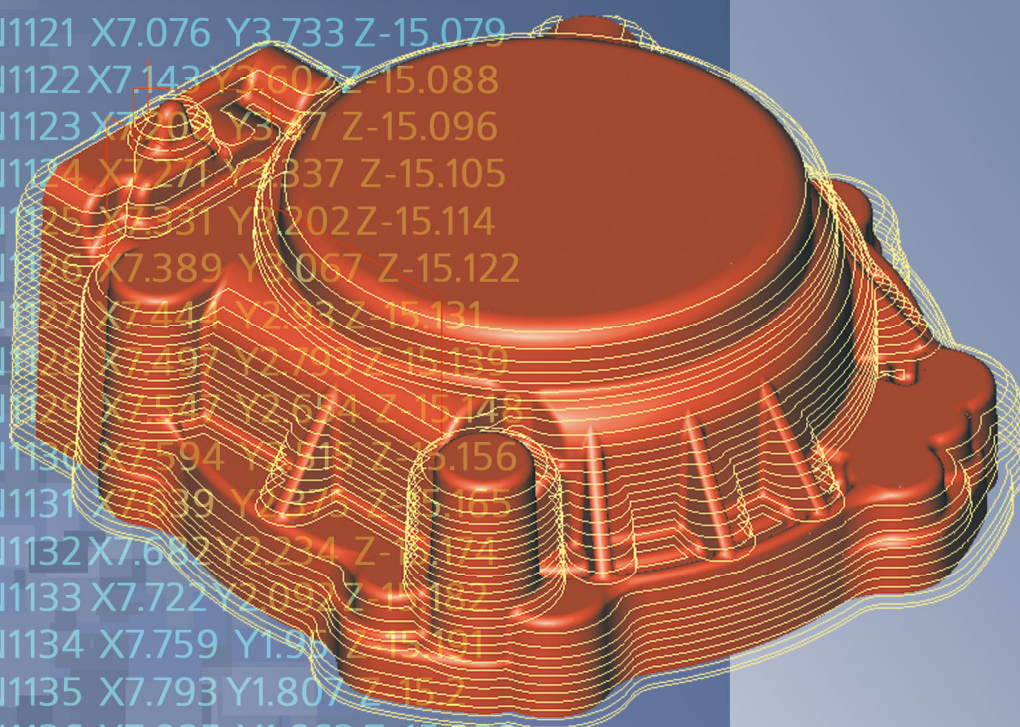




Post Processors Generator

N1120 X7.006 Y3.863 Z-15.07
 N1121 X7.076 Y3.733 Z-15.079
 N1122 X7.143 Y3.602 Z-15.088
 N1123 X7.210 Y3.477 Z-15.096
 N1124 X7.271 Y3.337 Z-15.105
 N1125 X7.331 Y3.202 Z-15.114
 N1126 X7.389 Y3.067 Z-15.122
 N1127 X7.447 Y2.932 Z-15.131
 N1128 X7.497 Y2.793 Z-15.139
 N1129 X7.547 Y2.654 Z-15.148
 N1130 X7.594 Y2.515 Z-15.156
 N1131 X7.639 Y2.375 Z-15.165
 N1132 X7.682 Y2.234 Z-15.174
 N1133 X7.722 Y2.093 Z-15.182
 N1134 X7.759 Y1.951 Z-15.191
 N1135 X7.793 Y1.807 Z-15.2
 N1136 X7.825 Y1.663 Z-15.208



CONTENTS

CONTENTS	2
INTRODUCTION	5
THE PURPOSE OF THE POSTPROCESSORS GENERATOR	5
THE FILES SET OF THE POSTPROCESSORS GENERATOR.....	5
1 THE COMMON ORGANIZATION OF THE WORK	7
1.1 THE PRINCIPLE OF POSTPROCESSOR OPERATION	7
1.2 THE MAIN WINDOW	9
1.2.1 The main menu.....	10
1.2.2 The main toolbar.....	12
1.2.3 The process indicator	13
1.2.4 System settings	13
1.2.5 Editor settings.....	15
1.2.6 Defining the data about the NC-machine and CNC-system	15
1.2.7 Postprocessor parameters inquiry while the first using.....	17
1.2.8 The block structure and format definition (Register list forming).....	20
1.2.9 The masks for the machining commands translation.....	22
1.2.10 The programs for the CLData commands processing	23
1.2.11 Subprograms	24
1.2.12 The command processing programs compilation	24
1.2.13 The work with the files of technological commands	25
1.2.14 The test NC-code generation	26
1.2.15 Programs debugging	26
1.2.16 Reinterpretations programs definition	27
2 MASKS.....	29
2.1 BASIC DEFINITIONS	29
2.2 MASK STRUCTURE	29
2.2.1 Mask element	29
2.2.2 Registers in the masks	30
2.2.3 Modifiers	31
2.2.4 Expressions	31
2.2.5 Nested mask.....	32
2.2.6 The separators of the mask elements	33
2.2.7 Variables assignment from the mask.....	33
2.2.8 Deferred masks	33
2.3 MASK MANAGEMENT	36
2.3.1 Mask Switches.....	36
2.3.2 The transformation of a Mask to the Subprogram	36
2.3.3 The interactive to create the masks.....	36
3 LANGUAGE DESCRIPTION	38
3.1 BASIC DEFINITIONS	38
3.1.1 Conditional indications.....	38
3.1.2 The processing programs of the technological commands, the comments in the programs	38
3.1.3 Subprograms	39
3.1.4 The language statement conception.....	39
3.1.5 The set of symbols	39

3.1.6	The variables	40
3.1.7	Arrays	40
3.1.8	Mathematical expressions and functions	41
3.1.9	Predefined variables	42
3.1.10	Predefined functions	43
3.2	OPERATORS	45
3.2.1	The processing program start operator: PROGRAM	45
3.2.2	The statement of assignment =	45
3.2.3	The output statement PRINT	46
3.2.4	The input statement INPUT	46
3.2.5	Conditional statement IF	47
3.2.6	Statement of the multiconditional execution CASE	49
3.2.7	JUMP statement	49
3.2.8	The cycle statement FOR	50
3.2.9	The cycle statement REPEAT	50
3.2.10	The cycle statement WHILE	51
3.2.11	Composite statement BEGIN ... END	51
3.2.12	Statement to call a subprogram CALL	52
3.2.13	The statement of the subprogram start SUB	52
3.2.14	The statement of the subprogram end SUBEND	52
3.2.15	The statement of the procedure start PROC	53
3.2.16	The RETURN from a procedure statement	53
3.2.17	The block output statement OUTBLOCK	54
3.2.18	The block forming statement FORMBLOCK	54
3.2.19	Statement of direct output into the block OUTPUT	55
3.2.20	The replace statement of the substring in a string REPLACE	55
3.2.21	The statement to form the block by mask MASK	56
4	APPENDICES	57
4.1	THE CLDATA DICTIONARY (THE DICTIONARY OF THE TECHNOLOGICAL COMMANDS)	57
4.2	FORMAT OF THE TECHNOLOGICAL COMMANDS	60
4.2.1	Part number	60
4.2.2	Ending record	60
4.2.3	Postprocessor function	60
4.2.4	Commentaries	63
4.2.5	Linear transition	63
4.2.6	Displacement along the circle	64
4.2.7	Working plane	64
4.2.8	Original point	65
4.2.9	Return to original position	65
4.2.10	The point of tool change	65
4.2.11	Table rotation	66
4.2.12	Tool compensation	67
4.2.13	Tool loading	68
4.2.14	Tool selection	68
4.2.15	Spindle	69
4.2.16	Stop	69
4.2.17	Auxiliary stop	69
4.2.18	Cooling	69
4.2.19	Feedrate	70
4.2.20	Rapid feedrate	70
4.2.21	Pause	70

4.2.22	Absolute or relative coordinate system assuming.....	71
4.2.23	Original coordinates.....	71
4.2.24	Canceling and recovering a cycle.....	71
4.2.25	Drilling cycle a type G81.....	72
4.2.26	Drilling cycle a type G82.....	72
4.2.27	Drilling cycle a type G84.....	73
4.2.28	Drilling cycle a type G85.....	73
4.2.29	Drilling cycle a type G86.....	74
4.2.30	Drilling cycle a type G87.....	74
4.2.31	Drilling cycle a type G88.....	75
4.2.32	Drilling cycle a type G89.....	76
4.2.33	Deep drilling with the full retraction of tool for removing a shaving.....	76
4.2.34	Deep drilling with drill retraction for removing the shaving.....	77
4.2.35	Threading	78
4.2.36	Palette changing.....	78
4.2.37	Head.....	78
4.2.38	Factors of a spline	79
4.2.39	The beginning of a spline phase of trajectory	79
4.2.40	Insertion.....	79
4.2.41	Optional skipping	79
4.2.42	Postprocessor printing	80

INTRODUCTION

THE PURPOSE OF THE POSTPROCESSORS GENERATOR

Postprocessors generator is an application for operating systems of the Windows™ family.

The purpose of the postprocessors generator is the generation of the postprocessor adjustment files to various NC-systems. These files are used by the run-time postprocessor system for the concrete NC-program generation.

It is necessary to perform the following steps to develop the postprocessor adjustment file:

- Define the data about the NC-machine and CNC-system;
- Describe the structure and the format of the block (form the list of registers);
- Design masks or programs to process technological commands;
- Save the postprocessor's tuning file;

The generation of new tuning files and the editing of existing files is allowed.

In addition to the work with data about NC-machine and CNC-system, there is the possibility of the examination the technological commands files and the trial generation of NC-programs in the environment of the postprocessors generator.

THE FILES SET OF THE POSTPROCESSORS GENERATOR

Inp.exe – executable module of postprocessors generator;

InpD.dll – executable system of the postprocessor;

***.spp** – the file with a data about concrete NC-system and with the sources of programs to process technological commands;

***.stc** – **SprutCAM** project files;

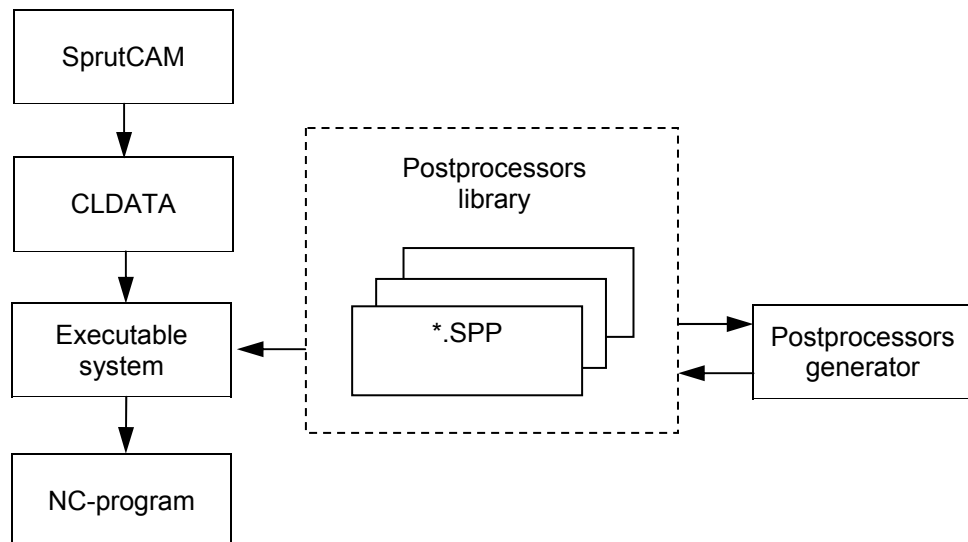
***.mcd** – the files of technological commands, linked with **SprutCAM** project files;

***.inp, *.ppp** – the postprocessor adjustment files for the previous version of the postprocessor generator;

1 THE COMMON ORGANIZATION OF THE WORK

1.1 THE PRINCIPLE OF POSTPROCESSOR OPERATION

Postprocessors generator allows to develop the postprocessor adjustment files for the different NC-systems (*.SPP files). An import of the files from the previous version of postprocessor (*.INP, *.PPP files) is available also. The postprocessor adjustment file contains the descriptions of all features to generate NC-program for the defined NC-system. The executable system of postprocessor uses this description to generate the NC-programs from the files of technological commands (*.MCD files), which can be produced, in turn, by **SprutCAM** system, for example.



To develop the postprocessor's tuning file, it's necessary to define the data about the NC-machine and NC-system, to describe the structure and the format of the block and to fill the masks or to design the programs to process the technological commands.

The data about the NC-machine and NC-system means the name of the NC-machine and of the NC-system, the limits for displacement along the axes and some additional data.

The structure and the format of the block are defined by the ordered sequence of the registers and by its parameters. The identifiers and the values of the registers will be output to the NC block in the same sequence as they are placed in the list.

The mask contains the list of registers in the required order. These registers will be out to the NC-program block for the corresponding technological command.

The special problem-oriented language is used to write the programs to process the technological commands. This language allows the mathematical expressions and functions, the statements for input/output, conditional statements, cycles, jump statement, calls of

subroutines, the statements to form the NC-program blocks and the statements to work with the technological commands file.

The data definition about the NC-machine and the CNC-system, the block structure and format description, the programs design to process technological commands are performed in the postprocessors generator environment. The examination of the technological commands files and the trial generation of the NC-program are allowed in this environment too.

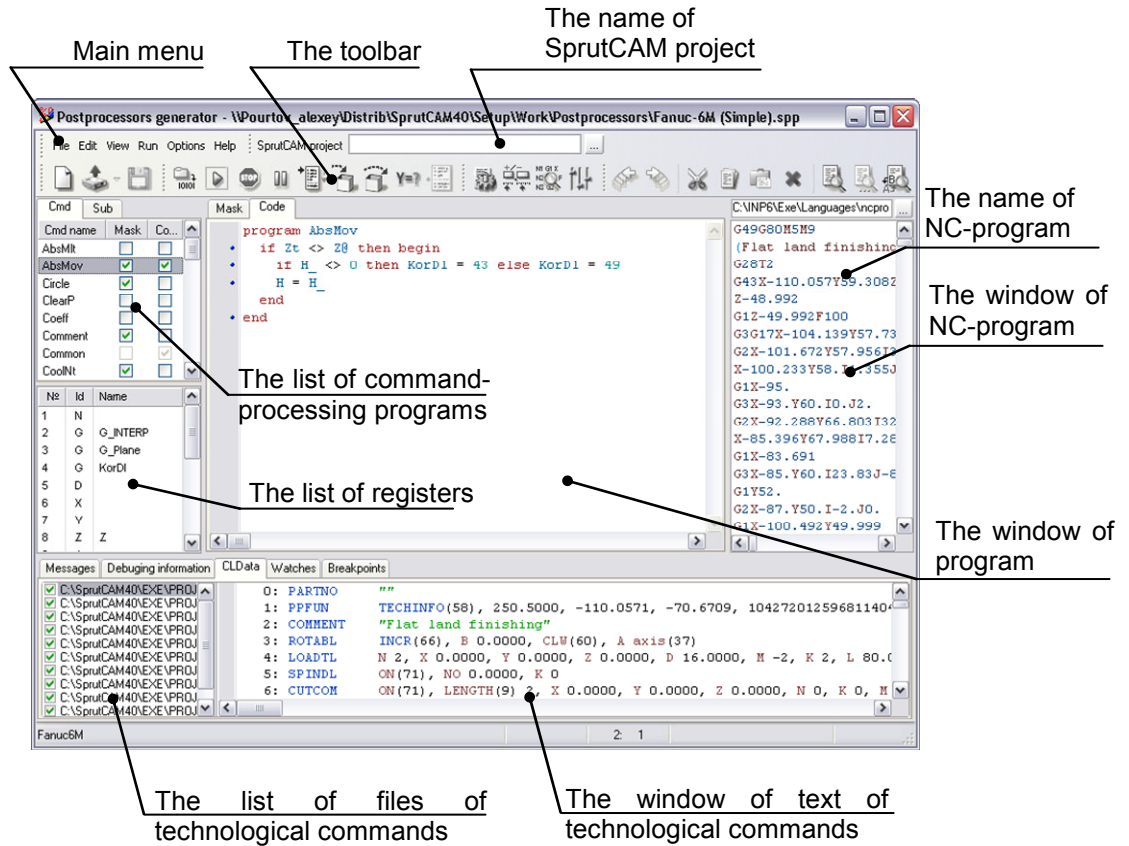
The programs sources to process technological commands, the data about the NC-machine and the CNC-system, the masks, the list and the format of registers are saved in the file with the name of concrete postprocessor and the extension ***.spp**. The data from this file is used by the executable system of postprocessor to generate the NC-program for the corresponding NC-system.

The run-time postprocessor system reads the data about the machining process from the file of technological commands, analyses the code of the command and activates the program, which process this command (the name of mask and processing program is coincident with the name of the corresponding command). Then if it was used mask, that forms line of the NC-program for corresponding of mask. The parameters of the technological command are passed via the predestined array CLD. Called program can change the registers values and internal variables values and it can form the block of the NC-program.

