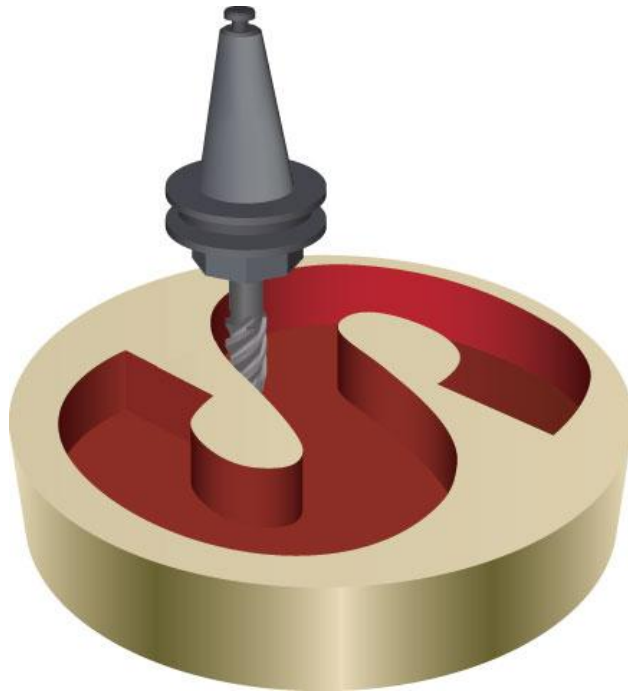


SolidCAM



SolidCAM GPPTool User Guide

©1995-2005 SolidCAM LTD.
All Rights reserved

www.cadfamily.com EMail:cadserv21@hotmail.com

The document is for study only,if tort to your rights,please inform us,we will delete

TABLE OF CONTENTS

Chapter 1. Introduction	1-1
1.1 Introduction.....	1-1
1.2 Types of Post-processors	1-1
1.3 About this manual.....	1-2
Chapter 2. Installation and Running of GPPtool.....	2-1
2.1 GPPtool Installation.....	2-1
2.2 Running GPPtool	2-1
Chapter 3. Mac Pre-Processor Parameters.....	3-1
3.1 Machine Specifications.....	3-1
3.1.1 Internal ParmS	3-1
3.1.2 Machine Initialize	3-3
3.1.3 Program numbers.....	3-7
3.1.4 Procedures control	3-8
3.1.5 Home.....	3-21
3.1.6 Positioning	3-22
3.1.7 Compensation	3-22
3.1.8 Line Definitions	3-23
3.1.9 Arc Definitions	3-25
3.1.10 Epsilon Values	3-30
3.1.11 Feed-Spin.....	3-31
3.1.12 Timing.....	3-31
3.1.13 Part Options	3-32
3.1.14 Job Options	3-32
3.1.15 Drill Cycles.....	3-32
3.1.16 Turning Cycles.....	3-33
3.1.17 Threading Cycles	3-33
3.1.18 Grooving Cycles	3-33
3.1.19 Turning Definitions.....	3-34
3.1.20 Fourth axis	3-35
3.1.21 Five axis.....	3-36
3.1.22 Wire Cut Cycles.....	3-41
3.1.23 Wire Cut Parameters	3-41
3.2. MacEdit Utility	3-43
3.2.1. MacEdit Installation.....	3-44
3.2.2. Running MacEdit.....	3-44
3.3 Examples of [machine.mac] files.....	3-45
3.3.1 FANUC.mac	3-45
3.3.2 MAHO.mac.....	3-50

Chapter 4. Internal Fast Post-Processor.....	4-1
4.1. Internal Fast Post-Processor.....	4-1
4.1.1 Introduction.....	4-1
4.1.2. Start Program	4-1
4.1.3. CAM Settings	4-2
4.2. User Documentation	4-3
4.2.1. Mac Doc-Processor.....	4-3
4.2.2. SOLIDCAM User Documentation Commands	4-3
4.2.3. Example of User Documentation.....	4-4
Chapter 5. GPPL language	5-1
5.1 Introduction.....	5-1
5.2 GPPL Variables	5-2
5.2.1 Variable definition	5-2
5.2.2 Variable types	5-2
5.2.3 Variable attributes.....	5-2
5.2.4 Variables groups	5-3
5.2.5 Array	5-4
5.3 GPPL Constants	5-5
5.4 GPPL Expressions	5-7
5.4.1 Expression types	5-7
5.4.2 Operators.....	5-8
5.4.3 Operators precedence.....	5-9
5.4.4 Conversions	5-9
5.5 GPPL statements.....	5-10
5.5.1 Global declaration.....	5-10
5.5.2 Local declaration.....	5-11
5.5.3 Assignment statements.....	5-12
5.5.4 Attribute assignment statements	5-13
5.5.5 Conditional statements.....	5-14
5.5.6 Subroutine Calls.....	5-15
5.5.7 Procedure Calls	5-16
5.5.8 Procudure definition	5-17
5.5.9 Comment.....	5-18
5.5.10 G-code generation.....	5-19
5.5.10.1 Conditional generation (modality).....	5-20
5.5.10.2 Display format	5-22
5.5.11 While Statement.....	5-25
5.5.12 Trace Statement	5-26
5.6 GPPL Internal functions	5-27

5.6.1 Numeric functions.....	5-27
5.6.2 String functions.....	5-38
5.6.3 Logical functions	5-45
Chapter 6. GPPtool System variables	6-1
6.1 Variables that are defined in SolidCAM and passed to GPPtool	6-1
6.2 Variables that are defined at the start of the [machine.gpp] file.....	6-3
6.3 Variables which have special values	6-6
Chapter 7. GPPtool commands	7-1
7.1 SolidCAM Tool-Path commands.....	7-1
7.2 User-Defined Commands.....	7-98
APPENDIX A:	
A.1. Example: User-defined Pre- and Post-processor for a FANUC controller	A-1
A.2. Example 2: User-defined Pre and Post-processor for mill-Turn with XZYCB axis controller.....	A-15
APPENDIX B: Example: FANUC G-Code file.....	B-1
APPENDIX C: GPPtool error messages.....	C-1

CHAPTER 1

1.1 Introduction

GPPtool is a General Pre- and Post-Processor tool that enables you to accomplish the following:

1. Define the Pre-processor parameters that affect the tool-path generation in **SolidCAM**.
2. Define the Post-processor parameters and the GPPL (General Post-processor language) procedures that define how the **SolidCAM** tool-path commands develop into G-Code for the particular CNC controller.

These capabilities provide you with the following:

1. The ability to change an existing pre- and post-processor for a CNC machine that is already supported by **SolidCAM**. This is done either to make small changes in the G-Code format for example or to customize the G-Code produced by the system in order to fit a certain template that the user is comfortable with.
2. The ability to develop a new pre- and post-processor to support a CNC machine that is not yet supported in **SolidCAM**.

1.2 Types of Post-processors

Three types of Post-processors are provided by GPPtool:

(a) User-defined Post-processor

For each particular CNC-machine controller, two files are needed as input to GPPtool. These files are:

- (1) machine.mac
- (2) machine.gpp

where [machine] is any name chosen by the user for the particular CNC machine controller.

The file [machine.mac] defines the Pre-processor parameters that affect the tool-path generation in **SolidCAM**.

The file [machine.gpp] defines the Post-processor parameters and the GPPL procedures that define how the **SolidCAM** tool-path commands develop into G-Code for the particular CNC controller.

Both these files can be generated and edited using any text-editor at the disposal of the user.

(b) Internal Fast Post-processor

In the case of an Internal post-processor only the file [machine.mac] is under control of the user; The [machine.gpp] functionality is hard-coded in SolidCAM and cannot be changed by the user. The advantage of the internal post-processor is that it generates G-Code faster than the user-defined post-processor.

(c) User-defined Doc-processor

For each particular CNC-machine controller, two files are needed as input to GPPTool. These files are:

- (1) machine.mac
- (2) machine.dpp

where [machine] is any name chosen by the user for the particular CNC machine controller.

The file [machine.mac] defines the Doc-processor parameters that affect the Documentation output in **SOLIDCAM**.

The file [machine.dpp] defines the Doc-processor parameters and the GPPL procedures that define the documentation output in **SolidCAM** for the particular CNC controller.

Both these files can be generated and edited using any text-editor at the disposal of the user.

1.3 About this manual

The GPPTOOL manual is organized in seven chapters and three appendices:

Chapter 1 gives an introduction to GPPTool.

Chapter 2 describes the installation and running procedures.

Chapter 3 describes the Pre-processor parameters that affect the tool-path generation in **SolidCAM** and that are included in the [machine.mac] file.

Chapter 4 describes the use of internal Fast Post-processor and User documentation.

Chapter 5 describes the GPPL (General Post-Processor language).

Chapter 6 describes the GPPtool system variables.

Chapter 7 describes the GPPtool commands.

Appendix A gives a listing of the [fanuc.mac] and [fanuc.gpp] files for a user-defined Pre and Post processor for a Fanuc controller.

Appendix B gives an example of a G-Code file produced by GPPtool for a Fanuc controller.

Appendix C gives a listing of the error messages produced by GPPtool.

CHAPTER 2

2. Installation and Running of GPPtool

This chapter describes the installation and running procedures of the GPPtool program.

2.1. GPPtool Installation

GPPtool is installed on the computer during the installation of the **SolidCAM** program.

2.2. Running GPPtool

The GPPtool program can be run in one of two modes:

1. As an integral part of **SolidCAM** program. Whenever the G-Code command is given inside **SolidCAM** the GPPtool program is activated to generate the G-Code.
2. As a separate stand-alone program; It receives as input a Part generated by **SolidCAM** and then generates the G-Code. This program supports list of postprocessors. The user should make sure that the same CNC-machine is used both by **SolidCAM** and **GPPtool**. Run **GPPtool** by Double click on file named GPPToolExe.exe that is located on the following folder:

\Program Files\SolidCAM2005\SolidCAM

CHAPTER 3

3. MAC (Pre-Processor Parameters)

These are the parameters that are defined by the user and that affect the tool-path generation in **SolidCAM**. They are defined in the file:

machine.mac

where [machine] is any name chosen by the user for the particular CNC machine controller.

The file [machine.mac] can be modified using the MacEdit editor Utility.

The parameters' types can be one of the following type: integer, numeric, logical and string.

- ◆ integer - can hold signed, integer numbers in the range of (-999999999, +999999999).
- ◆ numeric - can hold any number in the range of (-1.E300, +1.E300).
- ◆ logical - can hold the logical values TRUE (1) or FALSE (0).
- ◆ string - can hold any sequence of ASCII characters; the number of characters is unlimited.

3.1. Machine specifications

3.1.1. Internal ParmS

3.1.1.1. machine_type type: integer (enumerator)

{MILLING, WIRE_CUT, TURNING, MILL_&_TURN}

Defines the type of machining.

3.1.1.2. post_processor type: string { *GPP-filename , GPP-filename }

Tells SolidCAM whether to generate G-Code using the internal fast post processor (postprocessor name proceed with '*'), or whether to generate it using the [machine.GPP]

file. Note that it is sufficient to give the name of the machine; the GPP extension is automatically added.

3.1.1.3. doc_processor type: string { DPP-filename }

Define the file name of the DPP file using to customize the Documentation output of **SolidCAM**.

Note that it is sufficient to give the name of the machine; the DPP extension is automatically added.

3.1.1.4. gpp_file_ext type: string

Define the extension of the G_code file name that will be created.

3.1.1.5. mac_holder type: string

Define the name of the upper constant part of the holder. This part of the holder belongs to the machine. When you choose the tool holder option in the Tool menu of the CAM-Part, and the mac_holder name defined in the mac file exists in the holders.tnt table, the holder will join the user holders.

3.1.1.6. tool_table_name type: string

Define the tool table name of the machine to be the part tool table. Note that it is sufficient to give the name of the tool table; the TAB extension is automatically added.

3.1.1.7. max_g_name_length type: integer

Defines the maximum number of characters in the G-Code file name. The name of the file is created from the first Max_g_name_length of the part name.

If there is a split, the name will be built from the first characters of the part and the number of the split.

All the spaces are deleted from the part name.

Example:

Part name: upper plate side 1

Extension: min

Max_g_name_length = 8

Gcode file name: upperpla.min

Split 1: upperpl-1.min

Split 10: upperp-10.min

3.1.1.8. max_tool_numbers type: integer

Defines the maximum number of tools on the machine table.

3.1.1.9. default_lang type: integer

Define the language of the postprocessor from among the following options: CHINESE, CZECH, DANISH, ENGLISH, FRENCH, GERMAN, HEBREW, ITALIAN, JAPANESE, KOREAN, POLISH, PORTUGUESE, RUSSIAN, SPANISH and TURKISH.
The postprocessor cycle's parameters can be translated to other languages using the MacEdit Utility.

3.1.2. Machine Initialize**3.1.2.1. machine_plane** type: integer (enumerated) {XY, YZ, ZX}

Defines the default machine-plane for home definition. It can be changed in the home definition dialog for current home.

3.1.2.2. z_with_xy type: logical {Y/N}

Specifies if a linear movement in the Z axis together with movement in the X or Y axis is possible in the machine.

If No, the Jobs Translated Surface, Slot with section, 3D Milling and 3D Engraving can not be used.

3.1.2.3. mac_axes type: integer {XYZ, XZC, XYZC}

Defines in which axes the machine can work. If the parameter 'num_axes' is 4, 'mac_axes' will be considered as XYZC. GCode for simultaneous 4th axis can be generated only for XYZC or XZC machines. This parameter is not relevant for turning machines without milling capabilities.

3.1.2.4. num_axes type: integer

Number of machine axes. Max value is 5.

3.1.2.5. num_simult_axes type: integer

Number of allowed simultaneous machine axes movement.

3.1.2.6. abs_coord type: logical {Y/N}

Defines the default starting mode for the CNC machine;

- ◆ If Yes, then the default starting condition of the CNC machine is absolute coordinates;
- ◆ If No, then the default starting condition of the CNC machine is relative coordinates.

3.1.2.10. Loops

type: logical

{Y/N}

Defines if the CNC-machine has a G-code command for loops. If Yes, it generates G-code loop commands to reduce the amount of G-code. It effect only the job Slot with section..

Example:**Loops = Y**

-----Main program-----

O5000 (.TAP)

M01

G43 H1 D31 G0 X6.495 Y41.373 Z50. S1000 M3

M8

O1

(------)

(S-SLOT-T1 - SLOT)

(------)

G0 X6.495 Y41.373 Z10.

G91 #23 = 0

WHILE [#23 LT 11] DO 3

G0 Z-8.

