

The variables in the CNC control system SINUMERIK Operate

Zmienne w układzie sterowania CNC SINUMERIK Operate

BOGUSŁAW PYTLAK *

DOI: <https://doi.org/10.17814/mechanik.2017.3.44>

The types of variables occurring in the control system SINUMERIK Operate are presented. The system variables and user variables: R-parameters, global R parameters, GUD variables, PUD variables and LUD variables, with programming examples are described. The indirect programming with using of variables is discussed. The example of program for thread milling is prepared, which shows the differences of using of different variable types and advantages this solutions.

KEYWORDS: variable types, CNC control system, indirect programming, parametric programming

Parametric programming in the SINUMERIK control system is based on the use of variables that, combined with computational functions and control structures, provide high flexibility in programs, sub-programs and cycles. The purpose of this article is to familiarize SINUMERIK users with the different types of variables and how they can be used in processing programs. Variables in the SINUMERIK control system can be divided into: system and user [1].

System variables

System variables have defined meaning. They use system software, they can also be read and written in user programs. Despite the predefined meaning of system variables, it is possible to change their properties by so-called *redefinition*. System variables allow to parameterize the control system and provide access to the current state of control, machine, and processing.

These variables can be subdivided into preprocessing variables and main run variables [1]. Preprocessing variables are read and written when reading (interpret) the program block, in which they were programmed. They do not trigger preprocessing stops. In turn, the main run variables are read and written at the time of execution of the program block, in which they were programmed. These variables include variables that are programmed in synchronous actions, do not trigger preprocessing stops variables, and variables the value of which is determined in the preprocessing sequence, but are only written in the main run.

The standard denotation of system variables consists of \$ + first letter (type of data) + second letter (coverage). System variables of the preprocessing process are as follows [1]: \$M - Machine data, \$\$ - Set data, \$T - Tool data, \$P - Programmable values, \$C - Variable cycles ISO, \$O - Option data, R - Computing parameters. System variables of the main run are [1]: \$\$M - Machine data, \$\$\$ - Set data, \$A - Current main data, \$V - Servo data, \$R - Primary variables. Main run variables are distinguished by the additional \$ sign. It is reserved for system variables and must not be used in user variables.

There are different areas of variables: N - Global variables (NCK), C - Channel variables, A - Axis variables. For a detailed list and description of system variables, see the SINUMERIK control system documentation [2].




a)

Variable	Format	Value
SP_TOOLR	D	10
/Channel/State/actToolRadius[u1, 1]	D	10
SP_TOOLNO	D	5
/Channel/State/actTNumber[u1, 1]	D	5
SP_TOOL	D	1
/Channel/State/actDNumber[u1, 1]	D	1

b)

* Dr inż. Bogusław Pytlak (boguslaw.pytlak.ext@siemens.com) – Siemens Sp. z o.o.

Fig. 1. Windows: a) NC/PLC variable area, b) search and selection of NC/PLC variables

The area  →  (fig. 1a) is provided for displaying system variables. A variable can be displayed on a given line by entering its name directly. One can also use the button  that opens the search window and selects variables (fig. 1b).

In the SINUMERIK control system, most system variables occur under two names. The first one, such as \$P_TOOLR, is the radius of the active tool and can be used in programs. The other, e.g. /Channel/State/actToolRadius[u1,1], is the same variable and is useful in system software, such as COM files defining dialogs. Designation u1 specifies the channel number; for channel 2 it will be u2 and so on.

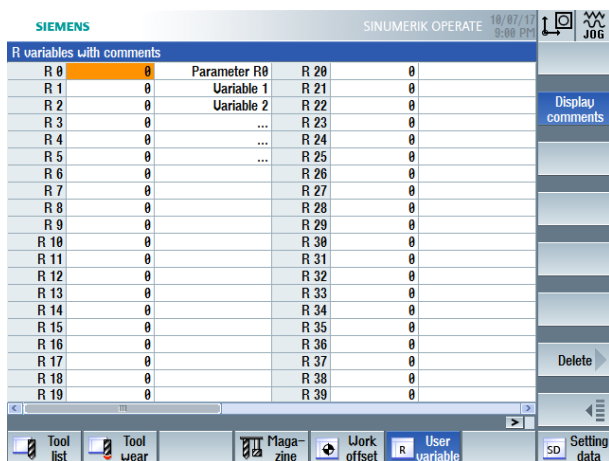
User variables

User variables include predefined variables in the system and variables defined by the user itself. The type of predefined user variables is predetermined and their number is determined in the corresponding machine data. These variables include: R parameters, global R-parameters RG, Link variables. User-defined variables are created on the system permanently or only at runtime - the user has full control over them. These variables can be grouped according to their scope of use: Global User Data (GUD), Program User Data (PUD), and Local User Data (LUD).