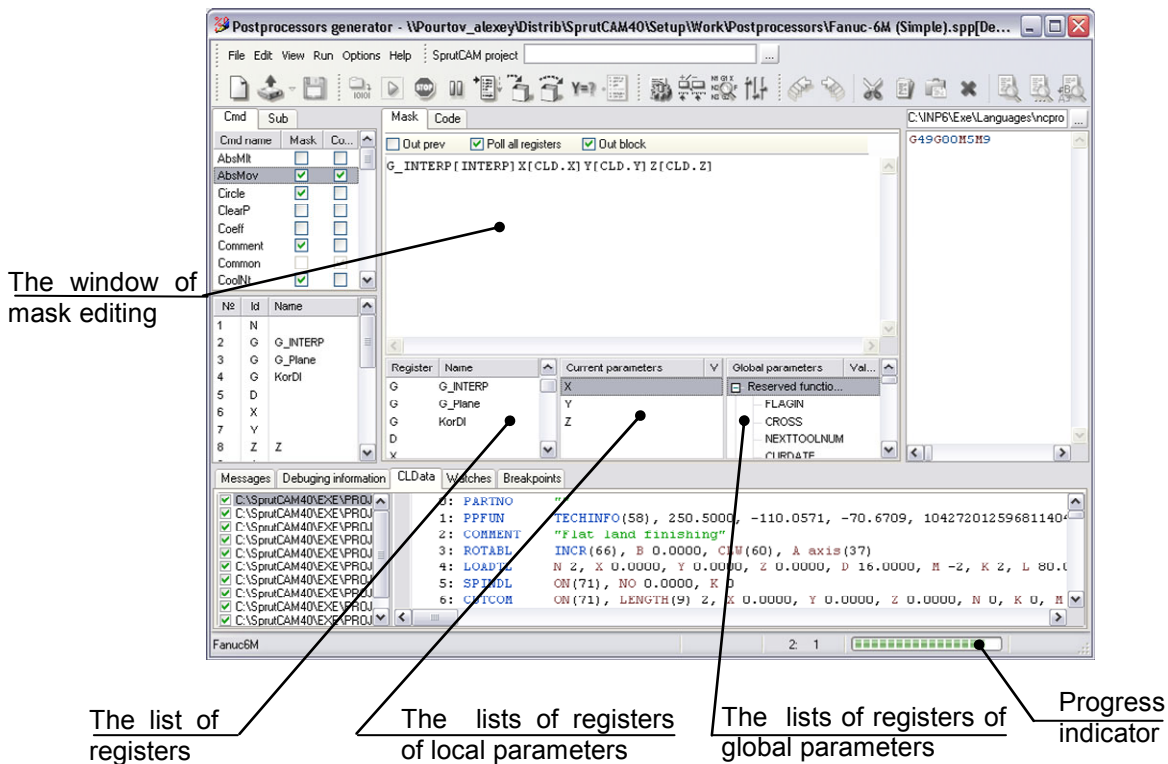
The corresponding statements in the command processing programs generate the block of the NC-program, moreover only the identifiers and values of the registers, which are changed since previous block, will be written in the current block.

1.2 THE MAIN WINDOW

The main window:



The main menu and the main toolbar are placed in the top part of the window. The list of command-processing programs and the list of registers are placed in the left part. There are the switching pages with editor of masks and editor of the command-processing program in the center. Except editor on bookmark, "**Mask**" is located lists of registers of local and global parameters. In the bottom - there are switching windows of the system messages, debug information, the list of the files, which contain the trajectory of the tool motion and the textual representation window of these files, the list of controllable variables and the list of break points.



1.2.1 The main menu

The main menu contains six general items. Some items of the main menu are duplicated in the main toolbar and in the context popup menus of the corresponding windows.

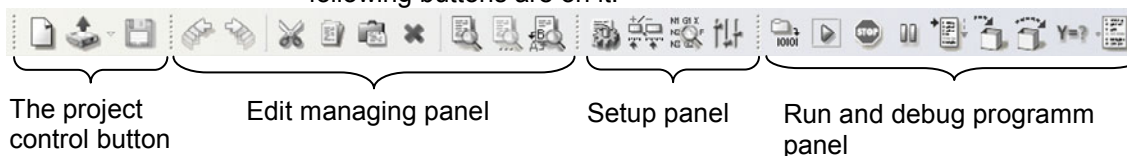
- **<File>**
 - **<New>** - the creation of a new tuning file for the postprocessor. Before the creation, the state of the system resets automatically: the open postprocessor files will be closed, the windows of the NC program and machine information will be cleared. This function can be activated from the main toolbar also.
 - **<Open>** - the opening of the earlier saved tuning file. The state of the system resets before the opening and the windows of the NC program will be cleared. This item is duplicated in the main toolbar.
 - **<Reopen>** - the opening of the earlier opened and closed files.
 - **<Save>** - saving the postprocessor adjustment file with the current name. If the file is new, the name will be asked before saving. This item is duplicated in the main toolbar.
 - **<Save As>** - saving the postprocessor is tuning file with new name.
 - **<Open CAM project>** - opening the files of technological commands from **SprutCAM** project.
 - **<Exit>** - exit from the post processors generator. If the current file isn't saved the saving will be prompted.
- **<Edit>** - All functions is duplicated in the main toolbar.

- **<Undo>** - undoes the last modification.
- **<Redo>** - returns the last modification.
- **<Cut>** - cuts the selected text or register into the clipboard.
- **<Copy>** - copies the selected text or register into the clipboard.
- **<Paste>** - inserts the text or register from the clipboard.
- **<Delete>** - deleting selected text.
- **<Find>** - searching word in the text of programs of processing of technological commands.
- **<Find next>** - searching of the following word in the text of programs of processing of technological commands.
- **<Replace>** - finding and replacing word in the text of programs of processing of technological commands.
- **<View>**
 - **<Messages>** - shows the window of the system messages.
 - **<Trace info>** - shows the window of the debug information.
 - **<CLData>** - shows the list of the technological commands and their textual representation.
 - **<Watches>** - shows the list of controllable variables with values (values of variables are displayed only if the post processors generator is in a debugging mode).
 - **<Break points>** - shows the list of break points.
 - **<Machine information>** - opens the window for input the data about NC-machine and CNC-system. This item is duplicated in the main toolbar.
 - **<Register property>** - opens the window for edit the register properties. This item is duplicated in the main toolbar.
 - **<Reinterpretation definitions>** - Opens a window of the data editing the back interpretation of NC program. This item is duplicated in the main toolbar.
- **<Run>** All functions is duplicated in the main toolbar.
 - **<Translate>** - compiles the commands-processing programs.
 - **<Run>** - runs the control program generation from the file of the tool motion trajectory. If the commands-processing programs are not compiled, then the compilation will be activated first.
 - **<Run to cursor>** - executes the program of the CLData processing to the cursor.
 - **<Step in>** - execute the next statement of the program with the entering into the subprograms.
 - **<Step out>** - execute the next statement of the program without the entering into the subprograms.
 - **<Program pause>** - stops the NC-code generation process
 - **<Program reset>** - breaks then NC-code generation process.
 - **<Evaluate expression>** - evaluates the value of the variable of the expression. It is possible in the debug mode only.
 - **<Add watches>** - Adds a new variable to the list of watches.
 - **<Add breakpoint>** - adds the breakpoint in the program of the machining commands processing or in the machining commands list.
- **<Options>** All functions is duplicated in the main toolbar.




- **<Folders>** - the path defining to the often-used folders.
- **<Editor settings>** - setting of the fonts and the syntax highlighting in the programs editor, CLData viewer, and NC-code editor.
- **<Help>**
 - **<Contents>** - shows the contents of the help system.
 - **<Sprut Technology Home page>** - loads WEB-page of JSC "Sprut-Technology". Server address: <http://www.sprut.ru>
 - **<Contact with Sprut Technology Inc.>** - prepares electronic mail for Sprut-Technology technical support department E-Mail: support@sprut.ru
 - **<About...>** - information about INP.

1.2.2 The main toolbar










The main toolbar is placed in the upper part of the main window. The following buttons are on it:







1. The project control button

	Creates the new postprocessor's tuning file. The system resets its state before creation.
	Opens the postprocessor's tuning file, which is saved earlier. The system resets its state before loading.
	Saves the postprocessor's tuning file with the current name.










2. Edit managing panel

	Cancel the last fluctuation made in the program of a handler technological command or a mask.
	Return the last fluctuation made in the program of a handler of a technological command or a mask.
	Cuts the selected text or register in the clipboard.
	Copies the selected text or register in the clipboard.
	Inserts the selected text or register from the clipboard.
	Erase the selected text.
	Searching of the gated in combination of characters in the text of the NC-program.
	Find next.
	Searching and replace of the gated in combination of characters in the text of the NC-program.

3. Setup panel

	Opens the window for editing the data about the NC-machine and the CNC-system.
	Register parameters window.
	Reinterpretations definitions window.
	System setup window.

4. Run and debug program panel

	Runs the compilation of the command processing programs.
	Runs the generation of the NC-program from the files of the tool motion trajectory. If the command processing programs aren't compiled, the compilation will be activated first.
	Break NC-program generation.
	Pause NC-program generation.
	Generate NC-program up to current position in a CLData.
	Generate NC-program with stopping in subprograms.
	Generate NC-program with without stopping in subprograms.
	Compute variable or expressions. The evaluation is possible only in a debug mode.
	Add breakpoint in a handler of technological commands or CLData files

5.



Opens the technological commands file, generated by SprutCAM.

1.2.3 The process indicator

The process indicator activates automatically when the system performs the operation, which takes a long time. The source programs, the control programs generation and the technological commands conversion in to the textual representation are such operations. These operations can be cancelled by pressing the left mouse button on the process indicator.

1.2.4 System settings

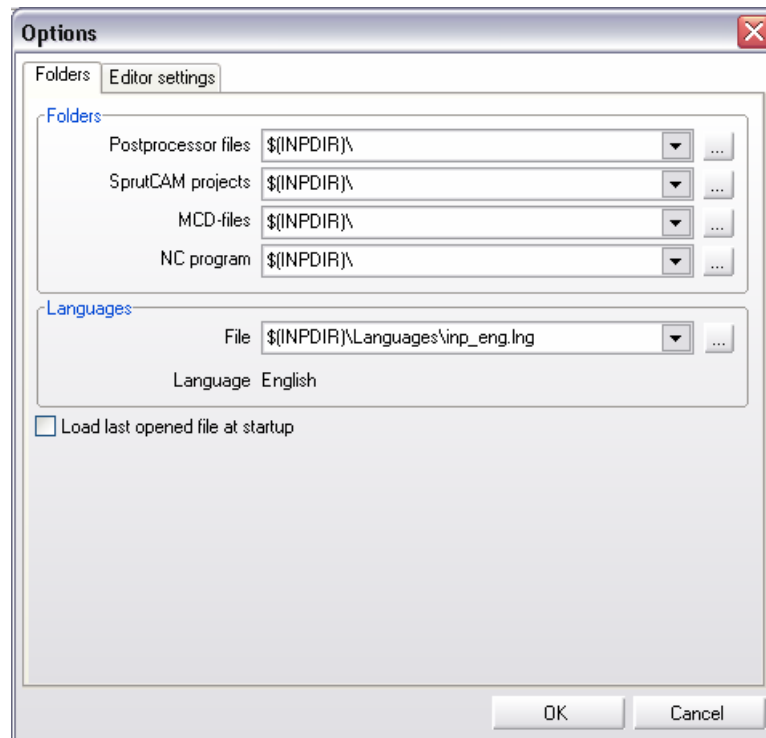
The window of system settings can be activated by pressing the button on the main toolbar or by choosing the items **<Options>**

<**Folders**> in the main menu. The default paths for the system files and languages are defined here.

The postprocessor adjustment files for various NC-system (*.inp and *.ppp - files) are loaded from the <**Postprocessor files**> directory. New postprocessor tuning files will be saved in this directory.


The **SprutCAM** projects are loaded as default from the <**SprutCAM project**> directory. The technological commands files (*.mcd), which are linked with opened project will be loaded from the corresponding paths, described in the project file.

The separate technological commands files (*.mcd) are loaded, as default, from the <**MCD-files**> directory.



Generated NC-programs are saved, as default, in the <**NC-programs**> directory.

To change program language it is necessary use <**Languages**> panel.

These paths may be edited manually or using dialogs, which can be activated by pressing  button.

In a system, there is a preconceived variable, which one may be used for definition of the conforming folders:

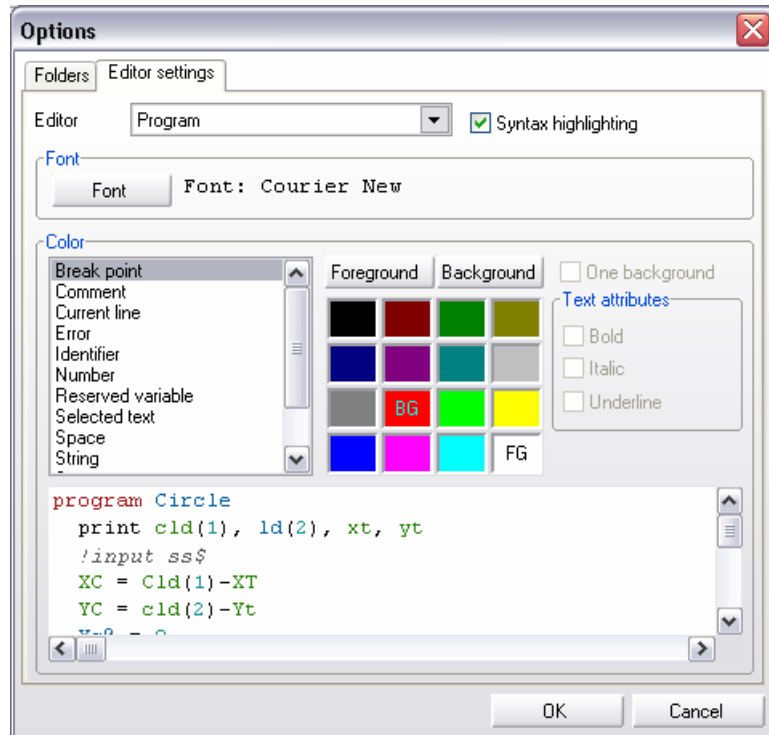
\$ (INPDIR) - the folder, from which one was, triggered the generator of postprocessors.

At determining actual names of folders during operation the indicated variable will be substituted by the conforming full path now of activation of a system or now of the last editing the system settings.

For load last open *.SPP the file at activation of the generator of postprocessors there is a check box <**Load last opened file at startup**>.

1.2.5 Editor settings

Programs of technological commands consist of identifiers, the numbers, identifying reserved variables and functions, commenting etc. For each of these element blocks it is possible to adjust the color and type style, a background color.



An editor settings window is called by selection of points **<Options>** **<Editor Settings>** in a main menu. In a window, the font and illumination of syntax for the editor of programs of technological commands, windows of a text mapping of CLData files and windows of map of a NC-program adjusted.

For definition of customizations in a falling out list **Editor**, it is necessary to select one of editors: **Program**, **CLData** or the **NC program**. For the selected editor on the panel **Font** sets a name and a font size.

On the panel, **Color** for each element block it is possible to assign color of the text, a background color of the text, type style.

For fast discoloration of a background of all element blocks, there is a check box **One background**. At selecting check box, the background color of all groups is substituted on a background color of a flowing member.

The **<OK>** button closes the window and saves all modifications. The **<Cancel>** button closes the window and discards all modifications.

1.2.6 Defining the data about the NC-machine and CNC-system

The editing window of the data about the NC-machine and the CNC-system can be opened by pressing the button on the main toolbar or by choosing the items **<View>** -> **<Machine information>** in the main menu.

The **<OK>** button closes the window and saves all modifications. The **<Cancel>** button closes the window and discards all modifications.

Machine information

Basic info

NC system name: Fanuc6M

Machine name:

Author: HAL

Comments: Simple

NC file extension: F6M Date: 28.07.2004

Machine and NC system information

Arc center coordinates: Incremental

Circle division: Dont break

Support helical moves: Supported

Maximal radius: 360

☐ Linear movement only

Comments in upper case: No

Spaces between commands: No

NC block numbers:

Symbol: N

Start value: 1

Step value: 0

Frequency: 1

Maximal transition along the axes:

Maximal X coordinate: 0

Maximal Y coordinate: 0

Maximal Z coordinate: 0

OK Cancel Apply

Basic info

The name of NC-machine, the name of CNC-system and the extension of NC-program files are displayed in the upper part of the window. The name of NC-machine and the name of CNC-system are informative parameters only. All NC-programs, which will be generated using edited tuning-file, will be saved with the specified extension.