G1 Z-23.009 F50

Y-30.316 F100

Y-0.661 Z0.25

Y-0.331 Z0.625

Y-0.008 Z0.125

Z13.156

Y-1.75 Z4.63

Y-4.375 Z2.315

Y-0.875 Z0.055

G0 Z9.853

X1. Y38.316

#23 = #23 + 1

G1

END 3

G90 M99

Loops = N

-----Main program-----

```
O5000 (.TAP)
M01
G43 H1 D31 G0 X6.495 Y41.373 Z50. S1000 M3
M8
O1
(-----)
(S-SLOT-T1 - SLOT)
(-----)
G0 X6.495 Y41.373 Z10.
  Z2.
G1 Z-21.009 F50
  Y11.056 F100
  Y10.395 Z-20.759
  Y10.064 Z-20.134
  Y10.056 Z-20.009
  Z-6.853
  Y8.306 Z-2.223
  Y3.931 Z0.092
G0 Z10.
  X7.495 Y41.373
  Z2.
G1 Z-21.009 50
  Y11.056 F100
  Y10.395 Z-20.759
  Y10.064 Z-20.134
  Y10.056 Z-20.009
  Z-6.853
  Y8.306 Z-2.223
  Y3.931 Z0.092
G0 Z10.
M99
```

3.1.2.11. _4th_axes_around type: logical {Y/N}

When the parameter num_axes has a value of 4 (4-axis CNC machine), the new parameter _4th_axes_around defines the constant axes for the machine. There are 2 possibilities:

Possibility #1

If
 num_axes = 4
 and
_4th_axes_around = X

This means that the fourth axis is along the X-axis of the machine (vertical machine) and the modul will be rotated around the X-axis of the screen.

Possibility #2

If
 num_axes = 4
 and
_4th_axes_around = Y

This means that the fourth axis is along the Y-axis of the machine (horizontal machine) and the modul will be rotated around the Y-axis of the screen.

Note:

It is not possible to change the part generated with post include `_4th_axes_around = Y` to `_4th_axes_around = X`. The reason is that the homes are saved relatively to no of axis definition.

3.1.2.12. `_5th_axes_around` type: logical {Y/N}

When the parameter `num_axes` has a value of 5 (5-axis CNC machine), the new parameter `_5th_axes_around` defines the rotate axes for the machine. There are 3 possibilities:

Possibility #1

If
`num_axes = 5`
and
`_5th_axes_around = X`

The modul will be rotated around the X-axis of the screen.

Possibility #2

If
`num_axes = 5`
and
`_5th_axes_around = Y`

The modul will be rotated around the Y-axis of the screen.

Possibility #3

If
`num_axes = 5`
and
`_5th_axes_around = Z`

The modul will be rotated around the Z-axis of the screen.

3.1.3. Program numbers**3.1.3.1. `prog_num_min`** type: integer

Minimum allowed program number.

3.1.3.2. `prog_num_max` type: integer

Maximum allowed program number.

3.1.3.3. `prog_num_dflt` type: integer

Default program number {`prog_num_min <.. < prog_num_max`}.

3.1.3.4. get_prog_num type: logical {Y/N}

This parameter decides whether the user is asked to give the program number in the Part dialog.

3.1.3.5. proc_num_min type: integer

Minimum allowed procedure number.

3.1.3.6. proc_num_max type: integer

Maximum allowed procedure number.

3.1.3.7. proc_num_dflt type: integer

Default first procedure number {proc_num_min <..<> proc_num_max}.

3.1.3.8. get_proc_num type: integer

This parameter decides whether the user is asked to give the first procedure number in the Part dialog.

3.1.4. Procedures control

3.1.4.1. full_gcode type: logical {Y/N}

This parameter decides whether the (X, Y, Z) coordinates are modal or not. If they are not, this means that in every line of the G-code, all 3 coordinates must be written even though they might not have changed from the previous G-code line.

3.1.4.2. gen_procs type: integer (enumerator) {Y,N,C, D}

Indicates whether procedures can be generated in the G-code.

- ◆ If Yes, Procedures will be generated (in the main program level). Each job will be generated as procedure that will be called from the main program.
- ◆ If No, Procedures will not be generated (in the main program level).
- ◆ If C, Procedures will be generated only if they do not include further procedure calls.
- ◆ If D, This procedure-generation gives the following results:
 1. There is only one level of sub-programs.
 2. @call_proc is changed by @call_prms (subroutine call with parameters).

3. A drill sub_program is generated for one drill or more (in the other modes a sub-program is generated only if there is more than one drill-hole) to get the same structure of sub-programs.
4. In the movement sub-program, all the necessary G0 positioning are entered and all the movements in the Z direction are incremental to be able to call this sub-program, with parameters, as many times as needed.
5. The numbers of calls in the main program depend on the depth and the down step.

For example:

- If the depth is 10 and down step 2, the result is:

```
G65 P1000 L5
```

- If the depth is 10 and down step 3, the result is:

```
G65 P1000 L3
G65 P1000 L1
```

6. Before the call to the sub-program, you have to create a movement to the relevant absolute Z- position from which the incremental movement in the sub-program will start.

Example:

```
gen_procs = Y
gen_internal_proc = Y
```

```
-----Main program-----
O5000 (.TAP)
M01
N1 M6 T1
G43 H1 D31 G0 X8. Y20. Z50. S3000 M3
M8
M98 P1 (F-profile-T1)
M30

O1
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X8. Y20. Z10.
Z2.
G1 Z-5. F33
M98 P3
G0 Z10.
Y20.
Z-3.
G1 Z-10. F33
```

M98 P3
G0 Z10.
M99

O3
G1 X8. Y125. F500
M99

gen_procs = N
gen_internal_proc = N

-----Main program-----
O5000 (.TAP)
M01
N1 M6 T1
G43 H1 D31 G0 X8. Y20. Z50. S3000 M3
M8
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
X8. Y20. Z10.
Z2.
G1 Z-5. F33
Y125. F500
G0 Z10.
Y20.
Z-3.
G1 Z-10. F33
Y125. F500
G0 Z10.
M98 P9010
M30

gen_procs = C
gen_internal_proc = Y

-----Main program-----
O5000 (.TAP)
G54
M01
N1 M6 T1
G43 H1 D31 G0 X8. Y20. Z50. S3000 M3
M8
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
X8. Y20. Z10.
Z2.
G1 Z-5. F33
M98 P3
G0 Z10.

```

Y20.
Z-3.
G1 Z-10. F33
M98 P3
G0 Z10.

M30

O3
G1 X8. Y125. F500
M99

```

```

gen_procs = D
gen_internal_proc = Y

```

```

-----Main program-----
O05000 (.MZK)
G43 H1 D1 Z50. M8
G65 P1 Z-5. I0. F500 E33 (F-profile-T1); @call_prms
M30

O01
G0 X8. Y20.
Z#4
G1 Z#26 F#8
G1 Y125. F#9
G0 Z50.
M99

```

3.1.4.3. drill_proc type: logical {Y/N}

If several drill jobs have the same drill-points and this parameter is Yes, then the drill-points for all the drill-jobs are put in a single separate procedure. If No, then the drill-points are generated in every job.

Example:

```

drill_proc = Y

O1
(-----)
(D-D-T3 - DRILL)
(-----)
G0 X17. Y50. Z50.
G98 G81 Z2. R5. F100
M98 P3 (d)
M99
O2
(-----)

```

```
(D-D-T4 - DRILL)
(-----)
G0 X17. Y50. Z50.
G98 G84 Z-1.8 F50
M98 P3 (d)
M99
O3
(-----)
(D - DRILLS)
(-----)
  X27.
  X17. Y20.
  X27.
  X77.
  X87.
  Y50.
  X77.
G80
M99
```

drill_proc = N

```
O1
(-----)
(D-D-T3 - DRILL)
(-----)
G0 X17. Y50. Z50.
G98 G81 Z2. R5. F33
  X27.
  X17. Y20.
  X27.
  X77.
  X87.
  Y50.
  X77.
G80
M99
O2
(-----)
(D-D-T4 - DRILL)
(-----)
G0 X17. Y50. Z50.
G98 G81 Z-2. R5. F33
  X27.
  X17. Y20.
  X27.
  X77.
  X87.
  Y50.
  X77.
G80
M99
```

3.1.4.4. turn_proc type: logical {Y/N}

This parameter determines whether **SolidCAM** will generate a separate procedure for the geometric points of a turning process ('Y'), or whether the points will be generated immediately after the cycle.

3.1.4.5. thread_proc

This parameter determines whether **SolidCAM** will generate a separate procedure for the geometric points of a threading process ('Y'), or whether the points will be generated immediately after the cycle.

3.1.4.6. gen_internal_proc type: logical {Y/N}

This parameter indicates whether internal procedures of the job will be generated.

- ◆ If Yes, internal procedures will be generated.
- ◆ If No, internal procedures will not be generated.

Note:

In order to obtain G-code that contains as much as possible calls to procedures, give 'Y' to the parameters 'gen_procs' and 'gen_internal_procs'. In order to obtain G-code that has no calls to procedure, give 'N' to both of these parameters.

3.1.4.7. turn_common_proc type: logical {Y/N}

This parameter determines whether or not to generate a common geometric points procedure for several cycles.

3.1.4.8. gen_1_line_proc type: logical {Y/N}

This parameter is used for turning cycles. It specifies the G-code format if the cycle geometry is a single line. Usually the cycle geometry is given in a G-code procedure.

- ◆ If Yes, a geometry procedure will be generated as any other geometry.
- ◆ If No, the geometry procedure will not be generated, and the geometry details will be given only in the turning command.

3.1.4.9. optimize_jobs_loop type : logical {Y/N}

- ◆ If Yes, **SolidCAM** saves tool pathes among continuous jobs that have the same Edit operations.

Example:

optimize_jobs_loop = Y

O5000 (.TAP)

G54

M98 P9011

M01

M98 P1 (F-profile1-T1)

M98 P2 (D-drill1-T1)

G10G91 L2 P1 X60. Y0. Z0.

G90

#22 = #22 + 1

G1

END 2

G10G91 L2 P1 X-120. Y-30. Z0.

G90

#21 = #21 + 1

G1

END 1

G10G91 L2 P1 X0. Y60. Z0.

G90

M98 P9010

M30

O1

optimize_jobs_loop = N

O5000 (.TAP)

G54

M98 P9011

M01

M98 P1 (F-profile1-T1)

G10G91 L2 P1 X60. Y0. Z0.

G90

#22 = #22 + 1

G1

END 2

G10G91 L2 P1 X-120. Y-30. Z0.

G90

#21 = #21 + 1

G1

END 1

G10G91 L2 P1 X0. Y60. Z0.

G90

#21 = 0

WHILE [#21 LT 2] DO 1

#22 = 0

WHILE [#22 LT 2] DO 2

M98 P2 (D-drill1-T1)

```

G10G91 L2 P1 X60. Y0. Z0.
G90
#22 = #22 + 1
G1
END 2
G10G91 L2 P1 X-120. Y-30. Z0.
G90
#21 = #21 + 1
G1
END 1
G10G91 L2 P1 X0. Y60. Z0.
G90
M98 P9010
M30
O1

```

3.1.4.10. seq_sub_number type: logical {Y/N}

Determines whether **SolidCAM** will renumber the output G-Code so that there will be no gaps in the sub-program numbering. Add the following marks in the output G_code:

- {nb,'//G65P' } line before
 {nb,'G65P'label }.
- {nb,'??O' } line before
 {nl,'O'label }

Note:

The syntax of the mark should be printed exactly like the printed line (no extra spaces).

Example:

```
seq_sub_number =- Y
```

```
O5000 (.TAP )
M98 P8060
```

```
G56 P122 Z-5. A5000. F5000. H0.(F-profile-T4)
```

```
G56 P123 Z-30. A33. F100. H0.(F-profile-T5)
```

```
G56 P124 Z-5. A33. F100. H0.(P-profile1-T5)
```

```
G56 P125 Z0. A33. F100. H0.(S-slot-T7)
```

```
G65 P126 J81. Z-10. R3. F100.(D-drill-T1)
```

M98 P8060

M30
O0122

M99
O0123

M99
O0124

M99
O0125

M99
O0126

M99
O0127
(- CLOSE PROGRAM -)
M30
%

seq_sub_number = N

O5000 (.TAP)
M98 P8060

G56 P111 Z-5. A5000. F5000. H0.(F-profile-T4)

G56 P112 Z-30. A33. F100. H0.(F-profile-T5)

G56 P113 Z-5. A33. F100. H0.(P-profile1-T5)

G56 P114 Z0. A33. F100. H0.(S-slot-T7)

G65 P109 J81. Z-10. R3. F100.(D-drill-T1)

M98 P8060
M30
O0111

M99
O0112

M99
O0113

M99
O0114

M99
O0109

M99
O0110
(- CLOSE PROGRAM -)
M30

3.1.4.11. Loop_exist type: logical {Y/N}

If loop_exist = 'N', the pcode @loop has not been generated in the case of transform. Instead, pcode @change_ref_point will be generated.

In the case of rotate, the pcode of @rotate will be generated instead of pcode @loop.

Example:

Loop_exist = Y

%
O5000 (.TAP)

G54
M98 P9011
M01

(TOOL -1- MILL DIA 6.0 R0. MM)
G90 G00 G40 G54
G43 H1 D31 G0 X7. Y12.5 Z50. S1000 M3
M8

#21 = 0
WHILE [#21 LT 2] DO 1
#22 = 0
WHILE [#22 LT 2] DO 2
M98 P1 (F-profile-T1)
M98 P2 (D-drill-T1)

G90
#22 = #22 + 1
G1
END 2

G90
#21 = #21 + 1
G1
END 1

G90
M98 P9010
M30
O1
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
M99
O2

(-----)
(D-DRILL-T1 - DRILL)
(-----)
G0 X15. Y15. Z10.
G98 G81 Z-5. R2. F33
M98 P3 (drill)
M99
O3
(-----)
(DRILL - DRILLS)
(-----)
X25.
G80
M99
%

Loop_exist = N

%
O5000 (.TAP)

G54
M98 P9011
M01

(TOOL -1- MILL DIA 6.0 R0. MM)
G90 G00 G40 G54
G43 H1 D31 G0 X7. Y12.5 Z50. S1000 M3
M8
M98 P1 (F-profile-T1)
M98 P2 (D-drill-T1)
G10G91 L2 P1 X60. Y0. Z0.
G90
M98 P1 (F-profile-T1)
M98 P2 (D-drill-T1)
G10G91 L2 P1 X60. Y0. Z0.
G90
G10G91 L2 P1 X-120. Y30. Z0.
G90
M98 P1 (F-profile-T1)
M98 P2 (D-drill-T1)

```

G10G91 L2 P1 X60. Y0. Z0.
G90
M98 P1 (F-profile-T1)
M98 P2 (D-drill-T1)
G10G91 L2 P1 X60. Y0. Z0.
G90
G10G91 L2 P1 X-120. Y30. Z0.
G90
G10G91 L2 P1 X0. Y-60. Z0.
G90
M98 P9010
M30
O1
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)

M99
O2
(-----)
(D-DRILL-T1 - DRILL)
(-----)

M98 P3 (drill)
M99
O3
(-----)
(DRILL - DRILLS)
(-----)
  X25.
G80
M99
%
```

3.1.4.12. same_sub_numbers type: logical {Y/N}

If same_sub_numbers = “Y”, all G-code programs that are created in a part with splits have subroutines that start with the same number of the first block.

