SOFTWARE

KRL (KUKA ROBOT LANGUAGE)

Programming User Messages

Expert Level

Length of documentation: 8 pages
PD Interleaf

Contents

1	**Programming user messages**	**3**
1.1	Definition of terms	3
1.1.1	Notification messages	3
1.1.2	Status messages	3
1.1.3	Acknowledgement messages	3
1.1.4	Dialog queries	3
1.1.5	Simulation	3
1.2	Application	4
1.2.1	Notification messages	5
1.2.2	Status messages	6
1.2.3	Acknowledgement messages	6
1.2.4	Dialog queries	6
1.2.5	Simulation	7
1.3	Sample program "MSG_DEMO.SRC"	7
1.4	Variables and declarations	8

1 Programming user messages

Within application programs and technology packages (KRL) it is possible to display freely definable messages on the KCP. A basic distinction is made here between notification messages, status messages, acknowledgement messages and dialog queries. The most important for this is a standardized interface between the kernel system and the user environment (GUI). The various message types require specific handling, a further distinction being made between messages with database access (TECH packages) and simple user messages without database access.

For commissioning and test purposes, the so–called simulation function provides a mechanism for simulating logic expressions by means of an AND operation with a variable that can be controlled directly from the user interface. This mechanism can be usefully employed above all in loop exit conditions.

1.1 Definition of terms

1.1.1 Notification messages

Simplest type of message with an informative character only. Execution of the current application program is not interrupted. The messages do not have to be acknowledged. It is also conceivable for them to be used for indicating the progress of a program in a special GUI window. Notification messages cannot be cleared from the KRL level.

1.1.2 Status messages

Status messages have an informative character and are reset quasi–automatically. Execution of the current application program can be conditionally interrupted.

Status messages cannot be acknowledged; they have to be explicitly canceled by the originator (application program). Only one active message at a time can be displayed.

1.1.3 Acknowledgement messages

Acknowledgement messages indicate disturbances in the progress of a program. Only after the active message has been explicitly acknowledged can the program be resumed. The message is automatically canceled when it is acknowledged.

1.1.4 Dialog queries

By means of dialog queries, the operator is prompted to choose between freely definable alternatives. The result of the operator input is stored in a corresponding KRL variable. With the operator input the dialog is automatically closed and the interrupted program is continued.

1.1.5 Simulation

Similar to the familiar simulation key in WAIT FOR instructions, the simulation key is automatically displayed. Pressing the simulation key clears the corresponding message and sets an associated boolean KRL variable to FALSE. Program execution is not interrupted. This simulation mechanism can be used for specifically exiting loops.

1.2 Application

The interface between the KRL application program and the user environment is represented by the so−called MSG structure. To enable it to cater for all possible applications, the MSG structure is of rather complex configuration.

STRUC ***MSG_T***

BOOL *VALID*

Triggers the message (TRUE). VALID is treated differently depending on the type of message (see Sections 1.1.1 to 1.1.4).

BOOL *RELEASE*

causes a status message to be cleared.

MSG_TYP *TYP,*

Definition of the message type

Possible values:	*#NOTIFY*	notification message
	#STATE	status message
	#QUIT	acknowledgement message
	#DIALOG	dialog query

CHAR *MODUL[12]*

Text variable containing an unambiguous module identifier (module key) serving the purpose of database access. If MODUL is a blank string, KEY is interpreted as the message text. If KEY contains a wildcard for PARAM, the two of them are combined. If MODUL is not blank and no entry can be found in the database with a corresponding key, then KEY will be output as the message text, though KEY cannot be combined with a parameter.

CHAR *KEY[40]*

Text variable containing an unambiguous key to the message text of the database. If MODUL is a blank string or is not found in the database, it is displayed in the message window under originator and the KEY is interpreted directly as the message text.

MSG_PRM_TYP *PARAM_TYP*

Type − definition of PARAM

Possible values:	*#VALUE*	numeric value or text without database access. PARAM is combined directly with the key.
	#WORDS	$MSG.PARAM[] is ignored.
	#KEY	database key; if there is no corresponding entry in "MODUL", the key is taken as the parameter string.