Machine and NC system information

In a field **Arc center coordinates**, the mode of the definition of circle center is determined. If the relative mode of the definition of center of a circle in the variables **XC**, **YC**, **ZC** are set concerning for current selected point. If the absolute mode of the definition of center are set in absolute values is selected.

In a field **Circle division** is offered to select a mode of introducing of arcs in a NC program (a quarter, half or a three sixty). The postprocessor during operation arrests intersection arcs of quadrants and, if necessary, dissects arcs into halves or quarters.

In a field **Support helical moves** are set with the information on support by a CNC system of helical moves. If helical moves are not supported, the postprocessor automatically approximates helical arcs on cuts and shapes commands of linear movements.

In a field **Maximal radius** is entered a value of maximum radius of the arc bolstered by a CNC system, at overflow which one the postprocessor substitutes arcs linear migrations. The value **0** in it a field disconnects monitoring, and the system will not approximate an arc at any value of radius.

For an event when the CNC system at all does not support arc interpolation and helical moves, check box **Linear movements only** is stipulated. In this case, all arcs without dependence from fields **Support helical moves** and **Maximal radius** are dissected into linear movements.

The capability stipulated to shape **Comments in upper case** and to place **Spaces between commands** of NC program.

For forming a block number there is a group of fields **NC block numbers**. The group consists of fields **Symbol**, **Start value**, **Step value**, **Frequency**. In a field **Symbol** is entered the identifier of the register or a symbol injected in block before a value of a block number. Block is numbered since **Start value**. After each output of block to a block number **Step value** is added. Block number outputs with the given frequency. If **frequency** is, peer 1 the block number outputs in each command of forming of block if 2 - through one command, 3 - through two commands etc. If frequency is peer to null the block number not output.

Maximal transitions along the axes allow inspecting a correctness of a NC program. At overflow of as much as possible admissible transition on one of coordinates, the conforming warning will be show.

1.2.7 Postprocessor parameters inquiry while the first using

Sometimes a dealer does not know some specific information about the user machine. It is possible to make the data inquiry. In this case the user will be asked about the NC-system information and inputted data will be saved in the postprocessor file.

It is necessary to edit the postprocessor file by simple text editor (wordpad.exe for example) to organize the data inquiry. The section **[Common definitions]** is shown below.

[Common definitions]

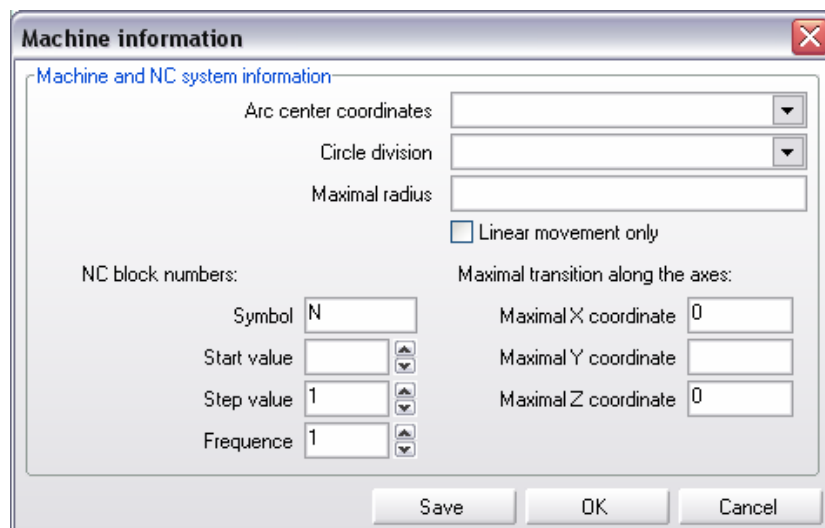
```
Spaces N          ! Y / N
CtrInc Y          ! Y / N
ArcDivision N     ! N / 90 / 180
Helical Y        ! Y / N
MaxRad 1000      !>= 0
LinearMovement N ! Y / N
UpperCaseComment N ! Y / N
Sequence N 1 1 1 ! Symbol Frequence StepValue StartValue
Xmax 0           !>= 0
Ymax 0           !>= 0
Zmax 0           !>= 0
```

If some parameter is changed to the “?” then it will be asked from the user before the Postprocessor run until the user save the postprocessor. For example if the section has the text below:

[Common definitions]

```
Spaces N          ! Y / N
CtrInc ?          ! Y / N
ArcDivision ?     ! N / 90 / 180
Helical Y        ! Y / N
MaxRad ?         !>= 0
LinearMovement ? ! Y / N
UpperCaseComment N ! Y / N
Sequence N 1 1 ? ! Symbol Frequence StepValue StartValue
Xmax 0           !>= 0
Ymax ?          !>= 0
Zmax 0           !>= 0
```

Then the next dialog will appear before the NC program creation.



The dialog box is titled "Machine information" and contains a tab labeled "Machine and NC system information". It includes the following fields and controls:

- Arc center coordinates: A text input field with a dropdown arrow.
- Circle division: A text input field with a dropdown arrow.
- Maximal radius: A text input field.
- ☐ Linear movement only: A checkbox.
- NC block numbers:
 - Symbol: A text input field containing "N".
 - Start value: A text input field with up/down arrows.
 - Step value: A text input field containing "1" with up/down arrows.
 - Frequency: A text input field containing "1" with up/down arrows.
- Maximal transition along the axes:
 - Maximal X coordinate: A text input field containing "0".
 - Maximal Y coordinate: A text input field.
 - Maximal Z coordinate: A text input field containing "0".

At the bottom are three buttons: "Save", "OK", and "Cancel".

If click “**save**” then the data will be saved in the postprocessor file and the dialog will not appear in the next time. If click “**OK**” then the NC program will be generated with the inputted data and the dialog, will appear in the next time again. If click cancel then the program will not be generated

Additional information input

It is possible to ask the user about the values of the global variables in the first time post running. To make the inquiry it is necessary to edit the **[Initial questions block]** section of the postprocessor file by the text editor. The section format is shown below:

```
[Initial questions block]
Variable_name      Variable_value Variable_description_or_query
  Value_1          Value_description_1
  Value_2          Value_description_2
  ...
  Value_N          Value_description_N

Variable_name      Variable_value Variable_description_or_query

Variable_name      Variable_value Variable_description_or_query
  Value_1
[Initial questions block end]
```

Where:

Variable_name – the name of variable that is available in the postprocessor. The variable can be one of three types: **String**, **Real**, **Integer**;

Variable_value – the start value that will be assigned to variable. If white “?” instead value then this variable will appear in the start input dialog;

Variable_description_or_query –This message appears in the start input dialog near the input field. The message must be in double quotes;

Value_N – the value that has the same type like the global variable;

Value_description_N – the value description. The value must be in double quotes.

Every variable in the **[Initial questions block]** adds the corresponding field to the start input dialog.

The first variable adds the field with the drop down list. The **Variable_description_or_query** will be written forepart. The drop down list will have the next items: **Value_description_1**,

Value_description_2,... Value_description_N. If some item is selected then the corresponding value will be assigned to the variable.

The second variable adds the simple field to data input. The **Variable_description_or_query** will be written forepart. The data field will be empty.

The third variable adds the simple field to data input. The **Variable_description_or_query** will be written forepart. The Value_1 will be inside the input field.

The values of the string type variables must be in double quotes.

For the [Initial questions block] section that is shown below:

```
[Initial questions block]
Intrp ?      "Select the circlecenter mode"
      1      "IJ"
      2      "R"

Init$ ?      " Input the word for the machine Initializing"

L      ?      "Input the distance"
      1000

P$      ?      "Select the parameter"
      "A"      "first parameter"
      "B"      "second parameter"
[Initial questions block end]
```

The start input dialog will be the next:

After the save button is pressed the section will be the next:

```
[Initial questions block]
Intrp 1      "Select the circlecenter mode"
      1      "IJ"
      2      "R"

Init$ "G9G1G20G40G90F4000Z0M5"      " Input the word for the
machine Initializing"

L      995      "Input the distance"
      1000

P$      "B"      "Select the parameter"
      "A"      "first parameter"
      "B"      "second parameter"
[Initial questions block end]
```

After that the start input dialog will not appear and the variables **Intrp**, **Init\$**, **L** and **P\$** will be assigned to **1**, **"G9G1G20G40G90F4000Z0M5"**, **995** and **"B"** correspondingly.

1.2.8 The block structure and format definition (Register list forming)

To define the structure and the format of a block, it's necessary to form the list of the registers and to define their parameters.

Block of NC program consist from words. Each word contains **address** and **value**. **Address** – letter (sometimes several letters), **value** – is number present in definite format.

The determined **Register** is connected to each address in the postprocessor.

The concept of the **Register** of the postprocessor integrates following properties:

- The identifier of the register (the address in a NC program);
- The format of an output of a value of this address in block;
- Current value conforming to the address;
- The previous value conforming to the address;

A name of the register (a variable of the postprocessor, the bound with the address, through which one is manufactured access both to flowing and to the previous value of the register).

The NC-program blocks are formed automatically when the **OUTBLOCK** and **FORMBLOCK** statements are performed. The NC-program block is formed by the system according the following algorithm. The system examines the registers sequentially; if the current register value differs from its previous value, then the register will be written in the block and its current value will be assigned to its previous value, else the register will not be written in the block.

When the system writes the register in the NC-program block, it writes the register identifier first, and then it writes the register value, multiplied by the register's scale. The value will be written in the block according the format, described for this register (the length and the precision of the register value, the presence of decimal point, of sign and of leading and insignificant zeroes).

Thereby, to define the structure and the format of the block, it is necessary to fill the registers list and define the registers properties. The registers must be placed in the list in this order, in which they and their values must appear in the blocks of NC-program. This rule is correct for the processing programs. If the block is formed with, the mask using then the sequence corresponds to the mask.

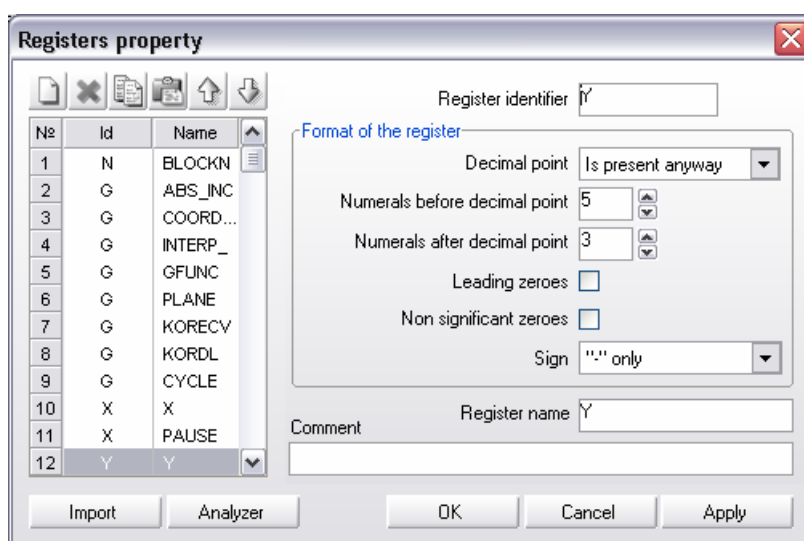
Note: *Different registers must not have same names, but they may have the same identifiers (the symbols, which will be written in the block before the value of register). This allows creating the separate register for each group of the functions with some type. For example, we create the register with the name "ABSOLUTE" and with the **G** identifier for the preparative function of the switching between the absolute/incremental coordinate systems. Suppose also, we create the register with the name **INTERP** and the same identifier **G** for preparative function of positioning, linear interpolation, circular interpolation with direction. This allows us: At the first to write both these commands in the same block of NC-program, if it is necessary, (for example, **N190 G91 G1 X50 Y30**). At three second to trace the current transitions mode (positioning, linear or circular interpolation with direction), to determine the current coordinate system (absolute or incremental) by examining the current values of registers **ABSOLUTE** and **INTERP**.*

The list of registers is shown in the left part of main window.

Nº	Id	Name
1	N	BLOCKN
2	G	ABS_INC
3	G	COORDSYS
4	G	INTERP_
5	G	GFUNC
6	G	PLANE
7	G	KORECV
8	G	KORDL
9	G	CYCLE
10	X	X
11	X	PAUSE
12	Y	Y
13	Z	Z
14	I	XC_
15	J	YC_
16	K	ZC_
17	A	ANGLE







The register properties description

The register properties edit window is opened when the register is added, when the item **<Properties>** in the context popup menu is selected or by double-clicking by left mouse button on the register in the list.



The 'Registers property' dialog box contains a list of registers on the left and configuration options on the right. The list includes columns for 'Nº', 'Id', and 'Name'. The right side has fields for 'Register identifier' (set to 'Y'), 'Format of the register' (with options for decimal point, numerals before/after decimal point, leading/non-significant zeroes, and sign), 'Register name' (set to 'Y'), and a 'Comment' field. At the bottom are buttons for 'Import', 'Analyzer', 'OK', 'Cancel', and 'Apply'.

Register list place in the left part of window. Edit list buttons place higher:

	Inserts the new register after selected one. The window to edit the properties of a new register will be opened automatically.
	Deletes selected register.
	Copies the selected register into the clipboard. This function is duplicated in the main toolbar.
	inserts the register from the clipboard after the selected one. The window to edit the properties of inserted register will be opened automatically. This function is duplicated in the main toolbar.
	Move the current register up on a list.
	Move the current register down on a list

To change the position of the register in the list, it's necessary to press the left mouse button on the register and drag it in the desired position, holding the left mouse button down.

Following fields can be defined in this window:

Register identifier – the symbols, which will be written in the NC-program block before the register value;

Decimal point – this field can have following values:

- Is absent,
- Is present,
- Is present anyway;

If the item **<Is present>** is chosen, then the decimal point will be present, if the register value has fractional part;

Numerals before decimal point – Maximum quantity of signs in the whole part of number;

Numerals after decimal point – Maximum quantity of signs in a fractional part of number;

Leading zeroes and Non-significant zeroes – defines the zeroes output mode before and after the register value;

Sign – defines the output mode for the sign of the register value. Following options are available:

- No,
- “-” only,
- “+” and “-” always;

Register name – the name of the register, which is used by the command processing programs;

Comment – comments to register.

The **<Import>** button is intended for import of a list of registers from postprocessors of **SPP** formats, **PPP** (the format of the aged version), and as from SurfCam postprocessors.

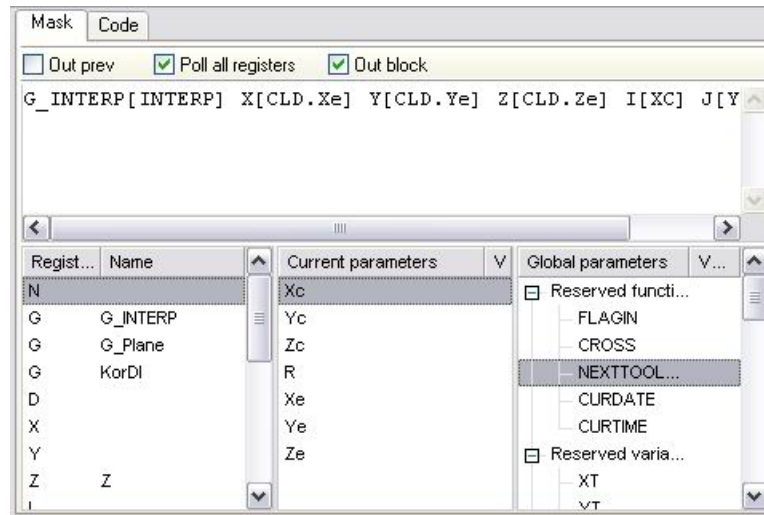
For forming a list of registers based on a NC program, the **<Analyzers>** button is intended. Thus the text of the indicated NC program and all retrieved addresses is analyzed are added in a list of registers.

The **<OK>** button closes the window and saves all modifications. The **<Cancel>** button closes the window and discards all modifications. The **<Apply>** button save all changes, window of registers is not closed.

1.2.9 The masks for the machining commands translation

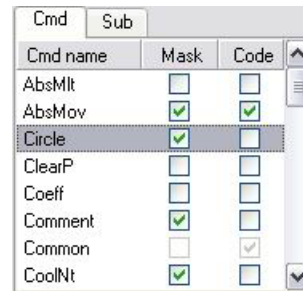
The usage of the masks quickly and simply allows defining the transformation method from the CLData machining commands to the NC code line. Any CLData command has the corresponding mask, in which the words and the values for the words are defined.

The mask editor is located on the **Masks** page.



For the simplifying the process of the mask definition there are the registers list, current command parameters list and the global variables list in the bottom part of window. The double click on any element of these lists inserts the element to the mask text. Under the mask, text the mask switches is located.

The mask activating is performed by the tick setting on the CLData commands list.