• **R-parameters** are among the most familiar predefined user variables. They were already in the early versions of the SINUMERIK control system and for the longest time they were the only way to parametrize part of programs and transfer parameters to cycles. Currently, with the availability of other types of user variables, the meaning of R-parameters is smaller, although they still apply. R-parameters are an array of Real variables.

For better readability of R-parameters, an additional description of the significance of the R parameter is often included in the part program, e.g. in the form of a comment in the part program: R10 = 200 ;coordinate w of axis X. In the HMI version of Operate v.4.7, one can see an additional column with their description (fig. 2).

Formally, referencing a given R-parameter should be a reference to an array variable, e.g. R[10]=..., but for historical reasons it is permissible to simplify the record, e.g. R10=... R-parameters are channel variables, which means that each channel has an identical set of R parameters.

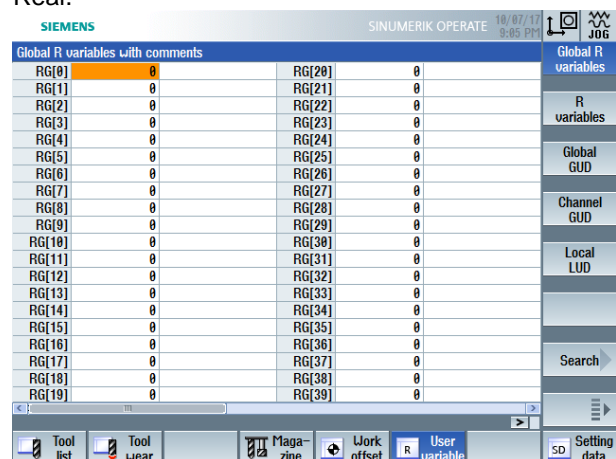


R	Value	Description	Value
R 0	0	Parameter R0	0
R 1	0	Variable 1	0
R 2	0	Variable 2	0
R 3	0	...	0
R 4	0	...	0
R 5	0	...	0
R 6	0	...	0
R 7	0	...	0
R 8	0	...	0
R 9	0	...	0
R 10	0	...	0
R 11	0	...	0
R 12	0	...	0
R 13	0	...	0
R 14	0	...	0
R 15	0	...	0
R 16	0	...	0
R 17	0	...	0
R 18	0	...	0
R 19	0	...	0

Fig. 2. R-Parameters window with additional column of comments

• **Global R-parameters RG.** For the exchange of information between channels, you can use the global R-parameters RG available in HMI version of Operate v.4.7 (fig. 3) or GUD global variables. Similar global R-parameters existed in the SINUMERIK 840C control system, where R-parameters above R700 were considered global. Global R-parameters RG should be referred to as an array variable, e.g. RG[10]=...

• **Link variables** are used to cyclical exchange information between different NCUs connected to the network using the NCU-Link function. Link variables are global - they can be written and read by machining programs on all connected NCUs. For only one NCU, Link variables can be used as additional global variables for the user. The following variables are of type Link: \$A_DLB[<i>] - byte, \$A_DLW[<i>] - word, \$A_DLD[<i>] - double word, \$A_DLR[<i>] - Real, where <i> denotes the variable index. This index changes from 0 to 1 for the byte, 2 for the word, 4 for the double word, and 8 for Real.