Note:

In order this parameter will be activated the parameter seq_sub_number must be set to ‘Y’.

Example:

```

same_sub_numbers = Y
seq_sub_number = Y
```

```

O5000 (.TAP )
M98 P8060
```

G56 **P109** Z-5. A5000. F5000. H0.(F-profile-T4)

M98 P8060

M30

O0109

M99

O0110

(- CLOSE PROGRAM -)

M30

%

----- split -----

%

O5001 (.TAP)

M98 P8060

G56 **P109** Z-30. A33. F100. H0.(F-profile-T5)

M98 P8060

M30

O0109

M99

O0110

(- CLOSE PROGRAM -)

M30

%

same_sub_numbers = N

seq_sub_number = Y

%

O5000 (.TAP)

M98 P8060

G56 **P109** Z-5. A5000. F5000. H0.(F-profile-T4)

M98 P8060

M30

O0109

M99

O0110

(- CLOSE PROGRAM -)

M30

%

----- split -----

%

O5001 (.TAP)

M98 P8060

G56 P111 Z-30. A33. F100. H0.(F-profile-T5)

M98 P8060

M30

O0111

M99

O0112

(- CLOSE PROGRAM -)

M30

%

3.1.4.13. init_var_after_split type: logical {Y/N}

All gpp variables will be initialized in every gcode program in part with splits.

3.1.5. Home

3.1.5.1. num_homes type: integer

Number of homes available in the CNC machine.

3.1.5.2. dflt_home type: integer {1 .. num_homes}

The number of the default homes of the CNC machine.

3.1.5.3. abs_zero_chng type: logical {Y/N}

Method for offsetting the Zero-Point (used in '@change-ref-point').

- ◆ If Yes, the Zero-Point is offset relative to the previous Zero- Point.
- ◆ If No, it is offset relative to the current position.

3.1.5.4. home_data_at_start type: logical {Y/N}

Defines the location where the @home_data will be printed either at the beginning of the program after @def_tool or in the end of the program after @end_of_program. The @home_data includes the information of the new home position, rotation of the home etc...

3.1.6. Positioning

3.1.6.1. **dflt_start** type: numeric (x, y, z)

This is the default starting position of the tool at the start of the manufacturing of the Part. This value can be changed inside **SolidCAM** through the Part Options field.

3.1.6.2. **dflt_end** type: numeric (x, y, z)

This is the default end position of the tool at the end of the manufacturing of the Part. This value can be changed inside **SolidCAM** through the Part Options field.

3.1.6.3. **set_z_chng** type: logical {Y/N}

This parameter decides how the (Z) value changes during the tool-change.

- ◆ If Yes, the tool change occurs at the (Z) position defined by the **dflt_tool_chng** point.
- ◆ If No, then the (Z) position of the tool-change is decided by the controller.

This flag can be changed inside **SolidCAM** through the Part Options field.

3.1.6.4. **dflt_tool_chng** type : numeric (x, y, z)

Default tool change place.

3.1.7. Compensation

3.1.7.1. **comp_exist** type: logical {Y/N}

- ◆ Tells SolidCAM whether there is a tool-radius compensation ability in the CNC machine.
- ◆ If No, This option is disabled in the Job dialog.

Note:

Even if the CNC machine has a tool-radius compensation capability, the user can decide not to use this machine feature in case he wants to generate G-Code that does not use the tool-compensation feature of the machine.

3.1.7.2. **comp_arc_arc** type: logical {Y/N}

Tells **SolidCAM** whether the machine controller can activate tool-radius compensation between two connected arcs.

3.1.7.3. comp_arc_line type: logical {Y/N}

Tells **SolidCAM** whether the machine controller can activate tool-radius compensation between a connected line and arc.

3.1.7.4. comp_line_line type: logical {Y/N}

Tells **SolidCAM** whether the machine controller can activate tool-radius compensation between two connected lines.

3.1.7.5. next_angle type: logical {Y/N}

If Yes, the next block direction angle is added to blocks which are not followed by an XY block.

3.1.7.6. comp_x_start type: logical {Y/N}

Specifies how the tool is positioned at the start of the first block, after the tool-radius compensation is activated.

3.1.7.7. comp_by_arcs type: logical {Y/N}

- ◆ If Yes, an arc is generated when the machine controller activates tool-radius compensation at the intersection of two lines.
- ◆ If No, a line is generated when the machine controller activates compensation at the intersection of two lines.

This option is effect only on Simulation tool-path.

3.1.7.8. Delta_for_TOOL_H type: integer

The delta address of the offset table in the machine controller. Used for the Documentation option only.

3.1.7.9. comp_by_zero_tool type: logical {Y/N}

Tells **SolidCAM** whether the tool-radius compensation is zero (if Yes) or tool radius (if No).

3.1.8. Line Definitions

3.1.8.1. divide_to_segments type: logical {Y/N}

Tells **SolidCAM** whether or not to divide line to segments.

Example:

divide_to_segments = Y
segment_length = 0.5

%
O5000 (.TAP)

M98 P9011
M01

M98 P101 (F-profile-T1)
M98 P9010
M30
O101
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X-3.002 Y-2. Z10.
Z2.

G1 Z1.5 F33
Z1.
Z0.5
Z0.
Z-0.5
Z-1.

G41 G1 Y-1.5
Y-1.
Y-0.5
Y0.
Y0.5
Y1.
Y1.5
Y2.
Y2.5
Y3.
Y3.5
Y4.
Y4.5
Y5.
Y5.5
Y6.
Y6.5
Y7.
Y7.5
Y8.
Y8.5
Y9.
Y9.5
Y10.
Y10.5
Y11.

```
G40 G1 X-3.002
GO Z10.
M99
%
```

divide_to_segments = N

```
%
O5000 (.TAP)
```

```
G54
M98 P9011
M01
```

```
M98 P101 (F-profile-T1)
M98 P9010
M30
O101
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
GO X-3.002 Y-2. Z10.
Z2.
G1 Z-1. F100
G41 G1 Y11.
G40 G1 X-3.002
GO Z10.
M99
%
```

3.1.8.2. segment_length type: numeric

This parameter defines the segment length (when line is divided into linear segments).

3.1.9. Arc Definitions

3.1.9.1. arc_exist type: logical {Y/N}

Tells **SolidCAM** whether or not the CNC machine controller has an arc command. If it doesn't, the arc is divided into linear segments.

3.1.9.2. arc_3d type: logical {Y/N}

Tells **SolidCAM** whether or not the CNC machine has helical arcs. If it doesn't, a helical arc is divided into linear segments.

Note:

Helical Arcs are generated when there is a limit on a curved surface.

Example:

arc_3d = Y

```
%
O5000 (.TAP)

G54
M98 P9011
M01
M98 P1 (P- POCKET -T1)

M98 P9010
M30
O1
(-----)
(P- POCKET -T1 - POCKET)
(-----)
G0 X182.414 Y32.5 Z10.
  Z2.
G3 X165.856 Y40.05 Z0. R10. F100
X171.015 Y22.598 Z-2. R10.
X173.813 Y42.402 R-10. F300
  X171.015 Y22.598 R-10.
G1 X172.414 Y32.5
  X177.5
  Y51.538
  X174.919 Y54.119
G2 X172.414 Y32.5 R38.

G0 Z10.
M99
%
```

arc_3d = N

```
%
O5000 (.TAP)

G54
M98 P9011
M01

M98 P1 (P- POCKET -T1)
M98 P9010
M30
```

```

O1
(-----)
(P- POCKET -T1 - POCKET)
(-----)
G0 X182.414 Y32.5 Z10.
  Z2.
G1 X182.271 Y34.185 Z1.852 F100
  X181.846 Y35.822 Z1.704
  X181.151 Y37.364 Z1.556
  X180.207 Y38.767 Z1.407
  X179.039 Y39.991 Z1.259
  X177.682 Y41. Z1.111
  X176.174 Y41.766 Z0.963
  X174.559 Y42.267 Z0.815
  X172.882 Y42.489 Z0.667
  X171.192 Y42.425 Z0.519
  X169.537 Y42.077 Z0.37
  X167.964 Y41.455 Z0.222
  X166.518 Y40.577 Z0.074
  X165.241 Y39.468 Z-0.074
  X164.17 Y38.159 Z-0.222
  X163.334 Y36.689 Z-0.37
  X162.758 Y35.099 Z-0.519
  X162.458 Y33.434 Z-0.667
  X162.443 Y31.743 Z-0.815
  X162.713 Y30.074 Z-0.963
  X163.261 Y28.473 Z-1.111
  X164.07 Y26.988 Z-1.259
  X165.118 Y25.661 Z-1.407
  X166.375 Y24.529 Z-1.556
  X167.805 Y23.626 Z-1.704
  X169.366 Y22.976 Z-1.852
  X171.015 Y22.598 Z-2.
G3 X173.813 Y42.402 R-10. F300
  X171.015 Y22.598 R-10.
G1 X172.414 Y32.5
  X177.5
  Y51.538
  X174.919 Y54.119
G2 X172.414 Y32.5 R38.

G0 Z10.
M99
%
```

3.1.9.3. arc_3d_4x type: logical {Y/N}

Tells **SolidCAM** whether or not the CNC machine has arcs with deferent depth on the 4 axis. If it doesn't, an arc is divided into linear segments

3.1.9.4. arc_quadrants type: logical {Y/N}

Tells **SolidCAM** whether or not it needs to divide the arc into quadrants.

Example:

arc_quadrants = Y

```
%
O5000 (.TAP)

G54
M98 P9011
M01
N1 M6 T1
M8

M98 P1 (F-profile-T1)
M98 P9010
M30
O1
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X26. Y-4. Z9.
  Z1.
G1 Z-6. F33
G2 X-4. Y26. R30. F100
  X26. Y56. R30.
  X56. Y26. R30.
  X26. Y-4. R30.
G0 Z9.
M99
%
```

arc_quadrants = N

```
%
O5000 (.TAP)

G54
M98 P9011
M01

M8
M98 P1 (F-profile-T1)
M98 P9010
M30
O1
(-----)
```

```
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X26. Y-4. Z9.
  Z1.
G1 Z-6. F33
G2 X26. Y56. R-30. F100
  X26. Y-4. R-30.
G0 Z9.
M99
%
```

3.1.9.5. arc_gt_180 type: logical {Y/N}

Tells **SolidCAM** whether or not the machine has arcs greater than 180 degrees. If it doesn't, such an arc is split into two arcs.

Example:

arc_gt_180 = Y

```
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X43. Y100. Z10.
  Z2.
G1 Z-10. F33
G3 X151.816 Y123.75 R-57. F100
G0 Z10.
M99
```

arc_gt_180 = N

```
(-----)
(F-PROFILE-T1 - PROFILE)
(-----)
G0 X43. Y100. Z10.
  Z2.
G1 Z-10. F33
G3 X112.155 Y44.311 R57. F100
  X151.816 Y123.75 R57.
G0 Z10.
M99
```

3.1.9.6. arc_max_chord type: numeric

Max chord length (when arc is divided into linear segments).

3.1.9.7. arc_max_angle type: numeric

Max angle size (when arc is divided into linear segments).

3.1.9.8. arc_max_radius type: numeric

The maximum arc radius value above which the arc is divided into linear segments.

3.1.9.9. arc_min_length type: numeric

The smallest arc value below which the arc is converted into a line.

3.1.9.10. arc_zx_yz type: logical {Y/N}

If Yes, it means that **SolidCAM** will generate G_Code that includes arcs in plane ZX (G18) and in plane YZ (G19).

If No, the arc is divided to into linear segments.

3.1.10. Epsilon Values

3.1.10.1. eps_angle type: numeric

The smallest arc angle below which the arc is converted into a line.

3.1.10.2. eps_line type: numeric

The smallest line length below which the line is eliminated.

3.1.10.3. zero_value type: numeric

The smallest value below which a coordinate is rounded to zero.

3.1.10.4. min_delt_arc_rad type: numeric

If tool-radius compensation is active, each arc radius must not be smaller than the tool radius plus this value.

3.1.10.5. safety_dist type: numeric

This parameter defines a default value for the 'safety_dist' parameter used in the various job data-screens.

3.1.11 Feed-Spin

3.1.11.1. rapid_feed type: numeric

This parameter defines the feed rate of the rapid positioning movement of the machine. Units: MM/MIN or INCH/MIN as defined by current units setting.

3.1.11.2. max_spin type: numeric

This is the highest spin rate allowed in the machine in cutting movements. The spin value you specify is limited by this value.

If CSS option is used (in Turning) this value is used to limit the machine spin rate.

3.1.11.3. max_feed type: numeric

The highest feed rate allowed in the machine in cutting movements. The feed value you specify is limited by this value.

3.1.11.4. spin_direction type: integer (enumerated) {CCW/CW}

This option is used to specify the spin direction in the G-code programs.

3.1.12. Timing

3.1.12.1. time_factor type:numeric

The tool path time calculated in the simulation process is multiplied by this factor. It is used to improve the accuracy of the working time calculations.

3.1.12.2. block_time type:numeric

This parameter determines the extra time (in seconds) of every movement block. It is added to every block in order to improve the accuracy of the working time calculation.

3.1.12.3. change_tool_time type:numeric

This parameter determines the time (in seconds) of tool change; it is used for working time calculations.

3.1.16. Turning Cycles

3.1.16.1. turn_type

This parameter can be given more than once, defining additional turning cycles in each line. Each turn_type is followed by:

- turn_type_name
- Flag indicating if it can be used as a cycle.

If Yes, this means that either the CNC-controller has such a cycle or that the cycle is defined inside the [machine.GPP] file.

- The turning parameters; these parameters are later prompted in the Turning-Job screen. All parameters are of numeric type.