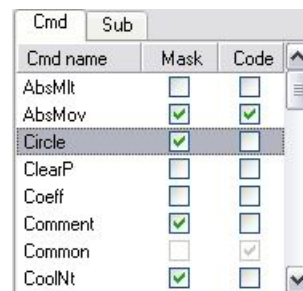


The masks definition rules is described in details in the chapter 2.

1.2.10 The programs for the CLData commands processing

Like the masks the programs is meant for the defining the transformation process from the CLData command to the NC code line. The programs can expand the mask or can be used instead the masks. Both the mask and the programs have merits and demerits. The program is the flexible and powerful tool for the realization of the very complicated transformation. However, the learning of this tool requires the programming experience.

Any CLData command has the corresponding processing program.



These programs are designed using the special problem-oriented language and can contain the mathematical expressions and the functions, statements for input/output, conditional statements, cycles, jump statement, subroutines calls, the statements to form the NC-program blocks and the statements to work with the technological commands file. This language is described in detail in the chapter 3.

Each program begins from the header, which consists of the **<PROGRAM>** keyword and the program name, and terminates by the **<END>** keyword. The program name is coincident with the command name, processed by it. When the program is called, the parameters are passed to it via the predefined CLD array

The program activating is performed by the tick setting on the CLData commands list in the code column. If the program is not active then it will not be translated and executed even if the text exists. The program **COMMON** is exclusion. It is translated and executed always and at the first. This program is intended to define the global variables i.e. variables that are accessible from any subprogram.

The **COMMON** program is executed always. Inactivated program is not translated and is not executed even if the program has the code.

The empty program is generated by double click in the list of technological commands.

When the right mouse button is pressed in the programs list, the context popup window will appear. This popup contains following items:

- **<Insert>** - adds a new technological command processing program. The window to input the program name will be opened.
- **<Delete>** - deletes selected program;
- **<Rename>** - renames current program. The window to input the program name will be opened.

The editor of the selected program is placed in the center of main window. When other program is selected in the list, its text will be displayed immediately in the edit window

1.2.11 Subprograms

Subprograms can be written in the same language like the processing programs. Write the **"CALL"** operator to call the sub program

Every subprogram is started from the header that is contains the key word **"SUB"** and the subprogram name. The key word **"SUBEND"** must be written in the end. The Subprogram name cannot coincide with the existed subprograms.

1.2.12 The command processing programs compilation

The compilation can be running by pressing the button on the main toolbar or by choosing the **<Run>** -> **<Compile>** items or by pressing the **<Ctrl-F9>** combination.

Note: An activated program is compiled only.

The system messages about the compilation process are outputted in the **messages** page in the bottom part of the main window.



If there are errors during the compilation, then the left mouse button double-clicking on the error message performs the corresponding program source loading and the cursor will be set in the incorrect string.

1.2.13 The work with the files of technological commands

The files of technological commands can be loaded from the **SprutCAM** project (*.stc) or from the separate files (*.mcd). The pressing the button or the choosing the corresponding item in the main or context menu loads the file

When the files of the technological commands are loaded from the **SprutCAM** project, the project name, which contains these files, will be displayed

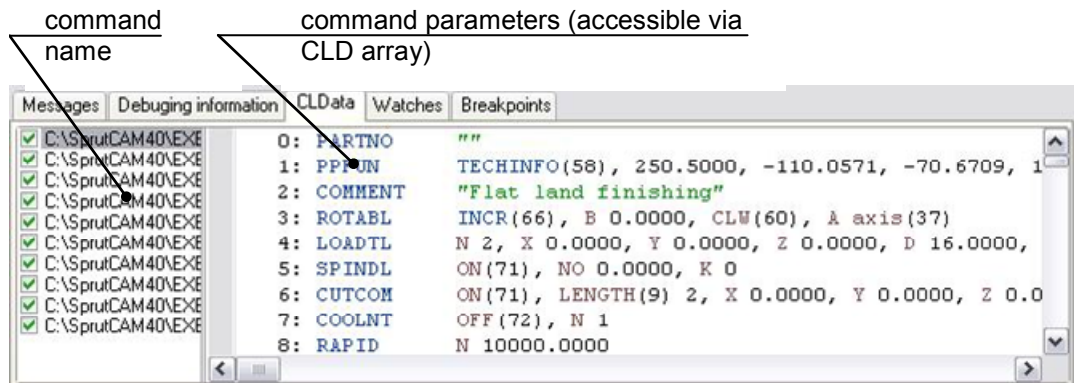
In the bottom part of main window on the **CLData** page the list of machining commands files is located. Any line of the list corresponds to a **SprutCAM** operation. Only ticked operations will form the NC code.

The operations list can be edited by popup menu. Click right mouse button in the list field to open popup menu. Menu contains the next items:

- **<Insert>** - adds the file of the technological commands;
- **<Delete>** - deletes selected file;
- **<Delete all>** - deletes all files.

Selected technological file is loaded in the textual representation window in the terms of the common unique intermediate language **CLDATA**.

Any technological command in the **CLData** text is the line contained the name and parameters. The parameters are accessible via the CLD array in the programs and via the current command parameters list in the masks.



When the left mouse button is double-clicked on the command name, the mask corresponding to the command will be loaded to the masks editor and the program corresponding to the command will be loaded to the programs editor.

1.2.14 The test NC-code generation

The trial NC-program generation can be performed by pressing the button on the main toolbar, by choosing the **<Run> -> <Run>** items of the main menu or by pressing the **<F9>** key. If the technological commands processing programs are not compiled at this moment, the compilation will be activated automatically. If there are errors during the compilation, the trial running will not be performed.

The loaded files of technological commands are the input data for the NC-program generation. The NC-program is formed using the technological files, which are ticked in the list.

The algorithm of generation is described below:

- 1.The system reads the next command form the file of the technological commands.
- 2.The system analyses the command code and according to the command parameters define the predefined CLD array and predefined variables.
- 3.If the corresponding program for the command is activated then the program is executed.
- 4.If the corresponding mask for the command is activated then the NC code line is generated by mask.

Note : *So for any command both program and mask can be activated In this case the program will be executed first and mask later. If both mask and program are not activated then the command will be ignored.*

- 5.If in result of the previous actions the NC-code line is generated then the subprogram **Filter** is executed. Usually this subprogram is used for the replacement of some elements in the NC block before the final output to the NC program.

The corresponding operators form the NC block from the mask or from the processing program. If the processing program is used then only the ids and values the same registers are output that was changed after the previous block output.

The NC-program will be saved in the file with the name, defined in the field of the NC-program name. This field is placed in the upper right corner of main window If this field is empty then the NC program is output to the window only and is not saved to the file.

The debug information, which is formed in the processing programs, is written in the debug window.

If there are errors during the generation process, they will be written in the system messages window.

After the faultless NC code, generation there is the link between the NC code and the CLData commands. By double click on the command name in the window of CLData commands the corresponding line in the NC code will be selected. Conversely, by double click on the code line in the NC program window the corresponding CLData command will be selected.

1.2.15 Programs debugging

The destination of the built-in debugger

Integrated debugger allows controlling the program execution. The program can be executed step by step with the checking the program code and the execution result. While debugging it is possible to enter into the subprograms or execute the subprogram in one-step, to

control the variable values and to view the debug messages. The debugger cardinally reduces the postprocessors designing time and lightens the bugs search and elimination.

Debugger functions

- Program run **F9**
- Run program to cursor **F4**
- Step over (execute the current statement without the entering into the subprogram) **F8**
- Breakpoint setting/resetting **Ctrl + F8**
- Step into (execute the current statement with the entering into the subprogram) **F7**
- Break the debugging **Ctrl + F2**
- Add a new variable to the watch list **Ctrl + F7**
- Evaluate expression **Ctrl + F4**
- Before the program is started the all modified subprograms is compiled. If there is no errors found while compilation then the program is started.
- To start the program execution it is possible by any of commands: **<F4>**, **<F7>**, **<F8>** or **<F9>**.
- **<F4>** runs the program to the statement on the cursor position. If current string has not the statement then the program is executed to the end. The string where the execution was break is the debugger cursor that is selected by color.
- If debugging is started then **<F7>** and **<F8>** executes the statement of a program where the debugger cursor is located else the debugger cursor is set on the first statement of the first program. Command **<F7>** as distinct from **<F8>** allows entering into the subprogram if the current statement is a **CALL**.
- **<Ctrl + F8>** allows set/reset a breakpoint. Then the program execution arrives at the breakpoint then the execution will be paused with the debugger environment activation.
- In any point of debugging process, it is possible to break the execution by **<Ctrl + F2>** or to continue the execution without debugging by **<F9>**.
- The changing of the lines number in the debugged program brings the incorrect indication of the current debugger cursor.
- **<Ctrl + F7>** allows adding the variable or the expression to the watches list. If **<Ctrl + F7>** is pressed in the editing mode then variable name is taken from the current position of cursor. In the debugging mode then the mouse pointer is located under the variable then the variable value is highlighted in the hint.

1.2.16 Reinterpretations programs definition

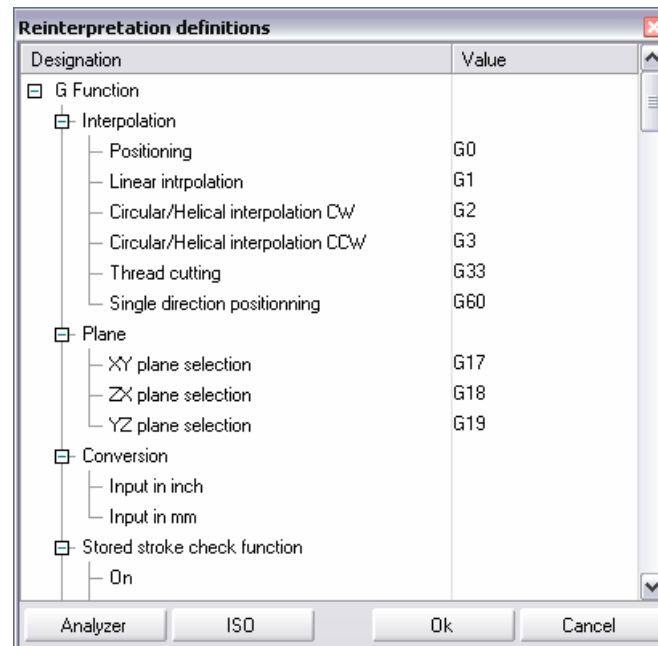
It is not necessary define reinterpretation parameters for postprocessor work.

Reinterpretation parameters it is necessary define on the machining simulation from ready NC program.

This function is executed with program **NCTuner**.

Reinterpretation definitions window is called from the <View> <Reinterpretation definitions>. As this window can be called from a toolbar the press Reinterpretation definition button.

The list of assignments is down on groups and shown by the tree. Fill up of the greater number of assignments reconciles in more precise simulation process.



<Analyzer> automatically shapes assignments on the filled masks of technological commands.

There is a capability automatically to fill in a list of assignments pursuant to standard ISO (for example, a CNC system Fanuc 6M). It is made with the <ISO> button.

The <OK> button closes the window and saves all modifications. The <Cancel> button closes the window and discards all modifications.

2 MASKS

2.1 BASIC DEFINITIONS

Masks describe the forming rules for the NC-program block that is corresponded with any CLData command.

Any mask is linked with the CLData command with the same name. The mask is the text that approximately looks like the block (line) of the NC-program.

2.2 MASK STRUCTURE

Mask can contain some lines. In general, the mask line looks like:

element1 element2 ... elementN

2.2.1 Mask element

Any mask element is out to the NC-program line in the form as it looks in the mask.

Sample:

Mask: "G0 rapid movement"

NC code: "G0 rapid movement"

If the mask element is in a brackets "[" and "]" then it is replaced by value or variable that located in the brackets.

Sample:

Mask: "[XT] [200]"

NC code: "100.12456 200"

In the sample, the value of the XT variable is equal to 100.12456

In the brackets, it is possible to use:

1. All variables of the COMMON subprogram;
2. predefined (reserved) variables: XT, YT, ZT, XC, YC, ZC, INTERP, TOOLRAD, CLDATA\$, ARCPLANE, XP, YP, ZP, FEED, TLCOMP, TRCOMP, FROMX, FROMY, FROMZ, CURCODE, NCNAME\$, NCPATH\$, BLOCKSTEP;
3. predefined (reserved) functions: FLAGIN, CROSS, NEXTTOOLNUM, CURDATE, CURTIME (see section "Predefined variables" for details),
4. all parameters of the current technological command passed by the CLD array.
5. The numbers has the different representation in the NC-program.

The number out method is defined by the next parameters:

1. **<Decimal point>** – this field can have following values: <Is absent>, <Is present if the number has the fractional part>, <Is present anyway>,
2. **<Integer width>** - the maximal digits number to represent the integer part of the number;
3. **<fractional width>** - the digits number to represent the fractional part of the number;

4. **<Leading zeroes>** and **<Non-significant zeroes>** – defines the zeroes output mode before and after the number;
5. **<Sign>** – defines the output mode for the sign of the number. Following options are available: **<No>**, **<"-" only>**, **<"+" only>**, **<"+" and "-" always>**;

Then the value is out to the NC-program block the **default number format** is used:

1. **<Leading zeroes>** and **<Non-significant zeroes>** - Is absent;
2. **<Decimal point>** Is present if the number has the fractional part;
3. **<Sign>** is present if the number is negative;
4. **<Integer width>** and **<fractional width>** allows to out number without rounding
5. Element identifier is output as text.

2.2.2 Registers in the masks

Registers is used to define the format of the number output according to the requirements of the NC systems.

The register in the postprocessor unites the next properties:

- Identifier of the Register;
- Current value of the register;
- Previous value;
- Value output format;
- The name of the register;

(See section "The block structure and format definition" for details).

If write the register name or register identifier before the open square bracket then the number output format of this register will be used for the value output.

Sample:

```
Mask: "G_INTERP[1] X[XT] Y[YT] Z[ZT] F[200]"
NC code: "G1 X100.100 Y-245.100 Z-010.560 F200"
```

While the mask analysis it is presumed that the word before the open square bracket defines the register. Executable system looks for the register by name, if the register is not found then the system looks by identifier. The registers are looked in order how it has defined in the registers list. So if the list has some registers with the same identifier name then the first will be used.

If register is not found then value is output by default number format and the word that is in mask before the open square bracket is output before the number.

Sample:

```
Mask: "G_INTERP[1] X[XT] YYY[YT] Z[ZT] F[200]"
NC code: "G1 X100.100 YYY-245.100034 Z-010.560 F200"
```

In the sample the word "YYY" is absent in the registers list.

If the register is found then the current register value is assigned to the old register value and the new value is assigned to the current register value. After that, the executable system compares the old and the current values and if its are not equal then the register identifier and value is output to the NC-program line accordingly to the defined number format.

Sample:

Lets the current register F value is equal to 200.

Mask: "G_INTERP[1] X[XT] Y[YT] Z[ZT] F[200]"

NC code: "G1 X100.100 Y-245.100 Z-010.560"

The old and new value of the register is equal, so the register F value is not output to the NC code.

2.2.3 Modifiers

Some times, it is required to output the new register value independently if it is changed or not, and conversely to change the register value and do not output it to the current line. The modifiers **On** and **Off** is intend to do it.

The modifier is written in the square brackets after the value and comma.

Modifier On. If modifier **On** is written then the register value is output to the NC code in any case.

Sample:

Lets the current register G_PLANE value is equal to 17.

Mask: "G_PLANE[17, On] X[XT] Y[YT]"

NC code: "G17 X100.123 Y200.456"

It is possible to out current value of register without the new value assign. To do it it is necessary to write **On** without value.

Sample:

Mask: "G_PLANE[On] X[CLD.X] Y[CLD.Y]"

NC code: "G17 X100.123 Y200.456"

Modifier Off.

If modifier **Off** is written then the new value is assigned to the register but the register is not output to the NC code.

Sample:

Mask: "G49G80M5M9 G_PLANE[17,Off]"

NC code: "G49G80M5M9"

It is possible to use modifier **Off** without value. It is needed to exclude the out of the current register value to the NC code without the assigning of a new value.

2.2.4 Expressions

It is possible to write the expressions to the masks. The expression is the mathematical formula. The result of expression is a number.

Syntactically, the math expression is the combination of the numbers, numerical variables and numerical functions, separated by the signs of the math operations and the parentheses. The simplest examples of math expressions are the number and the numerical variable

Following math operations are allowed in the language:

- + the addition
- - the subtraction
- * the multiplication
- / the division

- ^ the involution

It is necessary to remember, that two operational sign can't follow one-after-another.

Following math functions are allowed:

- SIN(x) -sinus of angle x (in degrees);
- COS(x) - cosine of angle x (in degrees);
- TAN(x) - tangent of angle x (in degrees);
- ATN(x) - arctangent of angle x (in degrees);
- ASIN(x) -arcsine of angle x (in degrees);
- ACOS(x) - arccosine of angle x (in degrees);
- SQR(x) - square root of x;
- ABS(x) - absolute value of x;
- SGN(x) - sign for x;
- ROUND(x, y) – rounding of x to y decimals after point;
- LOG(x) – decimal logarithm of x;
- LN(x) – natural logarithm of x;

Predefined variables and functions, all parameters of current CLDATA command transmitted via the CLD array are available in the expression.