Global R variables with comments	Value	Value
RG[0]	0	0
RG[1]	0	0
RG[2]	0	0
RG[3]	0	0
RG[4]	0	0
RG[5]	0	0
RG[6]	0	0
RG[7]	0	0
RG[8]	0	0
RG[9]	0	0
RG[10]	0	0
RG[11]	0	0
RG[12]	0	0
RG[13]	0	0
RG[14]	0	0
RG[15]	0	0
RG[16]	0	0
RG[17]	0	0
RG[18]	0	0
RG[19]	0	0
RG[20]	0	0
RG[21]	0	0
RG[22]	0	0
RG[23]	0	0
RG[24]	0	0
RG[25]	0	0
RG[26]	0	0
RG[27]	0	0
RG[28]	0	0
RG[29]	0	0
RG[30]	0	0
RG[31]	0	0
RG[32]	0	0
RG[33]	0	0
RG[34]	0	0
RG[35]	0	0
RG[36]	0	0
RG[37]	0	0
RG[38]	0	0
RG[39]	0	0

Fig. 3. Window of global R-parameters RG

• **Global user variables GUD** are variables that all programs have access to. These are the ones that most often supplant popular R-parameters. GUD variables can be of any type and have any name; they are in non-volatile NC memory. These characteristics determine their great utility. In the SINUMERIK control system, they are often used by machine manufacturers, for which the MGUD.DEF definition file is provided, and the UGUD.DEF definition file is then assigned to the users. The GUD variables are defined as follows:

```
DEF <range> <preprocessing_stop> <access_rights> <data_class> <type> <physical_unit> <limit_values>
<name>[<value_1>, <value_2>, <value_3>] = <init_value>
```

where:

<range> - variable range: global NCK, channel CHAN;
 <preprocessing_stop> - stopping moment of preprocessing run: write, read, write + read variable;
 <access_rights> - variable access level: for NC program and control panel;

<data_class> - specifying the data class to which the variable belongs (828D sl only);

<type> - variable type;

<physical_unit> - the unit in which the variable will be expressed (REAL and INT only);

<limit_values> - lower LLI and upper ULI values of the variable (REAL, INT and CHAR only);

<name> – variable name (can not be used as a command name for the SINUMERIK programming language and variables already defined);
 [<value_1>, <value_2>, <value_3>] – array dimensions for array variable;
 <init_value> – value that the variable takes at initialization.

In the SINUMERIK control system, variables of the following types can be defined: INT (Integer), REAL, BOOL, CHAR, STRING [<max_length>], AXIS, FRAME. Most of these types occur in computer programming languages, but the AXIS and FRAME types that only appear in the SINUMERIK programming language require clarifications.

AXIS is an axial type, meaning that any axis of the machine can be assigned to a variable of this type. In turn, a coordinate transformation (TRANS, ROT, MIRROR, SCALE) can be assigned to FRAME variable or a combination thereof.

When discussing types of variables, it is important to mention the great recommendation of the SINUMERIK programming language, which is automatic conversion of variables (if possible). Currently, it is possible to program array variables up to three-dimensional, except for STRING variables that may be two-dimensional (the third dimension is contained in STRING itself, which is an array of CHAR variables). Example syntax for global definitions file for UGUD.DEF variables is as follows:

```
DEF CHAN SYNW APRP 1 APWP 1 REAL PHU 1 LLI 0
ULI 500 DIMENSION[10] = REP(12) ; full definition of GUD
variable
;CHAN - channel variable,
;SYNW - stopping the preprocessing run on save,
;APRP 1 - NC program readings at machine manufacturer
level,
;APWP 1 - NC program record at machine manufacturer
level,
;REAL - real type,
;PHU 1 - unit of length, mm or inch,
;LLI 0 - lower limit is equal 0,
;ULI 500 - upper limit is equal 500,
;DIMENSION[10] - array variable with 10 elements,
;REP(12) - initialize all array elements from value 12,
DEF CHAN REAL VARIABLE_1 ; simplified definition
DEF CHAN INT VARIABLE_2=2
DEF CHAN BOOL VARIABLE_3=FALSE
DEF CHAN CHAR VARIABLE_4="a" ; or 97 acc. to ASCII
code
DEF CHAN STRING[10] VARIABLE_5="TEKST"
DEF CHAN AXIS VARIABLE_6=(X)
DEF CHAN FRAME VARIABLE_7
```