CHAR *PARAM[20]*

Additional parameter that is displayed in combination with the actual message text. The content of PARAM is interpreted as a function of PARAM_TYP.

CHAR *DLG_FORMAT[70]*

Format string defining the softkey labels for dialog queries. The individual softkeys are separated by "|". Up to 7 softkeys can be assigned, each of them representing a possible dialog response. It is important to remember that each softkey label may be no more than about 10 characters long and that only the required number of softkeys is defined in each case. The possible responses should have a meaningful relation to the message text:
e.g.: MSG.MODUL= "PART1.SRC"
MSG.KEY= "Repeat ?"
MSG.DLG_FORMAT= "Yes|No"
The softkeys are dynamically inserted into the softkey bar at unassigned positions.

INT *ANSWER*

Supplies the result of a dialog query. The returned value indicates which softkey has been pressed. The softkeys are counted in ascending order as defined in the format string DLG_FORMAT, starting at 1. In the example given above, pressing the softkey "No" would therefore return ANSWER =2.

Default setting of the MSG structure:

DECL MSG_T *$MSG_T=*
{ VALID FALSE, RELEASE FALSE ,TYP #NOTIFY, MODUL[] " ", KEY[] " ", PARAM_TYP #VALUE, PARAM[] " ", DLG_FORMAT[] " ", ANSWER }

The simulation mechanism works with a very simple interface implemented by means of the following global variables:

BOOL *$LOOP_CONT*

Simulation result that can be used as a loop exit condition. $LOOP_CONT goes FALSE as soon as the simulation key is pressed. It must be switched to TRUE before simulation is triggered by $LOOP_MSG.

CHAR *$LOOP_MSG[60]*

Triggers the simulation as soon as the value is not equal to a blank string and displays the text contained together with the simulation key.
It must be reset to a blank string in order to end the simulation.

Each type of message and the simulation mechanism necessitates a specific sequence in the application program (handshake). This chronological sequence is represented in Sections 1.1.1 – 1.1.5.

1.2.1 Notification messages

(1) Fill the $MSG_T structure.

(2) The message is triggered by setting $MSG_T.VALID = TRUE.

(3) The message is now displayed.

(4) Wait for $MSG_T.VALID==FALSE.

(5) Continue with the next message.

1.2.2 Status messages

(1) Fill the $MSG_T structure.

(2) Set $MSG_T.RELEASE = FALSE.

(3) The message is triggered by setting $MSG_T.VALID = TRUE.

(4) The status message is now displayed.

G

G

G

(5) Clear the status message by setting $MSG_T.RELEASE=TRUE.

(6) The message is no longer visible.

(7) Wait for $MSG_T.VALID== FALSE.

(8) Continue with the next message.

1.2.3 Acknowledgement messages

(1) Fill the $MSG_T structure.

(2) Stop program execution.

(3) The message is triggered by setting $MSG_T.VALID = TRUE.

(4) The acknowledgement message is now displayed.

G

G

G

(5) The acknowledgement message can be cleared if required by setting $MSG_T.RELEASE=TRUE.

(6) When the operator acknowledges the message, $MSG_T.VALID is automatically reset to FALSE.

(7) As soon as $MSG_T.VALID goes FALSE, program execution can be resumed.

(8) Continue with the next message.

1.2.4 Dialog queries

(1) Stop program execution.

(2) Fill the $MSG_T structure.

(3) The message is triggered by setting $MSG_T.VALID = TRUE.

(4) The dialog query is now displayed.

G

G

G

(5) The dialog query can be cleared if required by setting MSG.RELEASE=TRUE.

(6) When the operator acknowledges the message, $MSG_T.VALID and $MSG_T.RELEASE are automatically reset to FALSE.

(7) As soon as $MSG_T.VALID goes FALSE, the result of the query can be retrieved from $MSG_T.ANSWER and program execution can be resumed.

(8) Continue with the next message.