Sample:

Mask: "X[2+5/2+2*(sin(45))] Y[CLD.Y*2] F[FEED]"

NC code: "X005914 Y001234 F200"

In this sample the expression of the first element: 2+5/2+2*(sin(45)) is equal to 5.91421356237697. When out to the NC code, the X register was found in the registers list and the number output format was taken from the register. The expression of the second element has the parameter transmitted via the CLD array - CLD.Y (equal 0.617) multiplied on 2. The third element uses the predefined variable FEED. This variable contains the current feedrate value.

2.2.5 Nested mask

Sometimes it is required to out the register value if the value of other register is changed. Then the **nested mask** is used.

Sample:

Mask: "Z[CLD.Z, G[43]]"

NC code: "G43 Z0.123"

The value of the register **Z** is changed and value 43 is assigned to the register **G**.

It is possible to use the modifiers, expressions, and other nested masks in the nested masks. The registers of the nested mask will be included in to the NC-program line in the order according to the registers list.

Sample:

Mask: "X[CLD.X] Y[CLD.Y] Z[CLD.Z, G[43]]"

NC code: "G43 X6.141 Y-4.234 Z0.123"

The changed register **G** is located in the start of line according to the registers list.

2.2.6 The separators of the mask elements

In general, the mask line consists of some elements. It is possible to insert the spacebars between the elements. The spacebars has not influence on the NC-program line.

To control the spacebars between the words in the line it is possible to use the parameter “**Spaces between commands**”. The parameter is available on the **main menu->view->machine information**. If the parameter is checked then the spaces is added even if the separator is absent between the elements.

Sample:

```
Mask: " G[2] X[100.101]Y[234.89] Z[45.67]"
NC code: " G2 X100.101 Y243.890 Z045.670"
```

In the line, the space is added between the first and the second elements. The spaces quantity before the first between two and three, three and four is not changed.

If parameter “**Spaces between commands**” is off then all spaces between elements will be removed independently of its quantity.

Sample:

```
Parameter "Spaces between commands" is off.
Mask: "G_INTERP[INTERP] X[CLD.X] Y[CLD.Y] Z[CLD.Z]"
NC code: "G1X-49.47Y-6.513Z.033"
```

All spacebar is removed from the NC-program line.

2.2.7 Variables assignment from the mask

It is possible if necessary to assign any predefined variables or **COMMON** variables from the mask. Values can be assigned from the number and string variables. The constants only can be assigned to the string variable. Any expression including functions and other variables can be assigned to the number variable.

The variable values can be assigned in any line of the mask. NC program line is not generated for this mask line. The “,” sign is the delimiter between the variables.

Sample:

```
Mask: deferred
"INTERP=0;XP=XT;YP=YT-Y1+2;CLDATA$="test""
"G_INTERP[INTERP] X[XT] Y[YT] Z[ZT] ([CLDATA$])"
NC Code: "G0 X100.122 Y231.567 Z010.546 (test)"
```

2.2.8 Deferred masks

Sometimes it is necessary to use the current parameters of the current command when other command is processed. There are two ways to solve this task. The first way is to save the parameters to the temporary variables and use it later in the required command. The second best way is the using of the deferred masks. The current parameters of these masks are saved and the mask execution is deferred to the required program execution. When the deferred mask is executed, the current parameters are replaced by the saved parameters.

The deferred mask must be in the braces "{" and "}". Inside the braces necessarily to define one of the **CLDATA** commands. The mask is executed then this command appears. The lines before this command executed before command execution and the lines placed after the defined command are executed after the command execution

Sample:

```
Mask of command SafPos:
{
G_INTERP[0] X[XT] Y[YT] Z[CLD.Z]
G_FUNC[28] X[CLD.X] Y[CLD.Y]
LoadTL
G_INTERP[1] G_FUNC[29] G_LENGTHCOMPENS[43] X[XT] Y[YT]
G_LENGTHCOMPENS[49] Z[ZT]
}

Mask of command LoadTL:
H[CLD.N] T[CLD.N]

Mask of command AbsMov:
X[CLD.X] Y[CLD.Y] Z[CLD.Z]

CLData commands:
GOTO.abs X 20.0000, Y 20.0000, Z 20.0000
SAFPOS X 10.0000, Y 11.0000, Z 12.0000, N 0
LOADTL N 1, X 0.0000, Y 0.0000, Z 0.0000, D 1.0000, M 0,
K 0, L 0.0000, P 0.0000, A 0.0000, R 0.0000, PLANE XY(33),
Dur 0.0000

NC Code:
X20.0000 Y20.0000 Z20.0000
G0 Z0.0000
G28 X0.0000 Y0.0000
H1 T1
G1 G29 G43 X20.0000 Y20.0000
G49 Z20.0000
```

In this case, when the **SafPos** command is executed then the deferred mask is created. The values of **CLD.X**, **CLD.Y**, **CLD.Z** of **SafPos** command are saved and the mask execution is deferred to the LoadTL processing. Values of **XT**, **YT**, **ZT** is not saved because it is not the CLData command parameters. Before the **LoadTL** processing the mask below is executed

```
G_INTERP[0] X[XT] Y[YT] Z[CLD.Z]
G_FUNC[28] X[CLD.X] Y[CLD.Y]
```

After that, execute

```
LoadTL: H[CLD.N] T[CLD.N]
```

Finally execute:

```
G_INTERP[1] G_FUNC[29] G_LENGTHCOMPENS[43] X[XT] Y[YT]
G_LENGTHCOMPENS[49] Z[ZT]
```

If write the "**Keep**" identifier before the right brace then deferred mask will be always performed then the required program is executed (It is LoadTL in the previous sample)

Sample:

```
Mask of command SafPos:
{
G_INTERP[0] X[XT] Y[YT] Z[CLD.Z]
G_FUNC[28] X[CLD.X] Y[CLD.Y]
LoadTL
G_INTERP[1] G_FUNC[29] G_LENGTHCOMPENS[43] X[XT] Y[YT]
G_LENGTHCOMPENS[49] Z[ZT]
Keep
```

```
}
```

In this sample, the execution of the deferred mask will be performed every time when the LoadTL is processed.

2.3 MASK MANAGEMENT

2.3.1 Mask Switches

Any mask is linked with the three switches:


6. **Out previous.** If the switch is tick off then the NC-line is formed before the mask analyzing. All registers in which the old and current values are not equal will be out to the line.
7. **Poll all registers.** If the switch is checked then all registers in which the old and current values are not equal will be included in to the line in addition to the registers of the mask. If such registers is present (it is possible if the **Off** modifier was used in the previous mask) then this register will be located in the line according to the order in list of registers.

Sample:

```
Registers list: G_INTERP, G_PLANE, X, Y, Z
Plane mask: "G_PLANE[ARCPLANE,Off]"
AbsMov mask: "G_INTERP[1] X[CLD.X] Y[CLD.Y] Z[CLD.Z]"
CLData commands :PLANE XY(33)
GOTO.abs X100 Y100 Z100
NC code: "G1 G17 X100.000 Y100.000 Z100.000"
```

8. **Out block.** If the switch is not checked then all registers written in the mask will be modified but the NC-program line is not out.

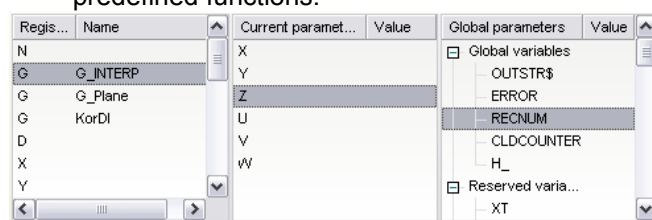
2.3.2 The transformation of a Mask to the Subprogram

The situation is possible then the mask possibility is not enough. To use more powerful and flexible subprograms without manual mask programming, there is the function "**Transform the mask to the subprogram**". Use the  button to transform the mask. The button is located near the mask switches.

To undo the transformation it is necessary to move by clipboard the mask line from the subprogram editor to the mask editor and to set the switches.

2.3.3 The interactive to create the masks

There are the three lists for the comfortable masks editing. These lists are placed under the mask editor. The left list is the list of the registers. The second list is the list of the current CLData command parameters. The right list is the list of the global variables and predefined functions.



The double click on the registers list inserts the selected register name to the cursor position. The chars "[]" added after the register name and the cursor moves to the position between.



The double click on the current parameters list inserts the CLData parameter name to the cursor position. Prefix CLD is added to the parameter.

3 LANGUAGE DESCRIPTION

3.1 BASIC DEFINITIONS

3.1.1 Conditional indications

The following rules and conditional indications are used in this document to describe the statements formats:

- Capital characters are used to indicate the reserved words;
- Line characters are used to indicate the names, the variables, numerical or arithmetical expressions;
- Linguistic expressions are indicated by <..> symbols;
- Unnecessary parts of the operators are indicated by {...} symbols;
- Alternative parts of the operators are divided by | symbol.

3.1.2 The processing programs of the technological commands, the comments in the programs

Each processing program of the technological commands begins by the header, which consists the <**PROGRAM**> keyword and the program name, and terminates by the keyword <**END**>. The program name is coincident with the command name to be processed. When the program is called, the parameters are passed on it via predefined CLD array. Parameters values for the different CLData commands are described in the Chapter 4.

These programs are designed using the special problem-oriented language and can contain the mathematical expressions and functions, the statements for input/output, conditional statements, cycles, jump statement, calls of subroutines, the statements to form the NC-program blocks and the statements to work with the file of the technological commands.

The text of the program can contain statements and comments. Any statement in the same string is allowed. In this case they must be divided by the < ;> symbol. The carrying of a statement part to another string is allowed if the statement is not logically finished. The comments are used to insert the explanative records in the text of program. The comment – is a text, which begins by <!>-symbol and terminates by end of string. It is not allowed to carry the comments to the next string.

The first executable program is the **COMMON** program, next-**PARTNO**, the last - program, which process the **FINI** command.

The peculiarity of the **COMMON** program is the global variables definition in it; the values of these variables are accessible in all programs. Each program can define the variables for its needs, but these variables are local and, consecutively, are accessible in this program only

3.1.3 Subprograms

Subprograms can be written in the same language like the processing programs. Write the **“CALL”** operator to call the sub program

Every subprogram is started from the header that is contains the key word **“SUB”** and the subprogram name. The key word **“SUBEND”** must be written in the end. The Subprogram name cannot coincide with the existed subprograms.

3.1.4 The language statement conception

The statement is a functional unit of the language. The statement formats are defined uniquely by the set of the available linguistic constructions and definition methods. The examples of the statement on the postprocessor language are:

```
PRINT "symbol" - prints the literal string "symbol";
JUMP 1 - jumps to the label 1.
```

The operators of the language can contain the identifiers, numbers, literal strings and auxiliary symbols.

3.1.5 The set of symbols

The following symbols are used for language statements designing:

a..z, A..Z - Capital and line letters of Latin alphabet;

0..9 - numerals from 0 to 9;

— - the underlining;

- the space;

,

- the comma;

= - the equal;

;

- the semicolon;

:

- the colon;

[- left bracket;

] - right bracket;

“ - double quotation marks;

(- left parenthesis;

) - ruling parenthesis;

+ - the plus;

- - the minus;

* - the asterisk;

/ - the slash;

\ - the backslash;

^ - the pointer;

\$ - the dollar;

< - left angular bracket;

> - ruling angular bracket;

- not equal;

. - the point;

@ - commercial A.

The comments and literal strings allow any symbol.

The identifier – is a unceasing letters sequence, numerals and symbols `<_>`, `<@>` and `<$>`, and must begin from the letter. Capital and line letters are indistinguishable in the identifiers.

The number – is an unceasing sequence of numerals, containing only one decimal point. The signs `<+>` or `<->` are allowed before the number. As default, the number is interpreted as positive.

The literal strings – are the arbitrary symbols sequences, marked by the double quotation marks.

3.1.6 The variables

The variables, used by the language, are divided into three types:

- **Integer.** Used for the representation of the integer values. Range: $-2147483648 \div 2147483647$.
- **Real.** Used for the representation of the number with the fractional part. Range: $\pm 2.9 \cdot 10^{-39} \div \pm 1.7 \cdot 10^{38}$ with 11-12 digits in mantissa.
- **String.** The char sequence with the maximal length $\sim 2^{31}$.

3.1.7 Arrays

An array is a data structure that allows storing under alone names the list of numerical variables. The array has the name, type of variables, size (number of elements) and elements numeration. The array is one-dimensional.

The usage of array is available anywhere, where the usage of numerical variables is allowed. To access any element of the array, it is necessary to specify the name of the array and the element index. Any number or numerical variable can be used as a index.

Declaration of the array variable has the sign:

```
<array name>: array <size> of <element type>
```

Sample:

```
V: array 10 of Real
V[5] = 10.67
i = 7
V[i] = 0.987
```

Maximal array size is unlimited, but it is not recommended to use the baseless huge size because the memory is used.

The minimal index value is 1. The maximal index value is limited by the array size.

The array with the beforehand known size is static array. It is possible do not specify the array size then array is dynamic. The size of dynamic array is defined by the maximal index of the filled element. The indexes of the dynamic array start from one too. Maximal value of the dynamical array index is not limited but it is not recommended to use the baseless huge indexes because the memory is required.

Sample:

```
R: array of Integer
R[17] = 10
```

3.1.8 Mathematical expressions and functions

Mathematical expression – is the exact operations description, which operations produce numerical value. They are the analogs of mathematical formulas

Syntactically, the math expression is the combination of the numbers, numerical variables and numerical functions, separated by the signs of the math operations and the parentheses. The simplest examples of math expressions are the number and the numerical variable following math operations are allowed in the language:

- + the addition
- - the subtraction
- * the multiplication
- / the division
- ^ the involution

It is necessary to remember, that two operational sign cannot follow one-after-another.

Following math functions are allowed:

- SIN(x) sinus of angle x (in degrees);
- COS(x) cosine of angle x (in degrees);
- TAN(x) tangent of angle x (in degrees);
- ATN(x) arctangent of angle x (in degrees);
- ASIN(x) arcsine of angle x (in degrees);
- ACOS(x) arccosine of angle x (in degrees);
- SQR(x) square root of x;
- ABS(x) absolute value of x;
- SGN(x) sign for x;
- ROUND(x, y) rounding of x to y decimals after point;
- LOG(x) decimal logarithm of x;
- LN(x) natural logarithm of x
- CHR (x) return a char, corresponding to number x (ASCII-code);
- ORD (Str) return a number (ASCII-code) of the first symbol of the string S;
- LEN (Str) the length of the string Str; defines the chars quantity in the string.;
- POS (Pat, Str) position of the start of the string Pat in the string Str. The result of the function is the position of the first substring Pat in the string Str. If there is no Pat in the Str then the function returns zero.
- NUM (Str) Use Num to convert a string, Str, to a floating-point value. If Str contains inadmissible chars for the number then the error is raised and the function return zero.
- STR (n) Converts a number n to the string;
- COPY (Str, n, m) Returns a substring of a string. Substring is started from the m and has the n chars.;
- UPCASE (Str) converts all characters of the Str to uppercase.;

where

x, y - arguments that can be number, variable of expression;

Str, Pat - literal strings, variables or string expressions. The string expression is a combination of the literal strings, literal variables and literal functions united by sign “+” and parentheses;

n, m - integer numbers or variables.

3.1.9 Predefined variables

Following predefined variables are used in the language:

- **Registers.** The current register values are stored in these variables. The type of these variables is real..
- **Previous values of the registers.** It is possible to obtain the previous values of the registers before writing the current block. In this case the identifier consists of register identifier and of <@> symbol at the end.
- There is a **predefined array of the real numbers**, named **CLD[n]**, where n – identifier of the array element – positive integer number. Identifier of the array element n can be the variable or the expression. The command parameters are passed in the program via CLD array.
- The **RecNum** variable contains the parameters number of the current technological command. I.e., the maximal index of the CLD array is stored in this variable. The usage of this variable is necessary, first of all, in the PPFUN command, because this command has unfixed parameters number.
- The **CLDCounter** variable contains the number of the current command in the technological file. It is used by SEEK statement for positioning in the technological file.
- The **CLDATA\$** variable. If the parameter of the command is not the array of real numbers, but the literal string, then this string is passed in the processing program via this variable. This string stores the value before moment, when it will be redefined by the next command. The type of this variable is string.
- The **OutStr\$** variable. The current NC-program block text is stored in this variable after the **FORMBLOCK** and **OUTBLOCKS** statements. The type of this variable is the string.
- **Xt, Yt, Zt** variables – current position of the tool. The type of these variables is real. Variables are filled automatically before the call of the next subprograms: **CIRCLE, FROM, ABSMOV, INCMOV, ABSMLT, INCMLT, GOHOME**.
- **Xp, Yp, Zp** variables - previous position of the tool. The type of these variables is real. Variables are filled automatically before the call of the next subprograms: **CIRCLE, FROM, ABSMOV, INCMOV, ABSMLT, INCMLT, GOHOME**.
- **Xc, Yc, Zc** variables – coordinates of the circle center. The variables are filled automatically before the call of the **CIRCLE** subprogram.
- **Interp** variable - current interpolation. The possible values are the next: "0" – rapid movement, "1" – linear interpolation, "2" – CW circular interpolation, "3" – CCW circular interpolation. The value is filled automatically before the call of the next subprograms: **CIRCLE, FEDRAT, ABSMOV, INCMOV, ABSMLT, INCMLT, RAPID, GOHOME**.
- **ToolRad** variable – Tool radius. Variable is filled automatically before the call of **LOADTL**.
- **Error** variable - integer variable. If some errors raised while program execution then it contains the error number.