3.1.17. Threading Cycles

3.1.17.1. thread_type

This parameter can be given more than once, defining additional thread cycles in each line. Each thread_type is followed by:

- thread_type_name
- Flag indicating if it can be used as a cycle.

If Yes, this means that either the CNC-controller has such a cycle or that the cycle is defined inside the [machine.GPP] file.

- The thread parameters; these parameters are later prompted in the Thread-Job screen. All parameters are of numeric type.

3.1.18. Grooving Cycles

3.1.18.1. groove_type

This parameter can be given more than once, defining additional grooving cycles in each line. Each groove_type is followed by:

- Groove_type_name
- Flag indicating if it can be used as a cycle.

If Yes, this means that either the CNC-controller has such a cycle or that the cycle is defined inside the [machine.GPP] file.

- The grooving parameters; these parameters are later prompted in the Groove-Job screen. All parameters are of numeric type.

3.1.19. Turning definitions

3.1.19.1. turning_cycle type: logical {Y/N}

This parameter determines whether the machine has a turning cycle.

3.1.19.2. groove_cycle type: logical {Y/N}

This parameter determines whether the machine has a groove cycle.

3.1.19.3. combined_cycles type: logical {Y/N}

This parameter determines whether the machine has a combined cycle process which supports Rough/Copy, semi-finish and finish movements together in one cycle. If not, **SolidCAM** will generate three different cycles, one for each movement.

3.1.19.4. optimize_cycle type: logical {Y/N}

This parameter determines the turning rough cycle method of the machine. It is used to determine the cycle start point automatically.

3.1.19.5. finish_retreat type: logical {Y/N}

This parameter determines whether the finish process of the machine's cycle retreats to the start point ('Y'), or stays on the last point ('N').

3.1.19.6. semi_finish_retreat type: logical {Y/N}

This parameter determines whether the semi-finish process of the machine's cycle retreats to the start point ('Y'), or stays on the last point ('N').

3.1.19.7. fanuc_cycle type: logical {Y/N}

This parameter determines the simulation method of a turning rough cycle.

- ◆ If Yes, the simulation is identical to the FANUC machine cycle.
- ◆ If No, the simulation is identical to the OKUMA machine cycle.

3.1.20. Fourth axis

3.1.20.1. `indexial_4th_axis` type:logical {Y/N}

This parameter is relevant for machines that have simultaneous 4th axis capabilities, but their 4th axis can get values in constant increments only.

3.1.20.2. `indexial_increment` type:numeric

When 'indexial_4th_axis' is 'Y', than this parameter defines the constant increment that the machine's 4th axis can get.

3.1.20.3. `init_cpos` type:logical {Y/N}

This parameter defines whether the machine can define the angle of the home i.e., the machine can set the current rotation angle to be a different given angle.

For example:

'init_cpos' should be 'Y' if the machine can be set so that the current rotation angle of 390 is set to 30.

3.1.20.4. `polar_4x` type : logical {Y/N}

If Yes, enables efficient G-Code for machines with forth axis, that know how to break tool path in polar units, into small lines.

3.1.20.5. `cartez_4x` type : logical {Y/N}

If Yes, enables efficient G-Code for machines with forth axis, that know how to break tool path in cartezian units, into small lines.

3.1.20.6. `set_dir` type : logical {Y/N}

If Yes, enables G-Code that uses C axis of machines with forth axis (CW/CCW). This enables use of normalized C units.

3.1.20.7. `Fourth_axis_letter` type: integer {C}

Defines the fourth axes letter in the show data simulation screen.

3.1.21. Sim Five axis

3.1.21.1. kinematic_type type: integer (enumerator)

{TABLE_TABLE , HEAD_HEAD, HEAD_TABLE}

This parameter determine the cinematic type of the machine: there are three different common types of 5 axis machines:

- ◆ HEAD_HEAD both rotation axes are mounted on the head of the machine
- ◆ TABLE_TABLE both rotation axes are mounted on the table
- ◆ HEAD_TABLE one rotation axis is mounted on the head another one on the table of the machine

3.1.21.2. spindle_direction type: numeric (x, y, z)

This parameter defines the spindle direction.

Default value: 0.0000 0.0000 1.0000

3.1.21.3. rotate_axis_dir1 type: numeric (x, y, z)

This parameter defines the directions of the first rotation axes.

Default value: 0.0000 0.0000 -1.0000

3.1.21.4. rotate_axis_dir2 type: numeric (x, y, z)

This parameter defines the directions of the second rotation axes

Default value: 0.0000 -1.0000 0.0000

Note:

If the spindle direction is parallel to z axis then the C axis must be always the first rotation axis. You will get an invalid machine configuration if the C axis is defined as the second rotation axis.

3.1.21.5. rot_axis_base_pnt1 type: numeric (x, y, z)

This parameter defines the base points of the first rotation axes.

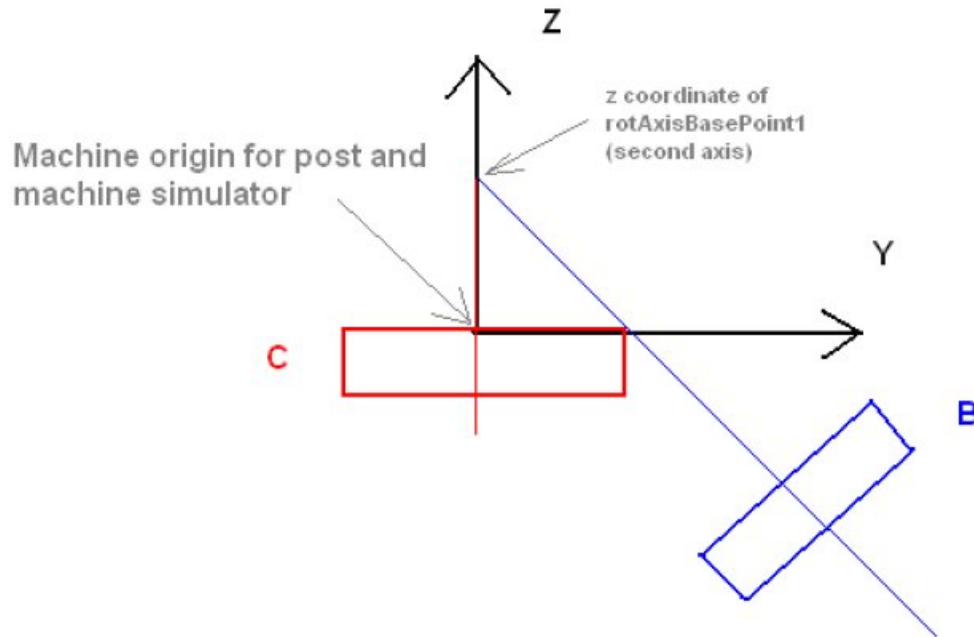
Default value: 0.0000 0.0000 0.0000

3.1.21.6. rot_axis_base_pnt2 type: numeric (x, y, z)

This parameter defines the base points of the second rotation axes.

Default value: 0.0000 0.0000 0.0000

In general, the `rotAxisBasePoint0` is expected to be 0,0,0. The z component of the `rotAxisBasePoint1` defines the height where lines describing both rotational axes are intersecting. The following picture shows the z component of the second rotary axis base point:



3.1.21.7. `abs_machine_coord` type : logical {Y/N}

If Yes, this means that the CNC-controller has a Mcode that enable to execute the wright movements according to the G_Code that **SsolidCAM** generate.

If No, **SolidCAM** make the calculation and set the wright value for the movements. The calculation is done as reference to the defined home.

3.1.21.8. `pivot_length` type : numeric

This parameter defines the distance from the face of the machine to the rotation point of the two rotary axis.

It is used only in case that `kinematic_type` is HEAD_HEAD.

3.1.21.9. `first_rot_axis_name` type: integer (enumerator) {A, B, C}

This parameter defines the first rotation axis name.

This parameter is related to the two parameters Spindle direction and `rotate_axis_dir1`.

The common value of this parameter is : C.

3.1.21.10. second_rot_axis_name type: integer (enumerator) {A, B, C}

This parameter defines the first rotation axis name.

This parameter is related to the two parameters Spindle direction and rotate_axis_dir2.

The common values of this parameter are :A or B.

3.1.21.11. machine_simulation type: string {filename}

This parameter defines the machine simulation name.

The parameter value must include the full path for necessary machine simulation file name as exists in the Sim 5Axes Machine Simulation definition subdirectory.

\\SolidCAM2005\Tables\Metric\MachSim\xml

3.1.21.12. rot_axis_min_limit0 type :numeric

This parameter defines the min limit for the rotational axis set in parameter first_rot_axis_name.

3.1.21.13. rot_axis_min_limit1 type :numeric

This parameter defines the min limit for the rotational axis set in parameter second_rot_axis_name.

3.1.21.14. rot_axis_max_limit0 type :numeric

This parameter defines the max limit for the rotational axis set in parameter first_rot_axis_name.

3.1.21.15. rot_axis_max_limit1 type :numeric

This parameter defines the max limit for the rotational axis set in parameter second_rot_axis_name.

3.1.21.16. auto_angle_pair type : logical {Y/N}

This parameter is the auto angle pair selection flag of the post processor. A 5 axis vector can always be mapped into two different angle pairs. This flag is set to true, if the post should determine automatically which pair of angle to use. In this case, the post would select the angle pair that would not create a very large angle change from previous posted position. Some machines can only use one of the angle pairs due to mechanical limitations. In such case, this flag must be set true and the angle select other pair parameter should be utilized to use the one or the other angle pair.

3.1.21.17. other_angle_pair type : logical {Y/N}

This parameter tells the post to take the one angle pair or the other angle pair (there are always 2 mathematically possible pairs) This parameter is used only if auto angle pair is turned off. If auto angle pair is turned on, then this flag is used to determine the selection of the very first angle pair. After the first move, all subsequent angle pair selections will be performed by the auto selection.

3.1.21.18. angle_change_limit type :numeric

This value in degrees sets the angle change limit in degrees from one posted tool path position to the next one. If the angle change between two tool path positions is larger than this limit, then a retract move is applied by the post. The retract distance is set by the user in a further parameter. The default value is 150 degrees. E.g. if this limit is 100 degree and the C axis is moving from C10 to C170, then such a move will be considered as a large angle change and a retract motion is applied.

3.1.21.19. interplat_angle_step type :numeric

This value is the interpolation angle step in degrees, the post processor will interpolate between posted tool path positions using this angle. The default value is 3 degrees, E.g. if the C axis moves from C10 to C22, then the interpolator will add C13, C16, C19 moves between C10 and C22.

3.1.21.20. interplat_for_dist type : logical {Y/N}

This parameter sets the interpolator for distance on and off; this is default wise set to false

3.1.21.21. interplat_distance type :numeric

This function sets the interpolation distance in units (mm or inch). The post processor will interpolate between two tool tip positions (part coord.) using this threshold value. The default value is 1mm for metric units and 0.05 in for English units E.g. if the tool tip moves from 0,0,0 to 0,0,100 and the interpolation distance is set to 10, then 9 moves are added between the start and end moves, the interpolator will add 0,0,10, then 0,0,20 etc.

3.1.21.22. retract_distance type :numeric

This value determines the retract distance of the tool from the part if a large angle change is detected based on the limit defined by angle change limit.
The default value is 100 mm for metric and 4 in for English units

3.1.21.23. **center_rot_mac_num** type: integer

This parameter defines the home number that around it all the rotation movements of the transformation done.

3.1.21.24. **min_inverse_feed** type :numeric

This parameter defines the line generation time.

This is the `inverse_feed` value which used for `Line_4x` and `Line_5x` in case that the user defined feed is small or equal 0.001.

If user defined feed is bigger then 0.001 the inverse feed is calculated according to the following formula:

$$\text{Inverse_feed} = \langle \text{Line length} \rangle / \text{feed}$$

3.1.21.25. **enable_mx_edit** type : logical {Y/N}

If yes, it enable to change inside the 5 Axes Job the value of the following two parameters:

`auto_angle_pair`
`other_angle_pair`

3.1.21.26. **pole_angle_tolerance** type :numeric

This parameter defines the pole areas where the rotary axis and spindle direction are parallel.

3.1.21.27. **use_machine_limits** type : logical {Y/N}

This parameter defines whether to use the machine limits in both translational and rotational axis.

If No, then the machine limits defined in machine definition will be ignored. This is the default setting.

If Yes, then the machine limits defined in machine definition object (both translational and rotational limits) are used.

In this case the result of post processing will be checked against the machine limits and an exception is thrown if the posted tool path is out of the given limits.

Further, this flag implies that the "Auto Angle Pair" selection (see documentation for this item) is based on the machine limits. E.g. if a B axis is limited between 0 and 90 degrees, then the post will select one of the angle pair results that is within this given range of 0 and 90 degrees. An example is that both $B = -30$ and $B = +30$ are possible solutions. In this case, the solution $B = -30$ is not used and $B = +30$ is used since $+30$ degrees is within 0 and 90 degrees. If "Use Machine Limits" is checked, before selecting the angle pair, the two pairs are checked against the angle machine limits (as they are set in the "Machine Definition" tab). If only one pair is valid, this is the pair to be used. If both angle pairs are valid, the one that

creates the smallest change is selected, as usual.

3.1.21.28. trans_axis_min_limit type: numeric (x, y, z)

This parameter defines the min limit for the translation axis.
Default value: -100000, -100000, -100000

3.1.21.29. trans_axis_max_limit type: numeric (x, y, z)

This parameter defines the max limit for the translation axis.
Default value: 100000, 100000, 100000

3.1.22. Wire Cut Cycles

3.1.22.1. wc_macro_type

This parameter can be given more than once, defining additional machine (macro) cycles in each line. Each `wc_macro_type` is followed by:

- `Wc_macro_type_name`
- The `wc_macro_type` parameters; these parameters are later prompted in the `wc_macro_job` screen. All parameters are of numeric type.

3.1.23. Wire Cut parameters

3.1.23.1. lower_guide_level type:numeric

This parameter defines the default value of the wirecut machine lower guide level. You can change this parameter in Home List table.

3.1.23.2. group_def type : logical {Y/N}

This parameter enables you either open or close the E-Group editing item in TOOL menu. If the `group_def` parameter is Yes the work with the E-Group definition table is enabled. If the `group_def` parameter is No the access to the E-Group definition table is denied.

