Declaration of the required variables
2	Variables of type KrlMsgPar_T must always be declared with 3 array elements.
4	Variables of type KrlMsgDlgSK_T must always be declared with 7 array elements.

Line	Description
7	**Definition of the message text, number and originator:** The variable `msg` of type KrlMsg_t defines the parts of the message that are displayed in the message window (originator, number, text).
8, 9	**Labeling buttons:** `#value` defines that the contents of `sk_txt[]` are a text. (Not a key to be searched for in a database.) The first button on the left is labeled with “yes”. The second button on the left is labeled with “no”. The other buttons are not to be labeled. SK[3] … SK[7] thus remain not initialized.
11	**Outputting the message:** The function Set_KrlDlg generates the message `msg`. Set_KrlDlg must always also contain a variable for parameters (type KrlMsgPar_T) and a variable for the message response (type KrlMsgOpt_T), even if the message contains no parameters or the default message response is retained.
12 … 24	This statement block is to be executed once the message has been generated successfully.
13 … 15	Wait until the user has answered the dialog. In this case, the dialog is deleted in the message window and Exists_KrlDlg (nHandle, keynumber) == false. The variable *keynumber* does not have to be initialized. It is written by the system. If the dialog is not answered, but deleted (e.g. by means of Clear_KrlMsg() by an interrupt or by a different process), Exists_KrlDlg has the return value 0.
16 … 23	If the dialog has been answered or deleted, different statement blocks are executed in accordance with the specific answer.

5 Appendix

5.1 Data types

EKrlMsgType ENUM EKrlMsgType notify, state, quit, dialog, waiting

KrlMsg_T STRUC KrlMsg_T **CHAR** modul[24], **INT** nr, **CHAR** msg_txt[80]

KrlMsgDlgSK_T STRUC KrlMsgDlgSK_T **KrlMsgParType_T** sk_type, **CHAR** sk_txt[10]

KrlMsgOpt_T STRUC KrlMsgOpt_T **BOOL** vl_stop, clear_p_reset, clear_p_SAW, log_to_DB

All components and the overall structure may be left not initialized (this does not cause a runtime error message).

KrlMsgPar_T STRUC KrlMsgPar_T **KrlMsgParType_T** par_type, **CHAR** par_txt[26], **INT** par_int, **REAL** par_real, **BOOL** par_bool

- All components and the overall structure may be left not initialized. Structures that are not initialized will be interpreted as Parameter=#empty.
- If the component par_type is not initialized, this parameter is interpreted as #empty.
- If the component par_type is initialized with #value or #key, but none of the subsequent components (par_txt[], par_int, par_real, par_bool) is initialized, this parameter is interpreted as #empty.
- If the component par_type is initialized with #key, the component par_txt[] is not initialized, but one of the remaining components is initialized (par_int, par_real, par_bool), this results in an error message.
- If the component par_type is initialized with #value or #key, and if more than one of the remaining components (par_txt[], par_int, par_real, par_bool) is initialized, the first initialized component is always deemed to be valid. (Order: par_txt[], par_int, par_real, par_bool)

  Example: `par[1]={par_type #value, par_int 7, par_bool true}`: The value of the parameter is “**7**”.

KrlMsgParType_T ENUM KrlMsgParType_T value, key, empty

MsgBuf_T STRUC MsgBuf_T **MsgBufMsgType_T** type, **INT** nr, modul[24], **CHAR** msg_txt[80], **KrlMsgParType_T** par_type1, **CHAR** par_txt1[40], **KrlMsgParType_T** par_type2, **CHAR** par_txt2[40], **KrlMsgParType_T** par_type3, **CHAR** par_txt3[40], **INT** handle

- type: message type (#sys_quit, #usr_State, …)
- nr: Message number
- modul[ ]: only initialized for messages of type #usr_…, because with type #sys_… the parameter is usually a database key that cannot be used by the user. Represents the originator of the message.
- msg_txt[ ]: message text or message key. (Once again, only initialized for type #usr_…, because with #sys_… the message number is always the key.)
- par_type1 - 3: parameter type (#empty, #key, #value)
- par_txt1 – 3: text or database key of the parameter
- handle: internal handle for this message (only initialized for user-defined messages)

MsgBufMsgType_T ENUM MsgBufMsgType_T sys_quit, sys_state, usr_quit, usr_wait, usr_state, usr_dlg

Message types:

- Status messages from the kernel system (#sys_state)
- Acknowledgement messages from the kernel system (#sys_quit)
- User-defined status messages (#usr_state)
- User-defined acknowledgement messages (#usr_quit)
- User-defined dialog messages (#usr_dlg)
- User-defined wait messages (#usr_wait)