M30

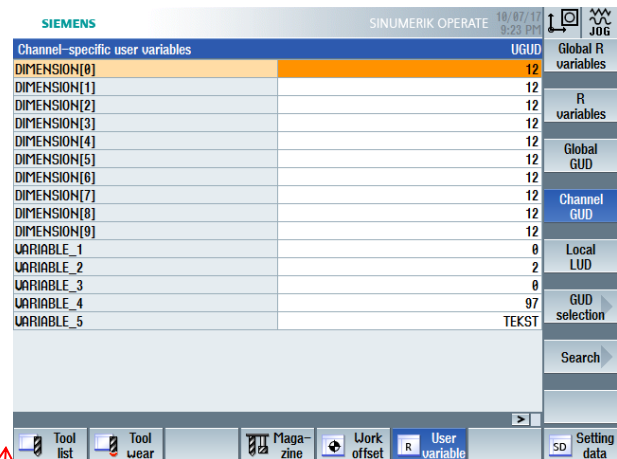


Fig. 4. Window Channel User Variables GUD

As it can be seen in fig. 4, the AXIS and FRAME variables are not displayed in the user variable window, and the FRAME variable can not be assigned an initialization value. This can be done later, e.g. in a part program, e.g. CHANGE=CTTRANS(X,100,Y,200,Z,300):CROT(Z,45), which means shift and rotation of the coordinate system. The order of definition of variables in a DEF file determines the order of displaying them in the GUD window (fig. 4).

- **Local user variables LUD** are defined within a given program (sub-program) of the processing. They are created at the time of execution of a given program and are deleted after its completion. Local variable definitions are made at the beginning of the part program using the DEF command (before EXTERN can be used). The DEF syntax is as follows:

```
DEF <type> <physical_unit> <limit_values>
<name>[<value_1>, <value_2>, <value_3>] = <init_value>
```

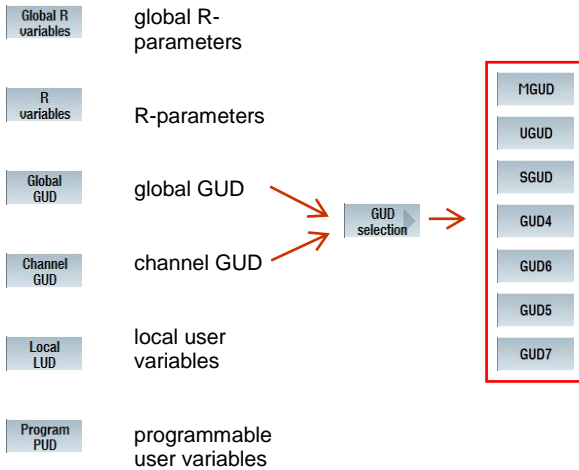
There may be several variables of the same type defined within a single DEF definition (only separated by commas). Each new variable type must be defined in the new block with a new DEF command. Normally, the LUD variables defined in the main program are not visible to the subroutine called from the main program. To become visible, you must enable PUD user programmable variables. To do this, set the machine data \$MN_LUD_EXTENDED_SCOPE=1. Then an additional button with PUD variables appears.

- **Programmable user variables PUD** are defined in the main program and all sub-programs of that program are accessed. This allows to pass information between the main program and the sub-program call. Previewing and modifying user variables is done under

User variable

Parameter →

The next buttons correspond to the individual variables:



- **Redefine variables.** After defining user variables, as with system variables, one can change their attributes using the REDEF redefine command. In one REDEF command, only one variable attribute can be changed:

REDEF <name> <attribute>

In the same command, the moment of the initialization of the variable can be specified (i.e. setting of the variable to the initialization value: INIPO - after Power On, INIRE - after Reset, INICF - after NEWCONFIG command, PRLOC - initialization of the variable, programmatically adjustable by the reset only if it has been changed from a part program).

For example: the set variable was initialized with \$SC_THREAD_START_ANGLE=0, which means a 0° start angle for thread cutting. Programming a command e.g. SF=45 changes the value of this data to 45°. By setting the reset time using PRLOC after resetting to a given setting, \$SC_THREAD_START_ANGLE is again set to 0.