3.1.23.3. offset_group_name type: string

This parameter defines the filename of wirecut offset table and E-Group definition table. The wirecut offset table will be saved in the file with this name and the ".TAB" extension. The wirecut E-Group definition table will be saved in the file with this name and the ".GRP" extension. The parameter value must include the full path for necessary file from the GPPTOOL subdirectory. For example,

offset_group_name = wire1 - offset table and E-group definition files will be placed in the machines directory (GPPTOOL)

offset_group_name = ..\user\wire2 - offset table and E-Group definition files will be placed in the USER subdirectory

3.1.23.4. u_max type:numeric

This parameter defines the maximal distance in X axis between lower-guide and upper-guide.

3.1.23.5. v_max type:numeric

This parameter defines the maximal distance in Y axis between lower-guide and upper-guide.

3.1.23.6. xy_abs type : logical {Y/N}

- ◆ If TRUE, lower-guide coordinates are given in absolute distance from home position.
- ◆ If FALSE, lower-guide coordinates are given in relative distance from previous position.

3.1.23.7. uv_abs type : logical {Y/N}

- ◆ if TRUE, upper-guide coordinates are given in absolute distance from home position.
- ◆ if FALSE, upper-guide coordinates are given in relative distance from previous position.

3.1.23.8. agie type : logical {Y/N}

Defines whether the CNC wirecut machine controller is agie.

3.2. MacEdit Utility

The Macedit Utility enable to change each of the parameters defined in the <machine>.mac file.

To change one of the parameters, click on the appropriate item in the tree structure of the MacEdit.

There are the following pages available within the MacEdit tree:

- Internal Parm
- Machine Initialize
- Program numbers
- Procedures control
- Home
- Positioning
- Compensation
- Line Definitions
- Arc Definitions
- Epsilon Values
- Feed-Spin
- Timing
- Part Options
- Job Options
- Drill Cycles
- Turning Cycles
- Threading Cycles
- Grooving Cycles
- Turning Definitions
- Fourth axis
- Five axis
- Wire Cut Cycles
- Wire Cut Parameters

3.2.1. MacEdit Installation

Macedit is installed on the computer during the installation of the **SolidCAM** program.

3.2.2. Running MacEdit

The **MacEdit** program is run as a separate stand-alone program; It receives as input a <machine>.mac and then update it.

Run MacEdit by Double click on file named MaceEdit.exe that is located on the following folder:

\Program Files\SolidCAM2005\SolidCAM

3.3. Examples of [machine.mac] files

3.3.1. FANUC.mac

; FANUC

pre_processor

```

;
;Internal parms
;

machine_type           = MILLING
post_processor         = fanuc
doc_processor          = fanuc
gpp_file_ext           = TAP
mac_holder             = holder
tool_table_name        = table
max_g_name_length      = 0
max_tool_numbers       = 1000
default_lang           = DEFAULT

```

```

;
;Machine Initialize
;

```

```

machine_plane          = XY
z_with_xy              = Y
mac_axes               = XYZ
num_axes               = 4
num_simult_axes        = 4
abs_coord              = N
rotate                 = Y
mirror                 = Y
variables              = Y
loops                  = Y
_4th_axes_around       = X
first_rotation_angle   = Z
_5th_axes_around       =

```

```

;
;Program numbers
;

```

```

prog_num_min           = 5000
prog_num_max           = 8999
prog_num_dflt          = 5000
get_prog_num           = Y
proc_num_min           = 1
proc_num_max           = 8999

```

```

proc_num_dflt      = 1
get_proc_num      = Y

;
;Procedures control
;

full_gcode        = N
gen_procs         = N
drill_proc        = N
turn_proc         = Y
thread_proc       = Y
gen_internal_proc = N
turn_common_proc  = Y
gen_1_line_proc   = Y
optimize_jobs_loop = Y
G_in_2_cols       = Y
seq_sub_number    = N
loop_exist        = Y
same_sub_numbers  = N
init_var_after_split = Y

;
;Home
;

num_homes         = 6
dflt_home         = 1
get_job_home      = N
abs_zero_chng     = Y
home_data_at_start = N

;
;Positioning
;

dflt_start        = 0.0000 200.0000 100.0000, 0.0000 7.8740 3.9370
dflt_end          = 0.0000 200.0000 0.0000, 0.0000 7.8740 0.0000
set_xy_chng       = N N
set_z_chng        = Y N
dflt_tool_chng    = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000

;
;Compensation
;

comp_exist        = Y
comp_arc_arc      = Y
comp_arc_line     = Y
comp_line_line    = Y
next_angle        = N
comp_x_start      = N
comp_by_arcs      = N

```

```
chgng_tool_table      = N
look_forward         = 2
delta_for_TOOL_H     = 50
comp_by_zero_tool    = N

;
;Arc definitions
;

arc_exist            = Y
arc_3d               = Y
arc_quadrants        = N
arc_gt_180           = Y
arc_max_chord        = 30.0000, 1.1811
arc_max_angle        = 10.0000
arc_max_radius       = 2000.0000, 78.7402
arc_min_length       = 0.0000, 0.0000
arc_zx_yz            = N

;
;Epsilon values
;

eps_angle            = 0.0020, 0.0001
eps_line             = 0.0010, 0.0000
zero_value           = 0.0010, 0.0000
movement_precision   = 0.0010, 0.0000
min_delt_arc_rad     = 0.0100, 0.0004
safety_dist          = 2.0000, 0.0787

;
;Feed-Spin
;

rapid_feed           = 5000.0000, 196.8503
max_spin             = 6000.0000
max_feed             = 6000.0000, 236.2205
spin_direction       = CW

;
;Timing
;

time_factor          = 1.0000
block_time           = 0.2000
change_tool_time     = 15.0000

;
;Part options
;

options              = COLENT INTEGER
```

```

;
;Job options
;

job_opt_type          = OPT1 Y DELY FEAD

;
;Drill cycles
;

drill_type            = Drilling Drilling Y
drill_type            = F_Drill F_Drill Y Delay
drill_type            = Peck Peck Y Delay
drill_type            = Tapping Tapping Y
drill_type            = Boring Boring Y Delay
drill_type            = R_Boring R_Boring Y Delay
drill_type            = F_Boring F_Boring Y Delay

;
;Fourth axis
;

indexial_4th_axis    = N
indexial_increment    = 0.0000, 0.0000
init_cpos             = Y
polar_4x              = N
cartez_4x             = N
set_dir               = N
fourth_axis_letter   = C

;
;Sim Five axis
;

kinematic_type        = HEAD_HEAD
spindle_direction     = 0.0000 0.0000 1.0000
rotate_axis_dir1      = 0.0000 0.0000 -1.0000
rotate_axis_dir2      = 0.0000 -1.0000 0.0000
rot_axis_base_pnt1    = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
rot_axis_base_pnt2    = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
abs_machine_coord     = N
pivot_length          = 0.0000, 0.0000
first_rot_axis_name   = C
second_rot_axis_name  = B
machine_simulation    = HeadHead
rot_axis_min_limit0   = -100000.0000
rot_axis_min_limit1   = -100000.0000
rot_axis_max_limit0   = 100000.0000
rot_axis_max_limit1   = 100000.0000
auto_angle_pair       = Y
other_angle_pair      = Y
angle_change_limit    = 30.0000

```

```
interplat_angle_step      = 3.0000
interplat_for_dist       = N
interplat_distance       = 5.0000, 0.1969
retract_distance        = 100.0000, 3.9370
center_rot_mac_num      = 20
min_inverse_feed        = 100.0000, 3.9370
enable_mx_edit          = N

;
endp
```

3.3.2. MAHO.mac

; MAHO-432

```
;Internal parms
machine_type           = MILLING
post_processor        = MAHO432
doc_processor        = MAHO432
gpp_file_ext         = TAP
mac_holder           = holder
tool_table_name      = table
max_g_name_length    = 0
max_tool_numbers     = 1000
default_lang         = DEFAULT
```

```
;Machine Initialize
machine_plane        = ZX
z_with_xy           = Y
mac_axes            = XYZC
num_axes            = 4
num_simult_axes     = 4
abs_coord           = N
rotate              = Y
mirror              = Y
variables           = Y
loops               = Y
_4th_axes_around   = Y
first_rotation_angle = Z
_5th_axes_around   =
```

```
;Program numbers
prog_num_min        = 10000
prog_num_max        = 9999999
prog_num_dflt       = 10000
get_prog_num        = Y
proc_num_min        = 10000
proc_num_max        = 99999
proc_num_dflt       = 10000
get_proc_num        = N
```

```
;Procedures control
full_gcode          = N
gen_procs           = Y
drill_proc          = Y
turn_proc           = Y
thread_proc         = Y
gen_internal_proc   = Y
turn_common_proc    = Y
gen_1_line_proc     = Y
optimize_jobs_loop  = N
```

```

G_in_2_cols           = N
seq_sub_number        = N
loop_exist            = Y
same_sub_numbers      = N
init_var_after_split  = Y

;Home
num_homes              = 6
dflt_home              = 1
get_job_home          = Y
abs_zero_chng         = Y
home_data_at_start    = N

;Positioning
dflt_start             = 0.0000 100.0000 50.0000, 0.0000 3.9370 1.9685
dflt_end               = 0.0000 100.0000 50.0000, 0.0000 3.9370 1.9685
set_xy_chng           = N N
set_z_chng            = N N
dflt_tool_chng        = 0.0000 100.0000 50.0000, 0.0000 3.9370 1.9685

;Compensation
comp_exist            = Y N
comp_arc_arc          = Y
comp_arc_line         = Y
comp_line_line        = Y
next_angle            = N
comp_x_start          = Y
comp_by_arcs          = N
chng_tool_table       = N
look_forward          = 2
delta_for_TOOL_H      = 50
comp_by_zero_tool     = N

;Arc definitions
arc_exist             = Y
arc_3d                = Y
arc_quadrants         = N
arc_gt_180            = Y
arc_max_chord         = 2.0000, 0.0787
arc_max_angle         = 10.0000
arc_max_radius        = 2000.0000, 78.7402
arc_min_length        = 0.0000, 0.0000
arc_zx_yz             = N

;Epsilon values
eps_angle             = 0.0020, 0.0001
eps_line              = 0.0010, 0.0000
zero_value            = 0.0010, 0.0000
movement_precision    = 0.0010, 0.0000
min_delt_arc_rad      = 0.0100, 0.0004
safety_dist           = 2.0000, 0.0787

;Feed-Spin

```

```

rapid_feed          = 15000.0000, 590.5512
max_spin            = 6000.0000
max_feed            = 15000.0000, 590.5512
spin_direction      = CW

;Timing
time_factor         = 1.0000
block_time          = 0.0000
change_tool_time    = 0.0000

;Part options
options             = inc_change LOGICAL
options             = G99_x NUMERIC
options             = G99_y NUMERIC
options             = G99_z NUMERIC
options             = G99_i NUMERIC
options             = G99_j NUMERIC
options             = G99_k NUMERIC
options             = Delta_Mt NUMERIC

;Job options
job_opt_type        = PLANE_4X N XY_0_YZ_1_ZX_2

;Drill cycles
drill_type          = G81 Drilling Y
drill_type          = G83 Peck Y
drill_type          = G84 Tapping Y
drill_type          = G73 Peck Y
;Wire Cut cycles

;Fourth axis
indexial_4th_axis   = N
indexial_increment  = 0.0000, 0.0000
init_cpos           = Y
polar_4x            = N
cartez_4x           = N
set_dir             = N
fourth_axis_letter  = C

;Sim Five axis
kinematic_type      = HEAD_HEAD
spindle_direction   = 0.0000 0.0000 1.0000
rotate_axis_dir1    = 0.0000 0.0000 -1.0000
rotate_axis_dir2    = 0.0000 -1.0000 0.0000
rot_axis_base_pnt1  = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
rot_axis_base_pnt2  = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
abs_machine_coord    = N
pivot_length        = 0.0000, 0.0000
first_rot_axis_name  = C
second_rot_axis_name = B
machine_simulation   = HeadHead
rot_axis_min_limit0 = -100000.0000

```

```
rot_axis_min_limit1      = -100000.0000
rot_axis_max_limit0     = 100000.0000
rot_axis_max_limit1     = 100000.0000
auto_angle_pair         = Y
other_angle_pair        = Y
angle_change_limit      = 30.0000
interplat_angle_step    = 3.0000
interplat_for_dist      = N
interplat_distance      = 5.0000, 0.1969
retract_distance        = 100.0000, 3.9370
center_rot_mac_num      = 20
min_inverse_feed        = 100.0000, 3.9370
enable_mx_edit          = N
```

```
endp
```

CHAPTER 4

4.1. Internal Fast Post-Processor

The Fast-Processor supports the following CNC-Controllers.

```
FANUC
TIGER_6L
TIGER_TL
_426_PARM
_426PRMCR
MAKINO_W
FANUC_3D
MAHO432_3D
_426_PARM_3D
SIN810_4_3D
OKUMA_3D
FANUC_3D_JAP
ROEDERS
MIKRON23
ROEDERS4
FANUC_3D_KOR
```

The postprocessor name defined in the mac file must be proceed with '*';
post_processor = *426_parm_3D

4.1.1 Introduction

The Program's name is 'GPPToolExe.exe'.
GPPToolExe.exe is located on the following folder:
 \Program Files\SolidCAM2005\SolidCAM
It allows you to choose generate G-code and Cam settings.

4.1.2. Start Program

Choose the menu item "Start" to start the Program and generate G-code.
A dialog box titled "SOLIDCAM-Parts" will appears. Choose the required part. The Gcode of this part will be generated.

If you want to generate G-code with traces click on the check box of "Gcode trace" in the dialog box Settings -> Mac Settings.

The Program produces a G-Code file named as the Part name. The G-Code file is placed in the directory of the chosen Part.

4.1.3. CAM Settings

In the settings dialog you can change the path to your post-processor files and to your user-directory.

There are the following related pages available within the SolidCAM Settings dialog:

- User Directories settings
- Default CNC-Controller

User Directories Settings

The user directory defines the default folder in which CAM-Parts and G-Code files created with SolidCAM are stored. Whenever you load a CAM-Part, SolidCAM will browse the user directory for SolidCAM-Part files (*.prt). All newly created CAM-Part files will be placed in the user directory by default.

Default CNC-controller settings

Post-processor files directory

GPP files with the extensions *.mac and *.gpp are post-processor files used by SolidCAM.

- The *.mac file contains information about the CNC-machine, e.g. number of simultaneous axes, available cycles, default tool positioning, etc..
- The *.gpp file translates the calculated tool path into G-Code. The G-Code format, cycle definitions, etc. of your G-Code file is controlled through this file.

You have to specify the path to the folder where you keep the *.mac and *.gpp files you have received from SolidCAM. The post-processors that come with the evaluation or demo version are installed in the SolidCAM program directory.