6 KUKA Service

6.1 Requesting support

Introduction

The KUKA Roboter GmbH documentation offers information on operation 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:

- 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

6.2 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

Headland Machinery Pty. Ltd.
Victoria (Head Office & Showroom)
95 Highbury Road
Burwood
Victoria 31 25
Australia
Tel. +61 3 9244-3500
Fax +61 3 9244-3501
vic@headland.com.au
www.headland.com.au

Belgium

KUKA Automatisering + Robots N.V.
Centrum Zuid 1031
3530 Houthalen
Belgium
Tel. +32 11 516160
Fax +32 11 526794
info@kuka.be
www.kuka.be

Brazil

KUKA Roboter do Brasil Ltda.
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 Automation Equipment (Shanghai) Co., Ltd.
Songjiang Industrial Zone
No. 388 Minshen Road
201612 Shanghai
China
Tel. +86 21 6787-1808
Fax +86 21 6787-1805
info@kuka-sha.com.cn
www.kuka.cn

Germany

KUKA Roboter GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-4000
Fax +49 821 797-1616
info@kuka-roboter.de
www.kuka-roboter.de

France

KUKA Automatisme + Robotique SAS
Techvallée
6, Avenue du Parc
91140 Villebon S/Yvette
France
Tel. +33 1 6931660-0
Fax +33 1 6931660-1
commercial@kuka.fr
www.kuka.fr

India

KUKA Robotics India Pvt. Ltd.
Office Number-7, German Centre,
Level 12, Building No. - 9B
DLF Cyber City Phase III
122 002 Gurgaon
Haryana
India
Tel. +91 124 4635774
Fax +91 124 4635773
info@kuka.in
www.kuka.in

Italy

KUKA Roboter Italia S.p.A.
Via Pavia 9/a - int.6
10098 Rivoli (TO)
Italy
Tel. +39 011 959-5013
Fax +39 011 959-5141
kuka@kuka.it
www.kuka.it

Japan

KUKA Robotics Japan K.K.
Daiba Garden City Building 1F
2-3-5 Daiba, Minato-ku
Tokyo
135-0091
Japan
Tel. +81 3 6380-7311
Fax +81 3 6380-7312
info@kuka.co.jp

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 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

Austria

KUKA Roboter Austria 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

Poland

KUKA Roboter Austria GmbH
Spółka z ograniczoną odpowiedzialnością
Oddział w Polsce
Ul. Porcelanowa 10
40-246 Katowice
Poland
Tel. +48 327 30 32 13 or -14
Fax +48 327 30 32 26
ServicePL@kuka-roboter.de

Portugal

KUKA 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

OOO KUKA Robotics Rus
Webnaja ul. 8A
107143 Moskau
Russia
Tel. +7 495 781-31-20
Fax +7 495 781-31-19
kuka-robotics.ru

Sweden

KUKA Svetsanläggningar + Robotar AB
A. Odhners gata 15
421 30 Västra Frölunda
Sweden
Tel. +46 31 7266-200
Fax +46 31 7266-201
info@kuka.se

Switzerland

KUKA Roboter Schweiz AG
Industriestr. 9
5432 Neuenhof
Switzerland
Tel. +41 44 74490-90
Fax +41 44 74490-91
info@kuka-roboter.ch
www.kuka-roboter.ch

Spain

KUKA Robots IBÉRICA, S.A.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
Fax +34 93 8142-950
Comercial@kuka-e.com
www.kuka-e.com

South Africa

Jendamark Automation LTD (Agency)
76a York Road
North End
6000 Port Elizabeth
South Africa
Tel. +27 41 391 4700
Fax +27 41 373 3869
www.jendamark.co.za

Taiwan

KUKA Robot Automation Taiwan Co., Ltd.
No. 249 Pujong Road
Jungli City, Taoyuan County 320
Taiwan, R. O. C.
Tel. +886 3 4331988
Fax +886 3 4331948
info@kuka.com.tw
www.kuka.com.tw

Thailand

KUKA Robot Automation (M)SdnBhd
Thailand Office
c/o Maccall System Co. Ltd.
49/9-10 Soi Kingkaew 30 Kingkaew Road
Tt. Rachatheva, A. Bangpli
Samutprakarn
10540 Thailand
Tel. +66 2 7502737
Fax +66 2 6612355
atika@ji-net.com
www.kuka-roboter.de

Czech Republic

KUKA Roboter Austria GmbH
Organisation Tschechien und Slowakei
Sezemická 2757/2
193 00 Praha
Horní Počernice
Czech Republic
Tel. +420 22 62 12 27 2
Fax +420 22 62 12 27 0
support@kuka.cz

Hungary

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

USA

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

UK

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

Index