- **Obtaining information about variables.** When defining another variable, it may turn out that the variable is already in the control circuit. It is recommended that the name of the user variable starts with the underscore character, but practice indicates that it is better not to use the underscore + axis name, e.g. _A, as they are internally used in machining cycles, and sometimes cause definition duplication problems.

It can be used the ISVAR(<variable_name>) command to check whether the variable entered in <variable_name> exists to avoid double-definition of the same variable, or for example to try to write or read a non-existent variable. The result of this check is best written to another BOOL variable. In addition to verifying the existence of a given variable, you can get much more information about it in the SINUMERIK control system: GETVARPHU, GETVARAP, GETVARLIM, GETVARDFT and GETVARTYP. These commands, like the ISVAR command, return the specified numeric values that are best written to the INT and REAL variables (fig. 5).

```
DEF BOOL IS_VARIABLE
DEF INT
UNIT,ACCESS_R,ACCESS_W,LIMIT_LO,LIMIT_UP,
STANDARD_VALUE,TYPE
DEF REAL LIM_L,LIM_U,VAL_STAND
```

```
IS_VARIABLE=ISVAR("DIMENSION") ;checking the exist-
ence of a variable
UNIT=GETVARPHU("DIMENSION") ;unit reading
ACCESS_R=GETVARAP("DIMENSION","RP") ;reading the
read permission
ACCESS_W=GETVARAP("DIMENSION","WP") ;reading
the write permission
LIMIT_LO =GETVARLIM("DIMENSION","L",LIM_L) ;lower
limit reading
LIMIT_UP=GETVARLIM("DIMENSION","U",LIM_U) ;upper
limit reading
STANDARD_VALUE=GETVARDFT("DIMENSION",
VAL_STAND,3) ;reading the value of an array element with
index 3
TYPE=GETVARTYP("DIMENSION") ;variable type reading
```

SIEMENS		SINUMERIK OPERATE	18.07.17	12:03 PM	MDA
Local user variables					Global R variables
IS_VARIABLE				1	R variables
UNIT				1	Global GUD
ACCESS_R				1	Channel GUD
ACCESS_W				1	Local LUD
LIMIT_LO				1	
LIMIT_UP				1	
STANDARD_VALUE				1	
TYPE				4	
LIM_L				0	
LIM_U				500	
VAL_STAND				12	

Fig. 5. Window of local user variables LUD with readings of variable DIMENSION

- **Indirect programming.** User variables are often used in so-called *indirect programming*. It involves using variables in extended addressing. For example: to program and activate the spindle rotation, the number of which specifies the variable NO_SPINDLE, simply enter the following program code:

```
DEF INT NO_SPINDLE=1
S[NO_SPINDLE]=2000 M[NO_SPINDLE]=3
```

Indirectly, one can also call sub-programs, for example, with the CALL command, where the path and program name (sub-program) can be specified with the STRING variable, for example, with an INT variable with program number:

```
DEF STRING[100] NAME
DEF INT NO_PROG=1
NAME="/_N_WKS_DIR/_N_EXAMPLE"<<NO_PROG<<
"_WPD/_N_EXAMPLE"<<NO_PROG<<"_MPF"
CALL NAME
```

This example uses the << string join operator, which allows to combine two STRING type strings and enclose this variable with the INT type. Type conversion is done automatically, as discussed earlier.

Another interesting option is to program G functions indirectly. For this purpose, information is used about which group G function is assigned and what position it occupies in the group. For a detailed description of the G function groups and positions occupied in these groups, see the documentation [3] in Chapter 17.4. For example, the table shows the G functions in group number 8.

TABLE. Group of functions G number 8: Zero offset adjustment

G Command	No	Meaning
G500	1	Turn off the zero offset (G54 ... G57, G505 ... G599)
G54	2	1 adjustable zero offset
G55	3	2 adjustable zero offset
G56	4	3 adjustable zero offset
G57	5	4 adjustable zero offset
G505	6	5 adjustable zero offset
...
G599	100	99 adjustable zero offset