CNC-controllers

SolidCAM enables you to define the default machine controllers:

- **Milling CNC-Controller**
- **Turning CNC-Controller**
- **Milling&Turning CNC-Controller**
- **WireCut CNC-Controller**

These post-processors you selected will be used as the default machine, which means it will appear in the CAM-Part Data dialog as the controller for the new CAM-Part. Naturally, you can change the controller for the CAM-Part with the list field if you want to use another machine controller.

4.2. User Documentation

This option enables you to generate documentation files in different formats for example: Html, Excel, txt, etc...

4.2.1. Mac Doc_processor

You can customize the Documentation output of SOLIDCAM by editing the machine *.mac file and the Doc-processor *.dpp file of your CNC-controller.

The documentation postprocessor includes two files: <machine_name>.dpp and <machine_name>.mac.

These files have to be located on the post-processor files directory defined in the SolidCAM Settings:

```
\Program Files\SOLIDCAM2005\Gpptool
```

A parameter named doc_processor has to be defined in the file: machine_name>.mac

The parameter is defined as follows:

```
doc_processor      = <machine_name>
```

4.2.2. SOLIDCAM User Documentation Commands

When the user asks for the generation of the User Documentation, the documentation is then generated according to the [MACHINE.MAC] and [MACHINE.DPP] files.

There are some new GPPL commands that are helpful in generating the documentation file :

- Create the doc file at the beginning of the documentation generation process:
{ '!! open file = c:\\name.ext !!' }
- Close the doc file at the end of the documentation generation process:
{ '!! close file = c:\\name.ext !!' }
- Copy doc file
{ '!! copy file = c:\\copy_from_name.ext !!' }
- Delete temporary file
{ '!! delete file = c:\\temporary_file_name.ext !!' }

4.2.3. Example of User Documentation

The following example generates the file in Html format named tool_table.html

1	OBR	Machine description	Arrow					
2	PAR	Type machine	0					
3	MZA	Work material	None					
.								
4	TPR	Part name	PROFILE					
5	TPR	Job name	F-1-T1					
6	PAR	Down step						3.
7	PAR	Toolpath length						1051.426
8	PAR	Surface offset						0.
9	PAR	Wall offset						0.
10	PAR	Floor offset						0.
11	PAR	Island offset						
12	PAR	Number passed						2.
13	PAR	Feed rate						100.
14	PAR	Feed Z						33.
15	PAR	Feed finish						100.
16	PAR	Spin rate						1000.
17	PAR	Spin finish						1000.
18	PAR	Job time						8.349
19	INS	Tool message						
20	PAR	Tool ID	0-0					
21	PAR	Tool material	None					
22	PAR	Holder_description						
.								

The following command needs to be added to the <machine>.dpp file:

Step 1: Define the Variables:

```
@init_post
  global integer tool_count
  global string tool_def<<20,50>>
  global integer job_count
  global string job_def<<20,150>>
```

endp

Step 2 – Enter values to the variable

```

@start_of_job
  message = msg

job_count = job_count + 1
  job_def<<1,job_count>> = job_name
  job_def<<2,job_count>> = tostr(down_step)
  job_def<<3,job_count>> = tostr(toolpath_length)
  job_def<<4,job_count>> = tostr(surface_offset)
  job_def<<5,job_count>> = tostr(wall_offset)
  job_def<<6,job_count>> = tostr(floor_offset)
  job_def<<7,job_count>> = tostr(island_offset)

Npr=(abs(depth/down_step)-round(abs(depth/down_step),0))
If Npr > 0 then
Np=round(abs(depth/down_step),0)+1
else
Np=round(abs(depth/down_step),0)
endif

  job_def<<8,job_count>> = tostr(Np)
  job_def<<9,job_count>> = tostr(feed_rate)
  job_def<<10,job_count>> = tostr(z_feed)
  job_def<<11,job_count>> = tostr(finish_feed)
  job_def<<12,job_count>> = tostr(spin_rate)
  job_def<<13,job_count>> = tostr(finish_spin)
  job_def<<14,job_count>> = tostr(job_dtime)
  job_def<<15,job_count>> = tool_message

t_length =tostr(tool_ID_number:'8.0(p)')
t1_length=left( t_length,4)
t2_length=right( t_length,4)

  job_def<<16,job_count>> = t1_length+'-'+ t2_length
  job_def<<17,job_count>> = tool_material
  job_def<<18,job_count>> = holder_description

; {(MSG, 'message')}
endp

@def_tool

```

```
{nb,'(TOOL NUMBER ' tool_NUMBER ' tool_length ' tool_length ' tool_teeth_number '  
tool_teeth_number }  
{ ' corner_radius ' corner_radius }  
  
tool_count = tool_count + 1  
tool_def<<1,tool_count>> = tostr(tool_number:integer_def_f)  
tool_def<<2,tool_count>> = tostr(tool_offset)  
tool_def<<3,tool_count>> = tostr(tool_length )  
tool_def<<4,tool_count>> = tostr(tool_teeth_number:integer_def_f)  
tool_def<<5,tool_count>> = tostr(corner_radius)  
tool_def<<6,tool_count>> = tostr(tool_type)  
tool_def<<7,tool_count>> = tostr(tool_feed_z)  
tool_def<<8,tool_count>> = tostr(tool_feed_finish)  
tool_def<<9,tool_count>> = tostr(tool_spin_finish)  
tool_def<<10,tool_count>> = tostr(tool_id_number:integer_def_f)  
tool_def<<11,tool_count>> = tostr(tool_angle)  
tool_def<<12,tool_count>> = tool_user_type  
tool_def<<13,tool_count>> = tostr(program_number:integer_def_f)  
tool_def<<14,tool_count>> = g_file_name  
tool_def<<15,tool_count>> = 'DMU70_M'  
tool_def<<16,tool_count>> = ''  
tool_def<<17,tool_count>> = tool_message  
endp
```

step 3 – Call @doc procedure that will generate the documentation.

```
@end_of_file
```

```
{nb, ''}  
call @doc  
endp
```

Step 4 – Commands generating the documentation.

The procedure has to include a command at the beginning to open the doc file you want to generate and to close it at the end.

```
@doc
```

```
local string t_num  
local integer i num  
local string job_table_file
```

```
job_table_file = 'f:\\SolidCAM2004\\Gpptoool\\job_table.html'
```

```

    {nl,'!!open file=' job_table_file '!!'}
{nl,'<html>'}
{nl,''}
{nl,'<head>'}
{nl,'<meta http-equiv="Content-Language" content="en-gb">'}
{nl,'<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">'}
{nl,'<meta name="GENERATOR" content="Microsoft FrontPage 4.0">'}
{nl,'<meta name="ProgId" content="FrontPage.Editor.Document">'}
{nl,'<title>SolidCAM SetUp Sheet</title>'}
{nl,'</head>'}
{nl,''}
{nl,'<body>'}
{nl,''}
;
{nl,' </table>'}
{nl,' </left>'}

```

```

{nl,' <table border="1" width="81%">'} ; ñîçääîèå ðàîèè

```

```

num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>OBR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Machine description</td>'}
{nl,' <td>'machine_descr'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Type machine</td>'}
{nl,' <td>'job_machine_type'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}

```

```

{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>MZA</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Work material</td>'}
{nl,' <td>'work_material'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
{nl,' </tr>'}
{nl,' <td bgcolor="#C0C0C0">.</td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' </tr>'}

i = 1
while i <= job_count
    if i <= job_count

num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>TPR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Part name</td>'}
{nl,' <td>'f_name '</td>'}
{nl,' <td>&nbsp;</td>'}

```

```

{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>TPR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Job name</td>'}
{nl,' <td>'job_def<<1,i>>'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Down step</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<2,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Toolpath length</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}

```



```

{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<6,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Island offset</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<7,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Number passed</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<8,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Feed rate</td>'}
{nl,' <td>&nbsp;</td>'}

```

```
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<9,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Feed Z</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<10,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Feed finish</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<11,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Spin rate</td>'}
{nl,' <td>&nbsp;</td>'}

```

```

{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<12,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Spin finish</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<13,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Job time</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>'job_def<<14,i>>'</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>INS</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Tool message</td>'}
{nl,' <td>'job_def<<15,i>>'</td>'}

```

```
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Tool ID</td>'}
{nl,' <td>'job_def<<16,i>>'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Tool material</td>'}
{nl,' <td>'job_def<<17,i>>'</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
num=num+1
t_num =tostr(num:'8.0(p)')
{nl,' </tr>'}
{nl,' <td>'t_num'</td>'}
{nl,' <td>PAR</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>Holder_description</td>'}
{nl,' <td>'job_def<<18,i>>'</td>'}

```

```
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' <td>&nbsp;</td>'}
{nl,' </tr>'}
;
{nl,' </tr>'}
{nl,' <td bgcolor="#C0C0C0">.</td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' <td bgcolor="#C0C0C0"> </td>'}
{nl,' </tr>'}
    else
endif
    i = i + 1
endw

    {nl,'!!close file='job_table_file'!!'}
endp
```

CHAPTER 5

5. GPPL language

5.1. Introduction

General Post Processor Language (GPPL) is a high-level programming language used to control the generation of G-code files out of **SolidCAM**'s internal language.

This chapter describes the GPPL language, its structure and functions. Although there are many examples in this chapter, you are also urged to see the examples in the next chapter, where the building blocks of GPPL are assembled together to form a complete general post processor definition.

5.2. GPPL Variables

5.2.1. Variable definition

A GPPL variable should adhere to the following rules:

- ◆ The first character should be a letter.
- ◆ All the other characters should be either a letter, digit or '_'.
- ◆ A variable name should not exceed 27 characters.

Note that GPPL does not distinguish between lower case and upper case (CAPITAL) letters.

Examples:

xpos , gcode_f , x0

5.2.2. Variable types

GPPL supports four types of variables:

- ◆ integer - can hold signed, integer numbers in the range of (-999999999, +999999999).
- ◆ numeric - can hold any number in the range of (-1.E300, +1.E300).
- ◆ logical - can hold the logical values TRUE (1) or FALSE (0).
- ◆ string - can hold any sequence of ASCII characters; the number of characters is unlimited.

5.2.3. Variable attributes

Every GPPL variable has two attributes associated with it. These attributes are defined and updated automatically by GPPL, but can also be changed directly by the user. These attributes are:

1. change attribute

This attribute is set to TRUE when a variable is first assigned a value. The next time the variable is assigned a value, if this value is different from the previous value then the change attribute is set to TRUE; if it is the same as the previous value, the change attribute is set to FALSE.

2. active attribute

This attribute is TRUE for most system variables. It is set to FALSE only for variables which are optional in certain conditions: The 'message' variable in '@call' procedure, and all user defined drill variables (as defined by 'drill-cycle' parameters in the [machine.mac] file). Local and global variables always have their active attribute set to TRUE, unless explicitly changed by the user by the attribute assignment statement.

5.2.4. Variables groups

A GPPL variable may fall into one of three groups; once a variable is defined to be in a certain group, its group cannot be changed. Variable groups are:

1. System variables

Under this group you can find all the system variables, that you can use for developing your postprocessor. These variables will be described in chapter 6. Note that you are not allowed to change the type of these variables.

2. Global user-defined variables

These variables are known (and can be changed or used) all over the GPPL procedures. They are most useful for inter-procedure data exchange. Global variables are initialized to zero and retain their previous value until assigned a new value.

3. Local user-defined variables

These variables are known (and can be changed or used) only in the GPPL procedure that defined them. They are unknown outside that procedure and their value is lost when the procedure is ended. Local variables are initialized to zero.

4. Internal variables

Under this group you can find all the internal variables, that pass the machining data information to the appropriate Tool path command. These variables will be described in chapter 7.

5. MAC external variables

Under this group you can find MAC external variables that is used to set user-defined fields in the Part-Option and Job-Option screen.

5.2.5. Array General

◆ Variable

A defined place in memory with a defined purpose and name. Variables can be of different types and different sizes. For example, a variable that is defined as an integer (i.e. can have whole values) will have two bytes in the memory.

A series of variables that have a certain logical relationship between them and are placed together are called array variables. Array variables are variables of the same type and name which can be differentiated only by their index.

For example, we want to save the first one hundred prime numbers. The array variable can be named PRIMES, its type can be defined as "integer" and its size can be 100. The first primary number will therefore be stored in the first place in this variable, the second primary number will be stored in the second place and so on.

The above is an example of a vector i.e., a variable with one dimension. It is also possible to define a variable with two or more dimensions. A two-dimensional variable is called a matrix. A simple example of a matrix is the multiplication table. A vector has one index that differentiates between its variables. A matrix, on the other hand, has two indexes: the first index symbolizes the number of the row and the second index symbolizes the number of the column.

Definition

Definition of an array variable:

```
local/global numeric/integer/string var_name<<i1, i2 ...>>
```

when $0 < i1, i2 < 10000$

The array variables are similar to the rest of the variables in GPPTOOL. You have to remember to use indexes when working with the array variables.

Examples:

1. `vec<<4>>` symbolizes the fourth element in the vector `vec`.
2. `mat<<2,3>>` symbolizes the element in the second line, in the third column in the matrix `mat`.

Below are complete examples of variables in GPPTOOL, including the definition, use of the variables and indexes.

5.3. GPL Constants

GPL supports the following constants types:

1. Integer constants:

Holds integer values in the range (-999999999, +999999999).

Examples:

1 , -123 , -99999

2. Numeric constants:

Holds any numeric value in the range (-1.E300, +1.E300). A numeric constant must have a decimal point (to distinguish it from integer constant), and may optionally have an exponent part (the letter 'E' immediately followed by the exponent value).

Examples:

1. , -123.0 , .123 , -0.23 , 1.2E2 (=120.) , 1.2E-2 (=0.012)

3. Logical constants:

Either 1 or 0. GPL has two system variables i.e., TRUE and FALSE, which have the values 1 and 0 respectively.

4. String constants:

Holds any sequence of ASCII characters. A string constant must be surrounded by either single quotes (') or by double quotes ("). A string constant might include the following characters:

- a. Printable characters: "Normal" characters (a, b, ..., z, 0, 1, ..., 9 etc). Note that GPL distinguishes between lower case and upper case (CAPITALS) letters.
- b. Special characters: Few useful non-printable characters might be used by their mnemonic, preceded by a back-slash (\):

\n	-	(ASCII 10) new line character.
\r	-	(ASCII 13) new line character.
\t	-	(ASCII 9) tabulator character.
\\	-	the back-slash character itself (\).
- c. Any character (hexadecimal notation): GPL gives you the ability to include any character in the string constant by typing its hexadecimal (base 16) value. The notation is:

\xnn - where nn are two hexadecimal digits defining the ASCII value of the character. Note that two hexadecimal digits must be specified, and that the ASCII character NUL (\x00) cannot be used.