- **ArcPlane** variable – current plane of the circle interpolation. The next values are possible: “33” – XY, “37” – YZ, “41” – XZ. The variable is filled automatically before the call of **PLANE**.
- **Feed** variable – current feedrate value. The variable is filled automatically before the call of **FEDRAT**.
- **TIComp, TrComp** variables – length offset number and radius offset number correspondently. Variables are filled automatically before the call of **CUTCOM** and **LOADTL**.
- **FromX, FromY, FromZ** variables – coordinates of the “HOME” point. (Filled automatically before call of **FROM**).
- **CurCode** variable – the code of the current **CLDATA** command (see the command codes in chapter 4, table 1).
- **NCName\$** - string variable contains the NC output file name without path and extension.
- **NCPATH\$** - string variable contains the NC output file name with full path and extension.
- **SPPName\$** - string variable contains the postprocessor file name without path and extension.
- **SPPPath\$** - string variable contains the postprocessor file name with full path and extension.
- **BlockStep** variable contains the increment value that is used for the line number updating.

3.1.10 Predefined functions

Following predefined functions are used in the language:

NextCode - returns the code of the next technological command;

FlagIn - return “1”, if there is the inner corner between the current and next command. Return 0, if there is the outer corner between the current and next command. Return –1 if the direction of the current and the next command is collinear.

Cross - return “0”, if current element is conjugated with the next ones and returns 1 if the elements cross. This function is used for the compensation;

NextToolNum - returns the number of the next tool;

ToolChange - return the flag of tool changing (Is where the **LOADTL** command in the next CLData commands):

- 1 – Tool change command (**LoadTL**) didn't executed still. But will be or in executing.;
- 0 – LoadTL command called yet and will be called again;
- -1 – this is the last LoadTL command in the CLDATA list

The functions result for the sample CLData commands list is shown below:

CLData commands sequence	ToolChange result
PARTNO	1
...	1
...	1

...	1
LOADTL	1
...	0
...	0
...	0
LOADTL	0
...	0
...	0
...	0
LOADTL	-1
...	-1
...	-1
...	-1
FINI	-1

CurDate - return the string with the current date;

CurTime - return the string with the current time.

GetCLDStr - return the **CLData** string of the current command with parameters.

3.2 OPERATORS

3.2.1 The processing program start operator: **PROGRAM**

Each processing program of the technological commands begins by the header, which consists the **PROGRAM** keyword and the program name, and terminates by the keyword **END**.

Format:

```
PROGRAM ProgramName
  <statement 1>
  ...
  <statement N>
END
```

Description:

The text of a program is written on the special problem-oriented language and can contain the mathematical expressions, functions, input/output operators, conditional operators, cycles, jump statement, calls of subroutines, the statements to form the NC-program blocks and the statements to work with the file of the technological commands.

Sample:

```
PROGRAM AbsMov
  FormBlock
  X = cld[1]
  Y = cld[2]
  if X <> LastX or y <> LastY then begin
    xs$ = Str(40*x)
    ys$ = str(40*y)
    if IsRapid> 0 then fs$ = "PU"
    else fs$ = "PD"
    output fs$ + xs$+", "+ys$+";"
    LastX = X
    LAstY = Y
  end;
END
```

3.2.2 The statement of assignment =

This statement is used to assign the value of an expression to the variable. The variables may change their values because of performing of this statement.

Format:

```
<numerical variable> = <mathematical expression>
or
<string variable> = <string expression>
```

Description:

The keyword of this operator is the variable name. The symbol <=> follows after the name, and then follows the expression, the value of this expression will be assigned to variable. This operator defines that

the expression interpretation result must be stored in the memory cell, identified by specified variable. The variable type must be coincident with the type, returned by the expression, i.e. the value of the math expression must be assigned to the numerical variable, and the value of the string expression – to the string variable.

Note: *When the statement is performed, the value of right expression is evaluated first, and then, retrieved value will be assigned to the variable in the left part of statement. This rule allows using the same variable in the left and right parts of statement. In this case, the old value of variable participates in the evaluation of the right expression and, after this; the value of right expression will be assigned to the same variable.*

Samples:

```
! Usage of the assignment statement
Zet = 41
Ddel = Mn / cos(Betta) * Zet
Number = Number + 5
! The generation of string about the NC-machine
System$ = "2C42"
! The assignment of CNC-name to the System$
NAME$ = "TEST PROGRAM " + System$.
```

3.2.3 The output statement PRINT

This statement is designed to write the results in the debug window during the trial NC-program generation. If the run-time postprocessor (InpD.dll or SprutPP.exe) generates the NC-program, it ignores this operator.

Format:

```
PRINT <math expression>|<string expression> {, <math
expression> | <string expression> {, ...}}
```

Description:

The keyword of this statement is the **PRINT** word. The list, which contains one or more math or string expressions, follows the keyword. The number of expressions is not limited in this statement, if there are more than one expressions in the statement, they must be separated by commas. The value of expressions will be consecutively evaluated and printed in the debug window.

Samples:

```
PRINT "Runs the subprogram ABSMOVE "
PRINT CLD[1]-XT, " increase by X "
PRINT AA, " ", CLD[1]
```

3.2.4 The input statement INPUT

This operator is designed to input data from the keyboard when the program is running.

Format:

```
INPUT <numerical variable> | <string variable> | <literal
string> {, <numerical variable> | <string variable> | <literal
string> {, ...}}
```

Description:

The **'INPUT'** is the keyword for this statement. The list, which contains one or more math or string expressions, follows after the keyword. The number of expressions is not limited in this statement, if there are more than one expressions in the statement, they must be separated by commas.

The input window, which contains literal strings and input fields, will be constructed as a statement result. The checking of correspondence of inputted values to types of its variables is performed during the parameters input. Only numbers can be inputted in the numerical variable, any symbol can be inputted in the string variable. It is necessary to close the window after the input by pressing the **<OK>** button. After that, the inputted values will be assigned to the corresponding variables.

This statement is analogue to the assignment, but the values of variables will be asked each running of program. This allows obtaining the new results of the same program by inputting various data.

Samples:

```
INPUT "The initial value of coordinate Z: ", Zt
INPUT "Input the NC-machine name: ", NCNam$
INPUT "Workpiece dimensions ", "Along X axis ", nx2, "Along Y
axis ", ny2, "Height by Z", nz2
```

3.2.5 Conditional statement IF

This statement is designed to perform one of statements, included by it, in depending of some condition.

Format:

```
IF <conditional expression> THEN <Statement1>
    {ELSE <Statement2>}
```

Description:

This operator allows changing the next steps of program dependently of some conditions

The keyword of this statement is the **'IF'** word. Then follows the conditional expression, when it is true, the statement, following the **THEN** keyword, is performed. If the conditional expression is false, then the statement, following the keyword **ELSE**, is performed. **ELSE** – is unnecessary part of this statement. If it is absent and the value of conditional expression is false, then the performing of IF- statement terminates and next statement of a program will be performed. **<Conditional expression>** may be a single **<Simple conditional expression>** or a sequence of the **<Simple conditional expressions>**, linked by logical **<OR>** and **<AND>** operations, i.e.:

Format:

```
<simple conditional expression> {{AND <simple conditional
expression>} | {OR <simple conditional expression>}}
```

<Conditional expression> is true, if all **<Simple conditional expressions>**, linked by **AND** operation are true or if at least one of **<Simple conditional expressions>**, linked by **OR** operation is true.

The **<Simple conditional expressions>** - is a two math or string expressions, linked by comparison operations, i.e.:

Format:

```
<Math expression> <Comparison operator> <Math expression>
or
<string expression> <Comparison operator> <string expression>
```

The expressions around the **<Comparison operation>** must be same type; otherwise, you will obtain the error message **<Incompatible types>** during the compilation.

The **<Comparison operation>** is one of following operations:

=	- the values of left and right operations are equal
>	the value of left expression is greater than value of right expression
<	the value of left expression is less than value of right expression
# or <>	the values expressions are different
>= or =>	the value of left expression is equal or greater than value of right expression
<= or =<	the value of left expression is equal or less than value of right expression

It is necessary to pay attention to the comparison of string expressions. The comparison of two strings is performed character-by-character from left to right respecting their ASCII-codes. If the lengths of the strings are different, but the longest string includes the shortest entirely, then the longest string is greater. The strings are equal, if they contain the same symbols and have the same length.

When the conditional statement is performed, the values of math and string expressions in the **<Simple conditional expressions>** are evaluated first, then the operations of comparison are performed, and then the **AND** and **OR** operations are performed without priority. After that, the **THEN** or **ELSE** statement will be performed, depending the result of entire **<Conditional expression>**.

Note: *The logical operations performing sequence can't be changed by using the parentheses. It's necessary to use the sequence of multiple conditional statements in this case. It is possible always to replace the sequence of logical operations by the sequence of conditional statements. When one conditional statement appears into another, **ELSE**-statement corresponds to nearest (along the program) **IF**-statement. The part of conditional operator can be carried in the next string, but the carrying from the **ELSE**-keyword isn't allowed, because in this case the first part of statement (**IF-THEN**) will be logically finite, and **ELSE** will be interpreted as a beginning of next statement.*

Samples:

```
! Example 1.
! The example, which uses the conditional statement.
IF kadr <224 THEN PRINT "This is a short program "
ELSE          PRINT "You are the monster in programming"

! Example 2
! This example shows how to use embedded
! conditional statements
IF POS(" ", Str$) = 2
  THEN b$ = " the first char in string is a letter "
ELSE
  IF POS(" ", Str$) <= 4
```

```

THEN b$ = " first in string " +
        " is a word, its length is 4 symbols"
ELSE b$ = " first in string is a word greater" +
        "than 5 symbols length "

```

3.2.6 Statement of the multiconditional execution **CASE**

The Case statement allows checking the value of the expression and depends of the result to execute different code.

Format:

```

CASE <expression> OF
    <values list 1>: <statement 1>
    ...
    <values list N>: <statement N>
ELSE <statement M>
END

```

Description:

The expression must be numerical in this construction. Inadmissible to use string expressions.

The values lists can contain one or more constants, divided by commas. The ":" symbol follows the values list. After that the statement is written that is executed if the expression is equal to one of the list. After the execution of this statement the execution of the **CASE** structure is finished and next statement is executed.

If the expression result is not defined in all lists then the statement after **ELSE** keyword is executed. The **ELSE** part is an optional. If **ELSE** absent and there is no corresponding value in the lists then nothing is executed.

Sample:

```

CASE i OF
    1,2,3,4,5: Str = "less or equal 5"
    6,7,8,9: Str = "greater 5"
    else Str = "Error value"
END

```

3.2.7 JUMP statement

The operator is used to break the order of statements performing.

Format:

```

JUMP <Label number>

```

Description:

After the '**JUMP**' keyword one of labels, which is defined in the current program, must be specified. The symbol <:> mustn't appear after the label number, unlike the label definition.

The operator, labeled by corresponding label, will be performed after the **JUMP** statement. It is necessary to define corresponding label, otherwise the error message will appear during compilation.

It is not recommended to usually use the **JUMP** statement in the program.

Sample:

```

AFF = 20  ! this statement is performed fist
JUMP 2    ! jump to label 2
ABC = 1   ! This statement never be performed
2: ATR = 4 ! this statement is performed second

```

3.2.8 The cycle statement FOR

This statement repeats the performing of specified statement specified number of times.

Format:

```

FOR <numerical cycle control variable> = <math expression> TO
<math expression> {STEP <number> | <number variable>} DO
<operator>

```

Description:

This statement provides repetitive performing of some statement and simultaneous incrementing the cycle variable, until the value is greater than specified limit.

The keyword of this operator is the **<FOR>** word. Then follows the construction, which is similar to assignment: **<Numerical cycle control variable> = <Math expression>**. In reality, they are much similar, because this construction assigns the initial value, defined by **<Math expression>**, to **<Numerical cycle control variable>**.

After the 'to' reserved word follows the math expression, which defines the upper limit. When the **<Numerical cycle control variable>** achieves this limit, the cycle statement terminates, and next program statement will be performed. Any math expression can be specified as an upper limit for cycle.

Unnecessary part of this statement, containing reserved word **<STEP>** and following number or numerical variable, defines the incrementing step for the cycle variable. As default, the step is 1. The expression cannot be the step

The statement, which must be performed cyclically, is specified after the **DO** – keyword. All other cycle parameters serve for organization of cyclical performing of this statement.

Samples:

```

! Example 1.
! Simple example of usage the cycle statement.
FOR as = 3 TO 10 DO PRINT as:0," ",as^2:0

! Example 2.
! Using embedded cycle statements
FOR i = 0 TO 0.9 STEP 0.5 DO
  FOR j = -1 TO 0 STEP 0.2 DO
    PRINT " i = ",i:1," j = ",j:1

```

3.2.9 The cycle statement REPEAT

The structure repeat...until is used for the repeating execution of the some operators named as cycle body, until some condition will be true.

Syntax of the structure repeat...until:

```

REPEAT
  <operators of the cycle body>
UNTIL <conditional expression>

```

The structure works the following way. The operators of the cycle body is executed, after that is calculated the conditional expression. If the conditional expression returns false then the execution of the cycle body is repeated and after that, the conditional expression is calculated again. So the repeating is continues until the conditional expression returns true. After that, the cycle execution is finished and the next statement after structure repeat...until is executed.

Because the condition checking is performed after the execution of the cycle body then it is executed one time even if the condition is true in advance. On the other hand, the condition must return true one time else, the loop will be performed infinitely.

3.2.10 The cycle statement WHILE

The structure while...do is used for the repeating execution of the some operators named as cycle body while some condition is true. The structure while...do have the following syntax:

Format:

```

WHILE <conditional expression> DO <statement>

```

The structure works the following way. At the first, the conditional expression is calculated. If the result is true then the cycle body is performed and after that, the conditional expression is calculated again. So the repeating is continues until the expression returns false. After that the cycle execution is finished and the next statement after structure **while...do** is executed.

Because the condition checking is performed before the execution of the cycle body then if the conditional expression is false in advance then the cycle body is never be performed. It is the main difference between the structures **while...do** and **repeat...until**. On the other hand the condition must return true one time else the loop will be performed infinitely.

3.2.11 Composite statement BEGIN ... END

if is used of the uniting of the group of statements to a one statement. It is the analogue of the parentheses

Format:

```

BEGIN
  <statement 1>
  ...
  <statement N>
END

```

Description:

The composite statement provides the execution of the included statements step by step. The auxiliary words **BEGIN** and **END** are analogue of the parentheses. The multiplicity level of the composite statements is unlimited. The composite statement can be used everywhere where a simple statement can be used.

Sample:

```

! Simple example of the composite statement

```

```

IF var <3 THEN BEGIN
    ac =12; bb =16
END ELSE BEGIN
    ac=15; bb=60
END ! word END close the statement

```

3.2.12 Statement to call a subprogram **CALL**

The operator is intended to call the subprogram from the CLData command-processing program or other subprogram.

Format:

```
CALL <program name>
```

Description:

The keyword of this operator is the **<CALL>** word. After this word it is necessary to define the **<program name>** - string expression or the literal string without the double quotation marks. The expression value is the name of the subprogram.

The **CALL** jumps the execution to the defined subprogram and after the execution makes the return to the program, the call was raised from.

Sample:

```
CALL SUBTASK
```

3.2.13 The statement of the subprogram start **SUB**

It is used for the declaration of the subprogram and the parameters list that is passed to the program then call.

Format:

```
SUB <subprogram name> {( <the list of the parameters> )}
```

Description:

The keyword of this operator is the **<SUB>**. Then follows the subprogram name – literal string without the double quotation marks and after that the optional parameters list in the parentheses.

The list of the parameters is the sequence of the numerical and string variables or arrays. If the parameters number is more the one then it is divided by commas.

The variables defined if the list will contain the values defined in the call statement when call. Therefore, these variables if declared in subprogram and can be used everywhere.