To generate G54 zero offset and sequential, program the following:

```
DEF INT NO_OFFSET
NO_OFFSET=2
G[8]=NO_OFFSET ;call G54
STOPRE ;stop the preprocessing run
NO_OFFSET=NO_OFFSET+1 ;increase NO_OFFSET by 1
G[8]=NO_OFFSET ;call G55
STOPRE ;stop the preprocessing run
NO_OFFSET=NO_OFFSET+1 ;increase NO_OFFSET by 1
G[8]=NO_OFFSET ;call G56
;...
```

It is also possible to indirectly program the NC code using the EXECSTRING(<string_variable>). This command converts string <string_variable> to NC code, which is executed by control, e.g.:

```
EXECSTRING("S1000 M3")
EXECSTRING("G4 F10")
EXECSTRING("M5")
```

- **Parametric programming.** In the SINUMERIK control system, different types of variables can be used in machining programs. To illustrate the differences between individual programs, a simple example of milling a full inner thread of M52×5 in a 50 mm blank was developed. The thread center coordinates are X50 Y50. In the first variant the program is called. A fixed program that does not use variables (except for the radius of the active tool \$P_TOOLR). In this program, the first block G3 reaches the wall of the thread after a semicircle. In the second block G3 is threaded full threaded thread. In the last G3 block, the thread returns to the center of the thread after the semicircle:

```
T="THREAD CUTTER"
```

```
M6
```

```
G17 G54 G94
```

```
WORKPIECE(,"","BOX",112,0,-50,-80,0,0,100,100)
```

```
G90 G0 X50 Y50 Z5 D1 F100 S2000 M3
```

```
G3 X=IC(26-$P_TOOLR) Y50 Z=IC(-5/4) CR=(26-$P_TOOLR)/2
```

```
G3 X=IC(0) Y=IC(0) Z=IC(-60) I=AC(50) J=AC(50)
TURN=12
```

```
G3 X50 Y50 Z=IC(-5/4) CR=(26-$P_TOOLR)/2
```

```
G0 Z5
```

```
M2
```

In the second variant, the parameterization of the program was made using R-parameters. In this example, attention should be paid to the parameterization of the blank visible during simulation, so called. WORKPIECE, whose dimensions have also been parameterized. Similar parameters can be entered in other G code cycles, as well as ShopMill and ShopTurn. As you can see, the use of R-parameters requires an additional description of their meaning in the form of comments:

```
R0=50 ;coordinate of the center in axis X
```

```
R1=50 ;coordinate of the center in axis Y
```

```
R2=5 ;start height in axis Z
```

```
R3=52 ;diameter of diameter of milling thread
```

```
R4=5 ;thread pitch
```

```
R5=12 ;number of turns
```

```
T="THREAD CUTTER"
```

```
M6
```

```
G17 G54 G94
```

```
WORKPIECE(,"","BOX",112,R2-5,-R4*(R5-2),-80,R0-50,R1-50,R0+50,R1+50)
```

```
G90 G0 X=R0 Y=R1 Z=R2 D1 F100 S2000 M3
```

```
G3 X=IC(R3/2-$P_TOOLR) Y=R1 Z=IC(-R4/4) CR=(R3/2-$P_TOOLR)/2
```

```
G3 X=IC(0) Y=IC(0) Z=IC(-R4*R5) I=AC(R0) J=AC(R1)
TURN=R5
```

```
G3 X=R0 Y=R1 Z=IC(-R4/4) CR=(R3/2-$P_TOOLR)/2
```

```
G0 Z=R2
```

```
STOPRE
```

```
R0=0 R1=0 R2=0 R3=0 R4=0 R5=0
```

```
M2
```

In the next example, LUD user variables defined at the beginning of the program are used for local parameters (e.g. GUD variables defined in UGUD.DEF file). This case is much easier to read because after the name of the variable you can see what it means:

```

DEF REAL COOR_X, COOR_Y, COOR_Z,DIAMETER,
PITH,TURN_NO
COOR_X=50 ;coordinate of the center in axis X
COOR_Y=50 ;coordinate of the center in axis Y
COOR_Z=5 ;start height in axis Z
DIAMETER=52 ;diameter of diameter of milling thread
PITH=5 ;thread pitch
TURN_NO=12 ;number of turns
T="THREAD CUTTER"