1.2.5 Simulation

(1) Set $LOOP_CONT=TRUE.

(2) The simulation is triggered by setting $LOOP_MSG= "message text".

(3) The simulation key and the message text are now displayed.

(4) The simulation can be actively ended by setting $LOOP_MSG[] =" ".

G

G

G

(5) As soon as the operator presses the simulation key, $LOOP_CONT is automatically set to FALSE.

(6) The simulation is ended by filling $LOOP_MESSAGE[] with BLANKS.

(7) The simulation message and the softkey "SIMULATION" are no longer displayed.

(8) Continue with the next simulation or message.

1.3 Sample program "MSG_DEMO.SRC"

```
DECL INT ANSWER
DECL INT OFFSET
DECL STATE_T STATE
DECL MSG_T EMPTY_MSG
EMPTY_MSG={MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] " ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0}
;----------ACKNOWLEDGEMENT MESSAGE----------
$MSG_T=EMPTY_MSG ;reinitialization
$MSG_T.MODUL[]=" "
$MSG_T.KEY[]="DEMO: QUIT-MESSAGE + %1"
$MSG_T.PARAM[]="PARAM"
$MSG_T.PARAM_TYP=#WORDS
$MSG_T.TYP=#QUIT
$MSG_T.VALID=TRUE ;trigger
WHILE $MSG_T.VALID ;wait for acknowledgement
WAIT SEC 0.05
ENDWHILE
;----------DIALOG QUERY----------
$MSG_T=EMPTY_MSG ;reinitialization
$MSG_T.MODUL[]=" "
$MSG_T.KEY[]="DEMO: DIALOG + %1"
$MSG_T.PARAM[]="PARAM"
$MSG_T.PARAM_TYP=#KEY
$MSG_T.TYP=#DIALOG
$MSG_T.DLG_FORMAT[]="A|B|C"
$MSG_T.VALID=TRUE ;trigger
WHILE $MSG_T.VALID ;wait for answer
```

```
WAIT SEC 0.05
ENDWHILE
ANSWER=$MSG_T.ANSWER
;------------------NOTIFICATION MESSAGE-----------------
$MSG_T=EMPTY_MSG ;reinitialization
$MSG_T.MODUL[]=" "
$MSG_T.KEY[]="DEMO: DIALOG ANSWER = %1" ;"%1" symbolizes a wildcard for the parameter string
OFFSET=0
SWRITE($MSG_T.PARAM[],STATE,OFFSET,"%d",ANSWER)
;formated output of the preceding
;dialog answer in the parameter string
$MSG_T.PARAM_TYP=#VALUE ;wait in order to reliably display the message
$MSG_T.VALID=TRUE ;trigger
WHILE  $MSG_T.VALID
WAIT SEC 0.05
ENDWHILE
;------------------- SIMULATION----------------------
$LOOP_CONT=TRUE              ;default setting
$LOOP_MSG[]="HELLO I'm the SIMULATION of $IN[10] !" ;trigger
WHILE  $LOOP_CONT or $IN[10]  ;simulation of the exit condition $IN[10]==TRUE
ENDWHILE
$LOOP_MSG[]="                                                  "
END
```

1.4 Variables and declarations

Implemented in $operate.dat and $option.dat at the control level.

```
ENUM MSG_TYP NOTIFY, STATE, QUIT, DIALOG
ENUM MSG_PRM_TYP VALUE, WORDS, KEY
STRUC MSG_T BOOL VALID, BOOL RELEASE, MSG_TYP TYP, CHAR MODUL[12], CHAR KEY[40], MSG_PRM_TYP PARAM_TYP, CHAR PARAM[20], CHAR  DLG_FORMAT[70], INT ANSWER
DECL MSG_T  $MSG_T={ VALID FALSE, RELEASE FALSE, TYP #NOTIFY, MODUL[] " ", KEY[] " ", PARAM_TYP #VALUE, PARAM[] " ", DLG_FORMAT[] " ", ANSWER 0, }
BOOL $LOOP_CONT=FALSE
CHAR $LOOP_MSG[60]
```