3.2.14 The statement of the subprogram end **SUBEND**

The statement is used for the declaration of the end of the subprogram.

Format:

```
SUBEND
```

Description:

The statement is declaring the end of subprogram. At the **SUBEND** execution the subprogram execution is closed, the values of all variables defined in the parameters list is assigned to the variables defined in the **CALL** statement and the execution of the parent subprogram is continued.

Sample:

```

SUB EndCycle ! Subprogram start
  if Cld[1] = 163 then L = 81 else           ! Drill
  if Cld[1] = 168 then L = 84 else           ! Tap
  if Cld[1] = 209 then L = 85 else           ! Bore5
  if Cld[1] = 210 then begin                 ! Bore6
    L = 86
    R4 = Spin
  end
  if Cld[1] = 213 then
    begin                                   ! Bore9
      if Cld[7] = 279.0 then R3 = Cld[8]
      L = 89
    end
  end
SUBEND ! Subprogram end

```

3.2.15 The statement of the procedure start PROC

The operator is intended to declare the start and the parameters list of the subprogram in the **CLDATA** command processing procedure.

Format:

```
PROC <procedure name> {(<list of formal parameters>)}
```

Description:

After the **PROC** keyword, the procedure name is declared. The **<procedure name>** - identifier it is not coincide with the postprocessor generator keywords and variable names. This name is for the procedure identification for calling, so the names of all procedures must be unique. After the **<procedure name>** follows the optional **<list of formal parameters>**. It is the sequence of the numerical and string variables and arrays. If the number of parameters is more then one then comma must divide it.

As distinct from **SUB** statement the variables defined in the **<list of formal parameters>** is not initialized i.e. it is necessary to assign some values to the variables before the including to the list.

The variables declared in the **<list of formal parameters>** will contain the values that are defined in the **CALL** statement. So these variables is defined in a program and can be used everywhere.

The **PROC** statement is looks like **SUB** statement. The main difference is that the procedure has the access to all variables and arrays of program of subprogram in the body of which it is located.

It is possible to place the declared procedure in any place of a program. See the example of the procedure in the topic 3.2.16.

3.2.16 The RETURN from a procedure statement

It is used to declare the end of procedure.

Format:

RETURN

Description:

This statement is the last statement of a procedure. While the execution of the **RETURN** statement the back assignment of the formal parameters to the variables defined in the **CALL** statement is executed and start the execution of the next after the **CALL** statement.

Sample:

```

program TestProc
  NErr: Integer
  S: String

  proc PrintDebugInfo(NErr, S)
  case NErr of
    1: Print "Interpolation error: ", S
    2: Print "Approximation error: ", S
    else Print "Unknown error: ", S
  end
  return

  PrintDebugInfo(CLD[1], "parameter 1")
  PrintDebugInfo(CLD[2], "parameter 2")
  PrintDebugInfo(CLD[3], "parameter 3")
end

```

3.2.17 The block output statement OUTBLOCK

This statement forms the content of a block, corresponding to specified variable format, order, register format, and outputs the block into NC-program.

Format:

OUTBLOCK

Description:

The block of NC-program is formed according following algorithm: The registers are checked from the first. If the previous and current values of register are different, then the register will be written in the block, and its current value will be assigned to its previous value; otherwise the register won't be written in the block.

When the register is written in the block, its identifier will be written first, then its value, multiplied by the scale. The number will be written in the block corresponding to specified format. After that, the processing program will be called, if it is defined. The block is formed in the variable **OutStr\$**, and will be written in the program as separate string after the formation is done.

3.2.18 The block forming statement FORMBLOCK

This statement forms the block of NC-program according specified format, order and registers format. The block is formed in **OutStr\$** variable without the output in the file of NC-program.

Format:

FORMBLOCK

Description:

The formation process is similar to the **OUTBLOCK** statement, but the **OutStr\$** variable won't be written in the block of NC-program.

It's possible, then, to work with this variable as a string variable and put it into NC-file by **OUTPUT** statement.

3.2.19 Statement of direct output into the block **OUTPUT**

This operator outputs the content of a string variable by separate block of NC-program.

Format:

```
OUTPUT S$
```

Description:

S\$ - the identifier of a string variable. The literal constant or string expression may be the parameter of this operator.

Samples:

```
! The output of the first symbol into NC-code
OUTPUT "%"
Name$ = " John "
F$ = "Lord"
! The output of sting into NC-program
OUTPUT " Programmer "+Name$+" "+F$
! The string <Programmer John Lord> will be written into NC-
program

! Sample of usage the
! FormBlock and Output instead OutBlock
FORMBLOCK
OUTPUT OutStr$
! Two these statements work as OutBlock
```

3.2.20 The replace statement of the substring in a string **REPLACE**

The statement is looking for the substring in a string and replace to the required.

Format:

```
REPLACE (<string variable>, <looked string>, <string for
replace>)
```

Description:

This operator is looking for the **<looked string>** from the start of **<string variable>**. If **<looked string>** is found then it is replaced on the **<string for replace>** and the search is break. if the **<looked string>** is not found then replacement don't take the place.

Sample:

```
! Filling the source variable
S$ = "GXX"
! Search and replace "XX" on "21"
```

```
REPLACE(S$, "XX", "21")
! The variable S$ has the "G21" value
```

3.2.21 The statement to form the block by mask MASK

The Mask statement forms the block of NC-code by mask string. According to the rules and output the string to the NC-program.

Format:

```
MASK (<mask string>) {, OutBlock {, Poll}}
```

Description:

The statement masks form the NC code line according to the rules that is described in the mask string. **OutBlock** option is analog of the **Out block** switch. Poll option corresponds to the **Poll all registers** switch. See chapter 2 "**Masks**" for details.

Sample:

```
! Form the NC code lines by mask
MASK(M[30]), OutBlock
MASK(% N[Off] ), OutBlock
MASK(G_INTERP[INTERP]X[CLD.X]Y[CLD.Y]Z[CLD.Z]),
      OutBlock, Poll
```

4 APPENDICES

4.1 THE CLDATA DICTIONARY (THE DICTIONARY OF THE TECHNOLOGICAL COMMANDS)

Table 1. General keywords

N	Keyword	Code	Value
	CIRCLE	15000	The motion along an arc
	COEFF	2330	Factor of a spine
	COMMENT	1065	Commentaries
	COOLNT	1030	The coolant
	CUTCOM	1007	Tool correction
	CYCLE	1054	The cycle
	DELAY	1010	Pause
	FEDRAT	1009	The feedrate
	FINI	14000	Ending record
	FROM	5003	Original position
	GOHOME	17	Return to original position
	GOTO	5004-5005	Tool motion
	HEAD	1002	The number of spindle heads
	HEADER	3001	
	INCR	66	Absolute or relative coordinate system assuming
	INSERT	1046	Direct output into the block
	LOADTL	1055	Tool loading
	MOVNRB	3041	The beginning of a spline phase of trajectory
	MULTGOTO	9004-9005	
	OPSKIP	1012	The optional skipping
	OPSTOP	2003	The auxiliary stop
	ORIGIN	1027	The original coordinates
	PALETA	1001	Palette changing
	PARTNO	1045	Part number
	PLANE	99	Work plane
	PPFUN	1079	The function of

N	Keyword	Code	Value
			postprocessor
	PPRINT	1044	The printing of postprocessor
	RAPID	2005	Rapid feedrate
	ROTABL	1026	Table rotation
	SAFPOS	1094	Safe position
	SELCTL	1056	Tool selection
	SPINDL	1031	Spindle
	STOP	2002	Stop
	THREAD	1036	Threading
	TRAILR	3004	

Table 2. **Auxiliary keywords**

N	Keyword	Code	Value
	BOTH	83	Both
	BRKCHP	288	Breaking the shavings
	CCLW	59	Counterclockwise
	CLW	60	Clockwise
	CUTS	511	The stepover
	DEEP	153	Deep drilling
	DEEPTH	510	The depth
	DRILL	163	The drilling
	DWELL	279	Pause (delay)
	FACE	81	Drilling G82
	FINCUT	512	Final cutting
	INCR	66	Increment
	LENGTH	9	The length
	MMPM	315	Millimeters per second
	MMPR	316	Millimeters per rotation
	MULTRD	119	Multi-Threading
	OFF	72	Switch off
	ON	71	Switch on
	ORIENT	246	The orientation
	TAP	168	Taping
	XYPLAN	33	Coordinate XY
	YZPLAN	37	Coordinate YZ
	ZXPLAN	41	Coordinate ZX
	R	23	Radius

N	Keyword	Code	Value
	RGT	24	Right position of the tool
	LEFT	8	Left position of the tool

4.2 FORMAT OF THE TECHNOLOGICAL COMMANDS

4.2.1 Part number

Global variables are defined and the first block of NC-program is formed in the program, which processes the PARTNO command. This program is called in the beginning of NC-program certainly; even there is no of PARTNO-command in the technological file.

Command:

```
PARTNO "....."
```

The "CLDATA\$" variable:

```
"....."
```

4.2.2 Ending record

Command:

```
FINI
```

4.2.3 Postprocessor function

Command:

```
PPFUN PPFUN(500) | STARTSUB(50) | ENDSUB(51) | CALLSUB(52) |  
REPSTART(53) | REPEND(54) | JUMP(55) | TECHINFO(58) {, a} {,  
b} {, c} {, d} ...
```

Parameters:

Parameter	CLD array		Description
PPFUN, or STARTSUB, or ENDSUB, or CALLSUB, or REPSTART, or REPEND, or JUMP or TECHINFO	CLD[1]	CLD.SubCode	The identifier of called postprocessor function
a, b, c, d ...	CLD[2] – CLD[257]		The unnecessary parameter list. When calling CALLSUB parameter a is obligatory and means the number of called subprogram

In case CLD [1] = 500 (PPFUN) CLD [2] possess the value stability of the tool in minutes. Such command is shaped when it is necessary to execute change of tools in connection with a wear.

In case CLD [1] = 58 (TECHINFO) optional parameters are listed in the table:

CLD array	Parameter	Value description	Group
1	SubCode	58 (TechInfo)	
2	2	CLData version	
3	3	Minimal coordinate by X	Trajectory shell
4	4	Minimal coordinate by Y	
5	5	Minimal coordinate by Z	
6	6	Maximal coordinate by X	
7	7	Maximal coordinate by Y	
8	8	Maximal coordinate by Z	
9	9	Safe plane	Levels
10	10	Top machining level	
11	11	Bottom machining level	
12	12	Stock	Workpiece (All operation shell)
13	13	Minimal coordinate by X	
14	14	Minimal coordinate by Y	
15	15	Minimal coordinate by Z	
16	16	Maximal coordinate by X	
17	17	Maximal coordinate by Y	
18	18	Maximal coordinate by Z	Tolerance
19	19	CLData tolerance (digits number after point)	
20	20	Measurement units (0 – mm; 1 – inch)	
21	21	Minimal arc length	
22	22	Deviation out of the model	
23	23	Deviation in the model	
24	24	Operation stock	

CLD array	Parameter	Value description	Group
25	25	Tool type: 0 – Cylindrical 1 – spherical 2 – Torus 3 – Double radial 4 – Limited double radial 5 – Conical mill 6 – Limited conical mill 7 – Engraver 8 – Drill	Tool
26	26	Tool number	
27	27	Diameter	
28	28	Work part length	
29	29	Work part angle { conical angle in rad}	
30	30	Round radius	
31	31	Compensation by length number Value equality 0 if corrector is off	
32	32	Compensation by radius number Value equality 0 if corrector is off	
33	33	Compensation by length value	
34	34	Compensation by radius value	
35	35	Programmed point (end/centre) 0 – Centre point 2 – End point	
36	36	Spindle rotation velocity (in RPM)	Speed
37	37	Rapid feedrate value (mm/min or inch/min)	
38	38	Work feedrate	
39	39	Cut-in feedrate	
40	40	Approach feedrate	
41	41	Retraction feedrate	
43	42	Next feedrate	
43	43	Return feedrate	
44	44	Trajectory length	Trajectory statistics
45	45	Machining time (8 byte in Delphi format)	
46	46	All machining time	

CLD array	Parameter	Value description	Group
47	47	First rotary axis: 0 – none 1 – X, 2 – Y, 3 – Z, 4 – Custom	Rotary axis
48	48	First rotary axis position (angle)	
49	49	Second rotary axis: 0 – none 1 – X, 2 – Y, 3 – Z, 4 – Custom	
50	50	Second rotary axis position (angle)	
51	51	Tool axis: 1 – X, 2 – Y, 3 – Z	
52	52	Coordinate X	Tool changing point
53	53	Coordinate Y	
54	54	Coordinate Z	
55	55	Coordinate X	Intermediate point
56	56	Coordinate Y	
57	57	Coordinate Z	

4.2.4 Commentaries

Output comments in NC program..

Command:

```
COMMENT "....."
```

The «CLDATA\$» variable:

```
"....."
```

4.2.5 Linear transition

Depending on an aspect of migration the postprocessor transmits data on migration either in ABSMOV, or in INCMOV, either ABSMLT, or INCMLT.

Command:

```
GOTO.Abs X x, Y y, Z z
GOTO.Inc X x, Y y, Z z
MULTGOTO.Abs X x, Y y, Z z, V v, U u, W w
MULTGOTO.Inc X x, Y y, Z z, V v, U u, W w
```

Parameters:

Parameter	CLD array	Description
x,	CLD[1]	CLD.X New coordinates of instrument

Parameter	CLD array		Description
y,	CLD[2]	CLD.Y	(absolute or relative)
z,	CLD[3]	CLD.Z	
u,	CLD[4]	CLD.U	
v,	CLD[5]	CLD.V	
w	CLD[6]	CLD.W	

4.2.6 Displacement along the circle

It is necessary to pay attention, in what planes goes a processing, since when partitioning a circle on quarters or on halves, the postprocessor takes necessary data from predefined variables Xt, Yt, Zt, and places resulting coordinates of new endpoint in that coordinate axes, in planes which goes a processing. That is to say, if processing is performed in XZ-plane, the it is necessary to trace variables Xt, Zt, and take coordinates of center in CLD(1) for X and CLD(3) for Z, in CLD(5) and CLD(7) - the coordinates of endpoint. Besides, in the postprocessor a radius of circle, sent in CLD(4), is negative in that case, if this circular interpolation clockwise.

Command:

```
CIRCLE XC xc, YC yc, ZTOOL z, R r, XK xk, YK yk, ZTOOL zk
```

Parameters:

Parameter	CLD array		Description
xc,	CLD[1]	CLD.Xc	The coordinates X and Y of circle center
yc	CLD[2]	CLD.Yc	
z	CLD[3]	CLD.Zc	The coordinate Z of a tool
r	CLD[4]	CLD.R	The radius of circle
xk,	CLD[5]	CLD.Xe	The coordinates X and Y of arc endpoint
yk	CLD[6]	CLD.Ye	
zk	CLD[7]	CLD.Ze	The coordinate Z of a tool

4.2.7 Working plane

Depending on working plane, the postprocessor outputs the displacements in corresponding coordinates. In the postprocessor it's needed to have information on that, in what planes goes a machining. These data are necessary in the circle-processing module.

Command:

```
PLANE XY(33) | YZ(37) | XZ(41)
```

Parameters:

Parameter	CLD array		Description
XY, or YZ or XZ	CLD[1]	CLD.Plane	The code of plane: 33 (XY), or 37 (YZ) or 41 (XZ)

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.G, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 33	ISO.G = 17
CLD[1] = 37	ISO.G = 19
CLD[1] = 41	ISO.G = 18

4.2.8 Original point

The original point coordinates are defined. This command meets once, as a rule, and is placed in the beginning of the part machining description. If it is absent, then the coordinates of the tool center original position are X=0, Y=0, Z=0.

If the command FROM is present, then the tool will be returned in the specified original point by the "GOHOME" command.

Command:

```
FROM X x, Y y, Z z
```

Parameters:

Parameter	CLD array		Description
x,	CLD[1]	CLD.X	The coordinates of original point
y,	CLD[2]	CLD.Y	
z	CLD[3]	CLD.Z	

4.2.9 Return to original position

The tool will be returned by this command in the position, defined by FROM command or in the point X=0, Y=0, Z=0, but only on the screen. In the postprocessor is transmitting the command code only.

Command:

```
GOHOME
```

4.2.10 The point of tool change

The coordinates of the point, where the tool center will be positioned for the change of tool (LOADTL command), are defined. Tool change make in the current point as default.