```

M6

G17 G54 G94

```

WORKPIECE(",","BOX",112,COOR_Z-5,-PITH*(TURN_NO-
2),-80, COOR_X-50,COOR_Y-
50,COOR_X+50,COOR_Y+50)

```

G90 G0 X=COOR_X Y=COOR_Y Z=COOR_Z D1 F100 S2000 M3

G3 X=IC(DIAMETER/2-\$P_TOOLR) Y=COOR_Y Z=IC(-PITH/4)

CR=(DIAMETER/2-\$P_TOOLR)/2

G3 X=IC(0) Y=IC(0) Z=IC(-PITH*TURN_NO)

I=AC(COOR_X)

J=AC(COOR_Y) TURN=TURN_NO

G3 X=COOR_X Y=COOR_Y Z=IC(-PITH/4)

CR=(DIAMETER/2-\$P_TOOLR)/2

G0 Z=COOR_Z

In the fourth example, the THREAD_MILLING parametric sub-program was created in the Subprograms directory (need to use the EXTERN command), where the thread milling process itself takes place. The rest of the commands needed to perform the machining and calling the parametric subroutine THREAD_MILLING are in the main program.

In the example for the THREAD_MILLING sub-program, the parameters are passed in three ways: by means of fixed values, via R parameters, and via LUD local variables.

EXTERN

```

THREAD_MILLING(REAL,REAL,REAL,REAL,REAL,REAL)

```

```

DEF REAL COOR_X, COOR_Y, COOR_Z,DIAMETER,
PITH,TURN_NO

```

COOR_X=50 R0=50 ;coordinate of the center in axis X

COOR_Y=50 R1=50 ;coordinate of the center in axis Y

COOR_Z=5 R2=5 ;start height in axis Z

DIAMETER=52 R3=52 ;diameter of diameter of milling thread

PITH=5 R4=5 ;thread pitch

TURN_NO=12 R5=12 ;number of turns

T="THREAD CUTTER"

M6

G17 G54 G94 D1 F100 S2000 M3

```

WORKPIECE(",","BOX",112,COOR_Z-5,-PITH*(TURN_NO-
2),-80, COOR_X-50,COOR_Y-
50,COOR_X+50,COOR_Y+50)

```

THREAD_MILLING(50,50,5,52,5,12) ;or

THREAD_MILLING (R0,R1,R2,R3,R4,R5) ;or

THREAD_MILLING(COOR_X,COOR_Y,COOR_Z,DIAMETER,PITH, TURN_NO)

R0=0 R1=0 R2=0 R3=0 R4=0 R5=0

M2

In turn, the content of the THREAD_MILLING sub-program is as follows:

```

PROC THREAD_MILLING (REAL _COOR_X,REAL
_COOR_Y,REAL _COOR_Z REAL _DIAMETER,REAL
_PITH,REAL _TURN_NO) SAVE

```

G90 G0 X=_COOR_X Y=_COOR_Y Z=_COOR_Z

G3 X=IC(_DIAMETER/2-\$P_TOOLR) Y=_COOR_Y Z=IC(-PITH/4) CR=(_DIAMETER/2-\$P_TOOLR)/2

G3 X=IC(0) Y=IC(0) Z=IC(-_PITH*_TURN_NO)

I=AC(_COOR_X) J=AC(_COOR_Y) TURN=_TURN_NO

G3 X=_COOR_X Y=_COOR_Y Z=IC(-_PITH/4)

CR=(_DIAMETER/2-\$P_TOOLR)/2

G0 Z=_COOR_Z

RET

In the parametric sub-program THREAD_MILLING, an additional underline was added to the sub-program to distinguish the main program from the sub-program names. In order for the THREAD_MILLING sub-program to become a cycle, simply copy it into the User Cycles directory and restart the control system. From this point on, the main program will no longer need the EXTERN command when the cycles do not require it.

REFERENCES

1. „SINUMERIK 840D sl/828D. Job planning. Programming manual". 10/2015, 6FC5398-2BP40-5NA3.
2. „SINUMERIK 840D sl/828D. System variables. Parameter Manual". 10/2015, 6FC5397-6AP40-5BA3.
3. „SINUMERIK 840D sl/828D. Fundamentals. Programming manual". 10/2015, 6FC5398-1BP40-5NA3.