Examples:

- 'abcdefg' - (or "abcdefg")
- 'AbcdefG'
- " - (the NULL string. Its length is 0)
- 'SUBROUTINES : ' - (note the blanks (spaces) at the end)
- 'no, no, no!' - (error: must begin and terminate with the same quote type.)
- 'abc\txyz' - (note the TAB character).
- 'abc\x04xyz' - (note the EOT (hex 04) character).

5.4. GPL Expressions

Expressions in GPL might be of numeric type (i.e expressions which produce integer/numeric values), of logical type (which produce logical TRUE/FALSE values), or of string type (which produce string values). Any expression basically consists of one or more operands, separated by operators which specifies the operation to be performed. The operands are either constants, variables or sub-expressions. The expression might contain any number of parentheses to change the normal sequence of evaluation. For example, in the expression (1+2-3), the numbers 1,2,3 are the operands and the characters +,- are the operators. You may use parantheses to change the operators' priorities or to clarify the expressions.

5.4.1. Expression types

GPL supports four types of expressions: integer, numeric, logical and string expressions.

1. Integer expression:

Any expression which includes integer or logical sub-expressions, but includes at least ONE integer sub-expression.

Examples:

1+(2-3) , tool_number+50

2. Numeric expression:

Any expression which includes integer or logical or numeric sub-expressions, but includes at least ONE numeric sub-expression.

Examples:

1.+(2-3) , xpos , ypos*100

3. Logical expression:

Any expression which includes integer or logical expressions, and results in TRUE/FALSE (1/0) value.

Examples:

xpos > -100 and xpos < 100 , FALSE , tool-number = 5

4. String expression:

Any expression which includes only string sub-expressions.

Examples:

'G-code generated on '+ date

5.4.2. Operators

GPPL supports three types of operators: integer/numeric, logical and string operators.

integer/numeric operators:

+, -, *, / basic arithmetic operators

logical operators:

NOTATION: TRUE means 1, FALSE means 0

- | | | |
|--------------|---|---------------------------------------------------------------|
| and | - | TRUE if both operands are TRUE. |
| or | - | TRUE if any of its operands is TRUE. |
| not | - | TRUE if its argument is FALSE. |
| eq (or '==') | - | TRUE if both operands are equal. |
| ne (or '<>') | - | TRUE if both operands are different. |
| le (or '<=') | - | TRUE if first operand is less than or equal to the second. |
| lt (or '<') | - | TRUE if first operand is less than the second. |
| ge (or '>=') | - | TRUE if first operand is greater than or equal to the second. |
| gt (or '>') | - | TRUE if first operand is greater than the second. |

string operators:

+ string concatenation. (concatenation is the operation of appending one string to the end of another string to form a new, longer string).

5.4.3. Operators precedence

The numeric/logical operators are evaluated according to the following precedence table (priority goes top down, left to right):

variable	constant	function	(...)		
- (unary)					
* /					
+ -					
gt ge	lt	le	eq	ne	
not					
and					
or					

Examples:

$1+2*3$

$- 1+(2*3)$

$\text{not } x\text{pos } \text{gt } 100 \text{ and } x\text{pos } \text{lt } -100 - (\text{not } (x\text{pos} > 100)) \text{ and } (x\text{pos} < (-100))$

5.4.4. Conversions

Conversion is the process used to evaluate any expression which consists of operands from different types. For example, trying to add an integer number and a numeric one should give a meaningful result.

GPPL uses a very simple (yet powerful) mechanism to handle conversions:

1st operand	2nd operand	result
numeric	logical/integer/numeric	numeric
integer	logical/integer	integer
logical	logical	logical
string	string	string

All other possibilities are meaningless and GPPL produces a proper error message (e.g., string and integer conversion).

Examples:

$10+1.2$ - 11.2 (numeric)

$x\text{pos } \text{gt } 100$ - Convert 100 to 100. and compare; result is logical

$'abc'+1$ - error (string expression cannot be converted).

$'abc'+'xyz'$ - $'abcxyz'$

5.5. GPPL statements

5.5.1. Global declaration

Format:

```
global <type> <var1>, <var2>, ...
```

Description:

This statement declares a variable to be global of type <type>. A global variable is one that is known (and can be used) all over the procedures. It retains its previous value until it assigned a new value. <type> should be logical, integer, numeric or string. It's advisable to declare all the global variables in the procedure '@init-post'.

Examples:

```
global numeric current_loc_x, current_loc_y  
global logical first_time  
global string line
```

5.5.2. Local declaration

Format:

local <type> <var1>, <var2>, ...

Description:

This statement declares a variable to be local of type <type>. A local variable is one that is known (and can be used) only in the procedure in which it is declared. Every time the procedure is started the variable is initialized to 0 (integer/numeric), FALSE (logical), and "" (string). <type> should be logical, integer, numeric or string.

Examples:

```
local numeric current_loc_x, current_loc_y
local logical condition
local string line
```

5.5.3. Assignment statements

Format:

<num var> = <num exp>
OR
<str var> = <str exp>

Description:

The assignment statement is used to assign a value to any GPL variable. The expression to the right of the equal sign is evaluated, converted to the type of the variable to the left of the equal sign, and then assigned. Note that the change attribute is calculated and set accordingly.

Examples:

- xpos = 100 - set variable xpos to 100
- ypos = ypos - 100 - Decrement variable ypos by 100
- gcode_f = '5.0(p)' - set (string) variable gcode_f to the string'5.0(p)'.

5.5.4. Attribute assignment statements

Format:

change(<var>) = <logical exp>
OR
active(<var>) = <logical exp>

Description:

The attribute assignment statement sets the change/active attribute of any variable. The logical expression at the right of the equal sign is evaluated, and must yield a TRUE/FALSE (1/0) value. Then the attribute is set.

Examples:

- change(xpos) = 1 - set the change attribute of 'xpos' to TRUE.
- change(xpos) = true - same as above (true is a system variable with a logical value of 1).
- active(xpos-1) = 0 - Error; only variables have attributes. Expressions do not have attributes.

5.5.5. Conditional statements

Format:

```
if <cond> then
  statements
endif
```

OR

```
if <cond> then
  statements
else
  statements
endif
```

Description:

The condition (<cond>) is evaluated and converted into logical type. If the result is TRUE (1) only the statements between the 'then' and the 'endif' or between the 'then' and the 'else' are executed. Otherwise, no statement is executed at all, or the statements between the 'else' and the 'endif' are executed. The keyword 'then' is optional and may be omitted. Conditional statements can be nested up to 8 levels.

Example:

```
if direction eq CCW then          ; results in either TRUE or FALSE
  gcode = 3                       ; execute ONLY if TRUE
else
  ;
  gcode = 2                       ; execute ONLY if FALSE
endif
;
```

5.5.6. Subroutine Calls

Format:

```
while <cond>  
  statements  
endw
```

Description:

The condition (<cond>) is evaluated and converted into logical type. If the result is FALSE, then the statements until the 'endw' are skipped (not executed). Otherwise, these statements are executed and the above process is repeated.

While statements can be nested up to 8 levels.

Example:

```
logical integer i  
i = 1  
while i <= 5  
  {nl, 'I=', i }  
  i = i + 1  
endw
```

The above code will generate 5 blocks (lines) with 5 different values (1...5) for i.

5.5.7. Procedure Calls

Format:

call @<proc name>

Description:

This statement suspends the regular execution of the program, executes the statements in the procedure '@<proc name>', then resumes the regular execution.

Example:

```
mode = TRUE - set mode to TRUE
call @stop_tool - execute procedure '@stop_tool'
mode = FALSE - set mode to FALSE
```

5.5.8. Procedure definition

Format:

```
@<proc name>  
  (procedure body)  
endp
```

Description:

The procedure definition is made up of three parts:

1. The name of the procedure
2. The body of the procedure
3. The end of the procedure

Example:

```
@stop_tool  
  (operational command)  
endp
```

5.5.9. Comment

Format:

; any text

Description:

All the text after the ';' till the end of the line is simply ignored.

Example:

```
gcode = 1 ; this is a comment and is ignored.
```

5.5.10. G-code generation

Format:

{ <item1>, <item2>, ... }

Note:

The comma (,) is optional.

Description:

The generate statement is used to generate the G-code. All the items in the statement are evaluated, and then written to the G-code file. Each <item> can be:

- A variable of any kind (e.g. {xpos}).
- An expression of any kind; it should appear between brackets (e.g. {(2*xpos)}).

Each item can optionally have a "display format" to control the way it will be generated. It might be explicitly written in conjunction to the item itself (e.g. {xpos: '5.3'}) or GPPL tries to find it according to the following rules:

(assume that the variable to be generated is 'xxx')

- Search for a system or global string variable named 'xxx_f'; if it exists, use it as the display format.
- Search for a local string variable named 'xxx_f'; if it exists, use it as the display format.
- If the variable is of integer/logical type, use the system variable named 'def_integer_f' as the display format.
- If the variable is of numeric type, use the system variable named 'def_numeric_f' as the display format.

This mechanism gives you full control over the G-code generation format.

Examples:

- { xpos } - use 'xpos_f'
- {(2*xpos)} - use 'def_numeric_f'
- { xpos:'5.3z' } - use this specific format

5.5.10.1. Conditional generation (modality)

GPPL supports conditional generation of items. A "modal group" is a group of items surrounded by square brackets ([...]). For example:

```
{ ['G', gcode, 'X', xpos, ' (FIRST POINT)' ] }
```

GPPL uses a simple rule to decide if the modal group should be generated: If at least ONE item of the modal group is required to be generated - all the group will be generated. An item is required to be generated if both its active and change attributes are TRUE.

The conditional generation is especially designed for modal G-code generation. For example, if a previously generated block was a 'G1' block (in procedure '@line' for example), and the current block is going to be also a 'G1' block, the 'G1' does not have to be generated. The way GPPL handles this is simply by defining the appropriate group as a modal group, as in the following example:

```
{ ['G', gcode] }
```

If the system variable 'gcode' has changed from the previous time it was used it, the string 'G1' will be generated; if has not changed then the string 'G1' will not be generated.

Examples:

- gcode = 1 - if previous value was 1, set change flag to FALSE
- {nb, ['G'gcode], ' X'xpos, ' Y'ypos} - N125 X123.45 Y-12.3 note that 'G1' was not generated.
- {nb, [' X'xpos], [' Y'ypos]} - Assume change(xpos) = FALSE
change(ypos) = TRUE
The generated block will be:
N125 Y-12.3
- {nb, [' X'xpos, ' Y'ypos]} - Assume change(xpos) = TRUE
change(ypos) = TRUE
The generated block will be:
N125 X123.45 Y-12.3

Note:

G-field modality is an important, heavily used feature of the G-code languages. GPPL gives special care for that facility. Consider the following examples:

- (1) N10 G1 X12.3 Y-5.
 N15 X35. Y7.14 ; G-field omitted.
 N20 G0 X0. Y0.
- (2) N10 G1 X12.3 Y-5
 N15 X35. Y7.14 ; Blanks reserve space of G-field for
 N20 G0 X0. Y0. ; readability

The GPPL code used for block N15 in the first example is the following:

```
gcode = 1
{nb, ['G' gcode], ' X' xpos, ' Y' ypos}
```

The GPPL code used for block N15 in the second example is a little bit more complicated:

```
gcode = 1
if change(gcode) then
  {nb, 'G' gcode, ' X' xpos, ' Y' ypos}
else
  {nb, ' ', ' X' xpos, ' Y' ypos} ; reserve space
endif
```

It would be better if we could write the second example in more condensed code (as in the first example). GPPL enables us to do so. If the system variable 'gcode_space' equals TRUE, GPPL will produce the second example's G-code, even if the GPPL generation statement is:

```
{nb, ['G' gcode], ' X' xpos, ' Y' ypos}
```

5.5.10.2. Display format

The display format is a string expression that defines how the item should be printed. It has the following components:

<sign><leading-zeroes><integer> . <fraction><trailing-zeroes><options>

<sign> sign control:

- '+' - Put '+' sign to indicate positive numbers.
- '-' - Put '-' sign to indicate positive numbers.
- none - put sign only for negative numbers.

<leading-zeroes> leading zeroes control:

- ' ' - Precede number with leading blanks (spaces).
- 'z' - Precede number with leading zeroes.
- none - do not append anything.

<integer> integer part control. This component might be either 'M' or 'M/m', where:

- M - number of digits before the decimal point.
- /m - minimum number of digits that must be generated. Default: 1.
- .

<fraction> fraction part control. This component might be either 'N' or 'N/n', where:

- N - number of digits after the decimal point. Default is 0 for integer expressions, 3 for numerics.
- /n - minimum number of digits that must be generated. Default:0

<trailing-zeroes> trailing zeroes control:

- ' ' - append number with trailing blanks (spaces).
- 'z' - append number with trailing zeroes.
- none - do not append anything.

<options>: Miscellaneous formatting options, that must be surrounded by brackets, and may be separated for readability purposes by blanks. The options may be specified either as lower-case letters or as CAPITAL letters. The options are:

- *s - scaling factor. The number to be generated is multiplied by 's' before generation (s stands for any number, including non-integer ones).
- /s - scaling factor. The number to be generated is divided by 's' before generation (s stands for any non-zero number, including non-integer ones).
- d - Delete if zero; if the number to be generated is zero, nothing is generated.
- n - No decimal point; the number is generated without the decimal point.
- p - Conditional decimal point; the number is generated without decimal point if its fraction part is equal to zero.
- i - Invert sign; the sign of the number is inverted, then it is generated.

Examples:

- ◆ {'|', (123.456):'5.2', '|'} ==> |123.46|
 - up to 5 places before the decimal point.
 - up to 2 places after the decimal point
 - number is rounded as necessary.
- ◆ {'|', (123456):'5.2', '|'} ==> |123456.|
 - 5 places before the decimal point are not enough. GPPL will not truncate any significant digits, so the number takes as much places as required.
 - fraction part equal to zero. Note the decimal point (see next example).
- ◆ {'|', (123456):'5.2(p)', '|'} ==> |123456|
 - fraction part equal to zero. 'p' option - remove the decimal point in such a case.