Command:

```
SAFPOS X x, Y y, Z z, N n
```

Parameters:

Parameter	CLD array		Description
x,	CLD[1]	CLD.X	Coordinates of point of tool change
y,	CLD[2]	CLD.Y	
z	CLD[3]	CLD.Z	

Parameter	CLD array		Description
N	CLD[4]	CLD.N	The number of point of tool change in the NC-machine

4.2.11 Table rotation

Command:

```

ROTABL    ABS (0) ,      A      ab,      CCLW (59) | CLW (60) ,      PLANE
XY (33) | YZ (37) | XZ (41)
ROTABL    INCR (66) ,      B      ab,      CCLW (59) | CLW (60) ,      PLANE
XY (33) | YZ (37) | XZ (41)

```

Parameters:

Parameter	CLD array		Description
ABS or INCR	CLD[1]	CLD.Incr	Rotation type: 0 (ABS) absolute, 66 (INCR) relative
ab	CLD[2]	CLD.AB	The angle of table rotation or incrementing of this angle
cclw or clw	CLD[3]	CLD.CCLW	The direction of rotating counterclockwise or clockwise
XY, or YZ or XZ	CLD[4]	CLD.Plane	The code of plane

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.A, ISO.B, ISO.C which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[4] = 33 CLD[1] = 0 CLD[3] = 59	ISO C = CLD[2]
CLD[4] = 33 CLD[1] = 0 CLD[3] = 60	ISO C = – CLD[2]
CLD[4] = 33 CLD[1] = 66	ISO C = CLD[2]
CLD[4] = 41 CLD[1] = 0 CLD[3] = 59	ISO B = CLD[2]
CLD[4] = 41 CLD[1] = 0 CLD[3] = 60	ISO B = – CLD[2]
CLD[4] = 41 CLD[1] = 66	ISO B = CLD[2]
CLD[4] = 37 CLD[1] = 0 CLD[3] = 59	ISO A = CLD[2]

CLD parameters value	ISO value
CLD[4] = 37 CLD[1] = 0 CLD[3] = 60	ISO A = – CLD[2]
CLD[4] = 37 CLD[1] = 66	ISO A = CLD[2]

4.2.12 Tool compensation

Command:

CUTCOR ON(71)|OFF(72), LENGTH(9)|R(23) lr, X x,Y y, Z z, XY n,
YZ m, XZ k ,RIGHT(24)|LEFT(8)

Parameters:

Parameter	CLD array		Description
ON or OFF	CLD[1]	CLD.OnOff	71 – switching on 72 – switching off a compensation
LENGTH or R	CLD[2]	CLD.Length	9 – correction on length or 23 – radius of instrument
LR	CLD[3]	CLD.LR	Corrector number for length or radius of instrument
x, y, z	CLD[4] CLD[5] CLD[6]	CLD.X CLD.Y CLD.Z	Axial corrector number
n, m, k	CLD[7] CLD[8] CLD[9]	CLD.N CLD.M CLD.K	Corrector number for planes
RIGHT or LEFT	CLD[10]	CLD.RGT	24 – right or 9 – left correction

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.G, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 72 CLD[2] = 9	ISO.G = 49
CLD[1] = 71 CLD[2] = 9	ISO.G = 43
CLD[1] = 72 CLD[2] = 23	ISO.G = 40
CLD[1] = 71 CLD[2] = 23 CLD[10] = 24	ISO.G = 42

CLD parameters value	ISO value
CLD[1] = 71 CLD[2] = 23 CLD[10] = 8	ISO.G = 41

4.2.13 Tool loading

On the machining contour by equidistant is used information about tool diameter. Displacement tool for changing in the point referred above by **SAFPOS** command. If **SAFPOS** command was not called then tool change make in the current coordinates.

Command:

```
LOADTL N n, X x, Y y, Z z, D d, M m, K k, L l, P p, A a, R r,
PLANE XY(33)|YZ(37)|XZ(41), FEEDCOLOR c1, RAPIDCOLOR c2,
DURABILITY d2
```

Parameters:

Parameter	CLD array		Description
n	CLD[1]	CLD.N	Tool number
x,	CLD[2]	CLD.X	Tool displacement along the axes LX, LY, LZ
y,	CLD[3]	CLD.Y	
z	CLD[4]	CLD.Z	
d	CLD[5]	CLD.D	Tool diameter
m,	CLD[6]	CLD.M	The numbers of correctors
k	CLD[7]	CLD.K	
l	CLD[8]	CLD.L	Tool length (overhang)
p	CLD[9]	CLD.P	Tool width
a	CLD[10]	CLD.A	Angle of tool approach (under external sample $180 < a < 0$, under internal sample $0 < a < 90$)
r	CLD[11]	CLD.R	The radius of tool rounding
XY, or YZ or XZ	CLD[14]		XY, YZ or XZ - the plane, which is perpendicular to the tool axis
c1, c2	CLD[15] CLD[16]		Tool color for drawing (work feedrate and rapid feedrate correspondingly)
d2	CLD[17]	CLD.Dur	Tool durability (min.), 0 - to leave out of account

4.2.14 Tool selection

Command:

```
SELCTL n
```

Parameters:

Parameter	CLD array	Description
-----------	-----------	-------------

Parameter	CLD array		Description
n	CLD[1]	CLD.N	Tool number

4.2.15 Spindle

Rotation velocity is using for examination switch on spindle in commands: **FEDRAT**, standard cycles and in **DELAY** command, for calculate delay value in **RPM**.

Command:

```
SPINDL ON(71) | OFF(72) | ORIENT(246) | SMM(205), NO n, K k
```

Parameters:

Parameter	CLD array		Description
ON, or OFF, or ORIENT or SMM	CLD[1]	CLD.OnOff	71 (ON) switch on, 72 (OFF) switch on, 246 (ORIENT) oriented stop or 205 (SMM) definition of rotation velocity (in RPM)
n	CLD[2]	CLD.NO	Rotating frequency or angle of spindle tumbling
k	CLD[3]	CLD.K	Range of rotation frequencies

For creation of postprocessors similar **ISO** in masks it is possible to use preconceived values **ISO.G**, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 71	ISO.M = 3
CLD[1] = 72	ISO.M = 5

4.2.16 Stop

Command:

```
STOP
```

4.2.17 Auxiliary stop

Command:

```
OPSTOP
```

4.2.18 Cooling

Command:

```
COOLNT ON(71) | OFF(72), N n
```

Parameters:

Parameter	CLD array	Description
-----------	-----------	-------------

Parameter	CLD array		Description
ON or OFF	CLD[1]	CLD.OnOff	Switch on /off the cooling
n	CLD[2]	CLD.N	Pipe line number

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.M, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 71	ISO.M = 8
CLD[1] = 72	ISO.M = 9

4.2.19 Feedrate

Command is define feedrate value down to the next call FEDRAT command or tool movement with defined feedrate command.

Command:

```
FEDRAT N nm, K k, MMPM(315), COLOR c
FEDRAT M nm, K k, MMPR(316), COLOR c
FEDRAT NM nm, K k, p, COLOR c
```

Parameters:

Parameter	CLD array		Description
nm	CLD[1]	CLD.NM	Feedrate in mm/mines, or in mm/on or other units
k	CLD[2]	CLD.K	Range of feedrates
MMPM, MMPR, or p	CLD[3]	CLD.MMPM	MMPM(315), MMPR(316) - units of feedrate measurement (mm/mines or mm/on). P- Parameter, which specifies the units of feedrate definition
c	CLD[4]		Color for drawing

4.2.20 Rapid feedrate

Command:

```
RAPID N n, COLOR c
```

Parameters:

Parameter	CLD array		Description
n	CLD[1]	CLD.N	The value of rapid feedrate
c	CLD[2]		Color for drawing

4.2.21 Pause

Command:

DELAY A a

Parameters:

Parameter	CLD array		Description
a	CLD[1]	CLD.A	The pause time at seconds

4.2.22 Absolute or relative coordinate system assuming.

After this command, all transitions are interpreted as absolute or relative.

Command:

INCR ON (71) | OFF (72)

Parameters:

Parameter	CLD array		Description
ON or OFF	CLD[1]	CLD.OnOff	Absolute or relative coordinate system

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.G, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 71	ISO.G = 91
CLD[1] = 72	ISO.G = 90

4.2.23 Original coordinates

The postprocessor begins to recalculate all transitions using the value of displacement after definition of displacement of original point.

Command:

ORIGIN X x, Y y, Z z, PPFUN f, N n

Parameters:

Parameter	CLD array		Description
x, y, z	CLD[1] CLD[2] CLD[3]	CLD.X CLD.Y CLD.Z	The displacement coordinates of the original point
f	CLD[4]	CLD.PPFun	Number of postprocessor function
n	CLD[5]	CLD.N	Number of original point

4.2.24 Canceling and recovering a cycle

All standard cycles G81-G92 fall into record CYCLE, so it is necessary in the module to produce an analysis of a cycle type and their processing description.

Command:

```
CYCLE ON (71) | OFF (72)
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	71 (ON) or 72 (OFF) – recovering or canceling cycle

4.2.25 Drilling cycle a type G81

Command:

```
CYCLE DRILL(163), A a, MMPM(315), N nm, F f, P p, T t
CYCLE DRILL(163), A a, MMPR(316), M nm, F f, P p, T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 163 (DRILL)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/min or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.26 Drilling cycle a type G82

Command:

```
CYCLE FACE(81), A a, MMPM(315), N nm, F f, P p, DWELL(279) h,
T t
CYCLE FACE(81), A a, MMPR(316), M nm, F f, P p, DWELL(279) h,
T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 81 (FACE)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or

Parameter	CLD array		Description
			mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
DWELL	CLD[9]	CLD.Dwell	A keyword for a time of pause
h	CLD[10]	CLD.H	The pause of a tool in the sec
t	CLD[11]	CLD.Top	Top hole level

4.2.27 Drilling cycle a type G84

Command:

```

CYCLE TAP(168), A a, MMPM(315), N nm, F f, P p, T t
CYCLE TAP(168), A a, MMPR(316), M nm, F f, P p, T t

```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 168 (TAP)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		Reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.28 Drilling cycle a type G85

Command:

```

CYCLE BORE5(209), A a, MMPM(315), N nm, F f, P p, T t
CYCLE BORE5(209), A a, MMPR(316), M nm, F f, P p, T t

```

Parameters:

Parameter	CLD array	Description
-----------	-----------	-------------

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 209 (BORE5)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		Reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.29 Drilling cycle a type G86

Command:

```
CYCLE BORE6(210), A a, MMPM(315), N nm, F f, P p, T t
CYCLE BORE6(210), A a, MMPR(316), M nm, F f, P p, T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 210 (BORE6)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		Reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.30 Drilling cycle a type G87

Command:

```
CYCLE BORE7(211), A a, MMPM(315), N nm, F f, P p, T t
CYCLE BORE7(211), A a, MMPR(316), M nm, F f, P p, T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 211 (BORE7)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		Reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.31 Drilling cycle a type G88

Command:

```

CYCLE BORE8(212), A a, MMPM(315), N nm, F f, P p, DWELL(279)
h, T t
CYCLE BORE8(212), A a, MMPR(316), M nm, F f, P p, DWELL(279)
h, T t

```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 212 (BORE8)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
DWELL	CLD[9]	CLD.Dwell	A keyword for a time of pause
h	CLD[10]	CLD.H	The pause of a tool in the sec
t	CLD[11]	CLD.Top	Top hole level

4.2.32 Drilling cycle a type G89

Command:

```
CYCLE BORE9(213), A a, MMPM(315), N nm, F f, P p, DWELL(279)
h, T t
CYCLE BORE9(213), A a, MMPR(316), M nm, F f, P p, DWELL(279)
h, T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 213 (BORE9)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
	CLD[6] CLD[7]		Reserved
p	CLD[8]	CLD.P	The fly back level in the mm.
DWELL	CLD[9]	CLD.Dwell	A keyword for a time of pause
h	CLD[10]	CLD.H	The pause of a tool in the sec
t	CLD[11]	CLD.Top	Top hole level

4.2.33 Deep drilling with the full retraction of tool for removing a shaving

Command:

```
CYCLE DEEP(153), A a, MMPM(315), N nm, F f, L l, I i, P p, T t
CYCLE DEEP(153), A a, MMPR(316), M nm, F f, L l, I i, P p, T t
```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 153 (DEEP)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
l	CLD[6]	CLD.L	The depth of first drop in (in the mm)

Parameter	CLD array		Description
i	CLD[7]	CLD.I	The connecting value
p	CLD[8]	CLD.P	The fly back level in the mm.
	CLD[9] CLD[10]		Reserved
t	CLD[11]	CLD.Top	Top hole level

4.2.34 Deep drilling with drill retraction for removing the shaving

Command:

```

CYCLE BRKCHP(288), A a, MMPM(315), N nm, F f, L l, I i, P p,
DWELL(279) h, T t
CYCLE BRKCHP(288), A a, MMPR(316), M nm, F f, L l, I i, P p,
DWELL(279) h, T t

```

Parameters:

Parameter	CLD array		Description
Type	CLD[1]	CLD.Type	Cycle type 288 (BRKCHP)
a	CLD[2]	CLD.A	The depth of hole in the mm
MMPM, MMPR	CLD[3]	CLD. MMPM	Units of feedrate measurement (mm/min or mm/rot)
nm	CLD[4]	CLD.NM	The feedrate in the mm/mines or in the mm/rot
f	CLD[5]	CLD.F	The safe level in the mm
l	CLD[6]	CLD.L	The depth of first drop in (in the mm)
i	CLD[7]	CLD.I	The connecting value
p	CLD[8]	CLD.P	The fly back level in the mm.
DWELL	CLD[9]	CLD.Dwell	A keyword for a time of pause
h	CLD[10]	CLD.H	Tool pause in the sec
t	CLD[11]	CLD.Top	Top hole level

For creation of postprocessors similar ISO in masks it is possible to use preconceived values ISO.G, which one vary pursuant to parameters of a command.

CLD parameters value	ISO value
CLD[1] = 163	ISO.G = 81
CLD[1] = 81	ISO.G = 82
CLD[1] =168	ISO.G = 84
CLD[1] =209	ISO.G = 85
CLD[1] =210	ISO.G = 86
CLD[1] =211	ISO.G = 87
CLD[1] =212	ISO.G = 88

CLD parameters value	ISO value
CLD[1] =213	ISO.G = 89
CLD[1] =153	ISO.G = 83
CLD[1] =288	ISO.G = 73
CLD[1] =72	ISO.G = 80

4.2.35 Threading

Command:

```
THREAD MPR m, TPI t, MulTrd mt, Depth d, Cuts c, FinCut f,
Ang a, Oset1 o1, Oset2 o2
```

Parameters:

Parameter	CLD array		Description
m	CLD[1]	CLD. MPR	WMmpr(316)/WPitch(1050)
t	CLD[2]	CLD.TPI	Pitch of thread /number thread of a screw per inch
mt	CLD[3]	CLD. MulTrd	Thread number
d	CLD[4]	CLD. Depth	Depth
c	CLD[5]	CLD. Cuts	Cuts number (all)
f	CLD[6]	CLD. FinCut	Finish cuts number
a	CLD[7]	CLD. Ang	Cutting angle
o1, o2	CLD[8] CLD[9]	CLD. Oset1 CLD. Oset2	No compensation, switch up alternately in each pass

4.2.36 Palette changing

Command:

```
PALETA N n
```

Parameters:

Parameter	CLD array		Description
n	CLD[1]	CLD.N	Palette number

4.2.37 Head

Command:

```
HEAD BOTH(83) |N n
```

Parameters:

Parameter	CLD array		Description
n or BOTH	CLD[1]	CLD.N	n - the required head number or 83 (BOTH) – work at the

Parameter	CLD array	Description
		same time two head

4.2.38 Factors of a spline

Intermediate breakpoints of a NURBS-curve. These commands go after initializing a NURBS-curve (command **MovNRB**), the last point of a curve sets command **GOTO**.

Command:

COEFF X x, Y y, Z z, Knout k, Fedrat f, Spindl s, Denom d

Parameters:

Parameter	CLD array		Description
x,	CLD[1]	CLD.X	Control point coordinates
y,	CLD[2]	CLD.Y	
z	CLD[3]	CLD.Z	
k	CLD[4]	CLD.K	Knot NURBS curve
f	CLD[5]	CLD.F	Feed factor
s	CLD[6]	CLD.S	Spindle rotation velocity factor
d	CLD[7]	CLD.D	Denominator NURBS curve Factor

4.2.39 The beginning of a spline phase of trajectory

The beginning of a spline phase of trajectory.

Command:

MOVNRB Order

Parameters:

Parameter	CLD array		Description
Order	CLD[1]	CLD.Order	Order NURBS (level + 1)

4.2.40 Insertion

This command assigns specified string to CLDATA\$ variable.

Command:

INSERT "....."

The «CLDATA\$» variable:

"....."

4.2.41 Optional skipping

Command:

```
OPSKIP ON(71) | OFF(72)
```

Parameters:

Parameter	CLD array		Description
ON or OFF	CLD[1]	CLD.OnOff	71 (ON) switch on or 72 (OFF) switch off optional skipping

4.2.42 Postprocessor printing

The text is placed in CLDATA\$ variable whereupon it can be output in the display as a description postprocessor behavior comments.

Command:

```
PPRINT "....."
```

The «CLDATA\$» variable:

```
"....."
```