- ◆ {'|, (-12.3):' 5.3z', '|'} ==> | -12.300|
 - This number is preceded by 3 leading blanks (the sign is not counted towards the 5 places limit).
 - 2 trailing zeroes are added.
- ◆ {'|, (-12.3):' 5.3z(n)', '|'} ==> | -12300|
 - 'n' option removes the decimal point.
- ◆ {'|, (1.234):'+5.0(*100)', '|'} ==> |+123.|
 - the number is multiplied by 100 before generation.
 - a sign is always generated.
- ◆ {'|, (-1234):'+5.0(/2.54i)', '|'} ==> |+486.|
 - the number is divided by 2.54 before generation.
 - the 'i' option inverts the number's sign.
 - a sign is always generated.
- ◆ {'|, (12.345):'8/5.7/4', '|'} ==> |00012.3450|
 - at least 5 places before the decimal point are required.
 - at least 4 places after the decimal point are required.
 - leading/trailing zeroes are added.

5.5.11. While Statement

Format:

```
while <cond>  
  statements  
endw
```

Description:

The condition (<cond>) is evaluated and converted into logical type. If the result is FALSE, then the statements until the 'endw' are skipped (not executed). Otherwise, these statements are executed and the above process is repeated.

While statements can be nested up to 8 levels.

Example:

```
logical integer i  
i = 1  
while i <= 5  
  {nl, 'I=', i }  
  i = i + 1  
endw
```

The above code will generate 5 blocks (lines) with 5 different values (1...5) for i.

5.5.12. Trace statement

Format:

trace <procedures>:<trace-level>

Description:

GPP will produce trace information while generating G-Code. The trace information will be generated only for those procedures which are defined in <procedures> list (see examples below). The <trace-level> determines how much information will be generated. The <trace-level> value must be in the range of 1 to 5, where 5 gives the maximum trace information available.

This information is of great interest in the development phase of a new post-processor; it can significantly shorten the time required to develop such a post-processor.

Examples:

trace "all": 1

- All procedures will be traced with the minimum trace information available.

trace "@line,@arc":5

- only procedures '@line' and '@arc' will be traced, generating the maximum available information.

5.6. GPPL Internal functions

GPPL supports a set of internal ("built-in") functions. Use these functions whenever you need. There are four categories of functions: numeric, string, logical and generation functions.

Following is a complete explanation of each function:

5.6.1. Numeric functions

◆ **abs** (number)

Parameters:

number: any numeric expression.

Description:

This function returns the absolute value of the number.

Examples:

```
abs(12.3-8) ==> 4.3  
abs(8-12.3) ==> 4.3
```

◆ **acos** (number)

Parameters:

number: any numeric expression in the range [-1, 1].

Description:

This function returns the arc-cosine of the number. The angle returned is in degrees in the range [0, 180].

Examples:

```
acos(1) ==> 0  
acos(-1) ==> 180
```

◆ **ang** (x, y)

Parameters:

x - X-axis coordinate of a point.
y - Y-axis coordinate of a point.

(x and y both should not be zero.)

Description:

This function returns the angle defined by the point and the positive direction of the X axis. The angle is returned in degrees in the range [-180, 180]. (see 'atan2' function).

Examples:

ang(5,5) ==> 45
ang(-3, 0) ==> 180
ang(5, -5) ==> -45

◆ **asin** (number)

Parameters:

number - any numeric expression in the range [-1, 1].

Description:

This function returns the arc-sine of the number. The angle returned is in degrees in the range [-90, 90].

Examples:

asin(1) ==> 90
asin(-0.5) ==> -30

◆ **atan** (number)**Parameters:**

number - any numeric expression.

Description:

This function returns the arc-tangent of the number. The angle returned is in degrees in the range [-90, 90].

Examples:

```
atan(1)      ==> 45  
atan(-1)     ==> -45
```

◆ **atan2** (y, x)**Parameters:**

y - Y axis coordinate of a point.
x - X axis coordinate of a point.

(y and x should not be both zero)

Description:

This function returns the angle defined by the point and the positive direction of the X axis. The angle is returned in degrees in the range [-180, 180] (see 'ang' function).

Examples:

```
atan2(5,5)   ==> 45  
atan2(-3, 0) ==> -90  
atan2(5, -5) ==> 135
```

◆ **cos** (angle)

Parameters:

angle - any numeric expression (angle is in degrees).

Description:

This function returns the cosine value of the angle.

Examples:

```
cos(90)      ==> 0
cos(810)     ==> 0
cos(540)     ==> -1
cos(acos(1)) ==> 1
```

◆ **dist** (x, y)

Parameters:

x - X axis coordinate of a point.
y - Y axis coordinate of a point.

Description:

This function returns the distance between the point and the origin of the axis system (point (0,0)). Dist(x, y) is equivalent to $\sqrt{x*x + y*y}$.

Examples:

```
dist(3, 4)   ==> 5
dist(-3, 4)  ==> 5
dist(12, -5) ==> 13
```

◆ **exp** (number)**Parameters:**

number - any numeric expression.

Description:

This function returns the value of e (the base of natural logarithms) to the power of number.

Examples:

```
exp(0)      ==> 1
exp(1)      ==> 2.718281828...
exp(-0.5)   ==> 0.606530659...
```

◆ **frac** (number)**Parameters:**

number - any numeric expression.

Description:

This function returns the fraction part of the number.

Examples:

```
frac(12.345) ==> 0.345
frac(-12.345) ==> 0.345
frac(12)      ==> 0
```

◆ **int** (number)**Parameters:**

number - any numeric expression.

Description:

This function returns the integer part of the number. The value is truncated, not rounded.

Examples:

```
int(12.345) ==> 12
int(-12.345) ==> -12
int(frac(12.3)) ==> 0
```

◆ **log** (number)

Parameters:

number - any positive numeric expression.

Description:

This function returns the value of the natural logarithm of the number.

Examples:

log(1) ==> 0
log(2.718281828) ==> 1 (approximately)

◆ **log10** (number)

Parameters:

number - any positive numerical expression.

Description:

This function returns the value of the base 10 logarithm of the number.

Examples:

log10(1) ==> 0
log10(10) ==> 1
log10(1000) ==> 3

◆ **mod** (a, b)

Parameters :

a : any numeric expression.

b : any numeric expression.

Description :

This function returns the remainder R of a / b, such that $a = i * b + R$, where i is an integer number, R has the same sign as a, and $abs(r) < abs(b)$.

Examples :

mod(5, 2) ==> 1
mod(-17, 3) ==> -2
mod(9.2, 4) ==> 1.2

◆ **pow** (b, p)**Parameters:**

b - any numeric expression.
p - any numeric expression.

Note:

if b equals zero, p must be positive;
if b is less than zero, p must be integer.

Description:

This function returns the value of b to the power of p.

Examples:

```
pow(10, 3)    ==> 1000
pow(0, 12)    ==> 0
pow(25, 0.5)  ==> 5
pow(-2, 2)    ==> 4
pow(-2, 3.5) ==> error. see note limitation above.
```

◆ **round** (number, dig)**Parameters:**

number - any numeric expression.
dig - number of digits to round (must be integer).

Description:

This function returns the number rounded to the required number of digits either after or before the decimal point. Note that it is possible to round the number to the left of the decimal point by giving a negative number for dig (see examples below).

Examples:

```
round(12.3456, 1) ==> 12.3
round(12.3456, 3) ==> 12.346
round(12.5, 0)    ==> 13
round(-12.5, 0)   ==> -13
round(123.4, -1)  ==> 120
round(126.4, -1)  ==> 130
round(123, -3)    ==> 0
```

◆ **round2val** (c, d)

Parameters:

c - any numeric expression.
d - any numeric expression.

Description:

This function returns c rounded to smallest multiple of d.

Examples:

```
round2val(730, 360) ==> 720  
round2val(17.5, 2) ==> 18  
round2val(8.5, 2.1) ==> 8.4
```

◆ **sign** (number)**Parameters:**

number - any numeric expression.

Description:

This function returns an integer number, representing the sign of the number. The values returned are:

- 1 - if the number is less than zero.
- 0 - if the number is equal to zero.
- 1 - if the number is greater than zero.

Examples:

```
sign(123.4) ==> 1
sign(-123.4) ==> -1
sign(cos(90)) ==> 0
```

◆ **sin** (angle)**Parameters:**

angle - any numerical expression (angle is in degrees).

Description:

This function returns the sine value of the angle.

Examples:

```
sin(90) ==> 1
sin(990) ==> -1
sin(540) ==> 0
sin(asin(1)) ==> 1
```

◆ **sqrt** (number)**Parameters:**

number - any non-negative numeric expression.

Description:

This function returns the square root of the number. This function is equivalent to 'pow(number, 0.5)'.

Examples:

sqrt(25) ==> 5
sqrt(0) ==> 0
sqrt(-4) ==> error. Only non-negative numbers have real square root.

◆ **square (number)**

Parameters:

number - any numeric expression.

Description:

This function returns the value of the number squared. It is equivalent to 'pow(number, 2)'.
'

Examples:

square(5) ==> 25
square(-5) ==> 25
square(0) ==> 0

◆ **tan (angle)**

Parameters:

angle - any numerical expression (angle is in degrees).

Description:

This function returns the tangent value of the angle.

Examples:

tan(45) ==> 1
tan(90) ==> 1.632455E16 (approximation of infinity)
tan(540) ==> 0

◆ **todeg** (angle)**Parameters:**

angle - any numeric expression (angle is in radians).

Description:

This function returns the angle (which is assumed to be in radians) into its equivalent in degrees.

Examples:

```
todeg(0)      ==> 0
todeg(1)      ==> 57.295779...
todeg(pi)     ==> 180 (note that 'pi' is a system variable with the value 3.1415926...)
```

◆ **torad** (angle)**Parameters:**

angle - any numeric expression (angle is in degrees).

Description:

This function returns the angle (which is assumed to be in degrees) into its equivalent in radians.

Examples:

```
torad(0)      ==> 0
torad(57.295779) ==> 1 (approximately)
torad(180)    ==> 3.1415926...
```

5.6.2. String functions

◆ **date**

Parameters:

none.

Description:

This function returns the string value of the current date in the format 'DD-MMM-YYYY' where DD is the current day, MMM is the current month and YYYY is the current year.

Examples:

```
date ==> "12-Apr-1994"
```

◆ **dup (string, count)**

Parameters:

string - the string expression to be duplicated.
count - any non-negative numeric expression.

Description:

This function returns a string value, consisting of the string 'string' duplicated 'count' times.

Examples

```
dup ('abc', 3) ==> 'abcabcabc'  
dup (date, 2) ==> '26-Nov-198926-Nov-1989'
```

◆ **instr (str, sub_str)**

Parameters:

str - the string expression to be searched.
sub_str - the string expression to be searched for.

Description:

This function searches for the first appearance of the sub_str in str and returns its position from the beginning of str. If sub_str was not found the function returns 0. Note that GPL does distinguish between lower and upper case letters.

Examples:

```
instr('abcd', 'cd')    ==> 3
instr('abcd', 'x')    ==> 0
instr('abcd', 'Cd')   ==> 0 (lower/upper case sensitivity)
instr('abcdabcd', 'cd') ==> 3 (the first occurrence is taken)
```

◆ **left** (str, num)**Parameters:**

str - any string expression.
num - any positive numeric expression.

Description:

This function returns the sub-string which consists of num left characters of str. If str is too short, all str is returned (see function 'right').

Examples:

```
left('abcdef', 4) ==> 'abcd'
left('abcdef', -2) ==> error. (meaningless request)
left('xyz', 6) ==> 'xyz' (there are no 6 left characters).
```

◆ **lower** (string)**Parameters:**

string - any string expression.

Description:

This function returns the same string, where all upper case (CAPITAL) letters (A-Z) are replaced by their lower case equivalent (a-z). Please note that the original string is left unchanged. If you want to change a certain string into lower case you must use an assignment statement (see examples below).

Examples:

```
lower('TITLE') ==> 'title'
lower('Title no. 1') ==> 'title no. 1'
line = lower(line) ==> converts string variable 'line' into lower case.
```

◆ **rev** (string)

Parameters:

string - any string expression.

Description:

This function returns the same string while reversing the order of its characters.

Examples:

```
rev('abcd') ==> 'dcba'  
rev('123') ==> '321'
```

◆ **right** (str, num)

Parameters:

str - any string expression.
num - any positive numeric expression.

Description:

This function returns the sub-string which consists of the num right characters of str. if str is too short, all str is returned (see function 'left').

Examples:

```
right('abcdef', 4) ==> 'cdef'  
right('abcdef', -2) ==> error.(meaningless request)  
right('xyz', 6) ==> 'xyz' (there are no 6 right characters).
```

◆ **space** (len)

Parameters:

len - any non-negative numeric expression.

Description:

This function returns a string of blanks (spaces) of length len. If len is zero, then a null string is returned. This function is equivalent to 'dup(' ', length)'.

Examples:

```
space(5)    ==> '   '
space(0)    ==> "" (the null string)
space(-3)   ==> error
```

◆ **strlen** (string)**Parameters:**

string - any string expression.

Description:

The function returns the length of the string, i.e. the number of characters in the string.

Examples:

```
strlen('abcd') ==> 4
strlen('abcd ') ==> 5 (note the blank at the end of the string)
strlen("") ==> 0 (length of null string is zero)
```

◆ **substr** (string, from, len)**Parameters:**

string - any string expression.
 from - position from beginning of string where sub-string starts. 'from' must be in the range [1, strlen(string)].
 len - length of the sub-string to extract

Description:

This function returns the sub-string of length len starting in the string from position 'from', and with length len. If 'from + len' is greater than strlen(string), the returned sub-string is of length less than len.

Examples:

```
substr('abcdefg',5,2) ==> 'ef'
substr('abcdefg',5,4) ==> 'efg' (no more characters)
substr('abcdefg',9,2) ==> error (9 is not legal position)
```

◆ **time**

Parameters:

none.

Description:

This function returns the string value of the current time in the format 'HH:MM:SS' where HH is the current hour, MM is the current minutes and SS is the current seconds.

Examples:

time ==> 10:17:30 (10 AM, 17 minutes, 30 secs)
time ==> 22:17:30 (10 PM, 17 minutes, 30 secs)

◆ **tonum (string)**

Parameters:

string - any string expression, which holds a legal numerical value (see description below).

Description:

This function converts the number represented in the string from string format to a numeric format. Valid number in the string must have the following format:

[spaces] [sign] [digits] [.digits] [E [sign] digits]

- spaces - blanks (' ').
- sign - none ('+' assumed), '+' or '-'.
- digits - any sequence of decimal digits (0-9).
- .digits - decimal point, followed by any sequence of digits.
- E - beginning of the exponent part.
- sign - none '+' or '-'; the exponent sign.
- digits - exponent value.

The reverse operation of 'tonum' is the function 'tostr'. See its description below.

Examples:

```
tonum(' 12')    ==> 12
tonum('-12.34') ==> -12.34
tonum('12.3e2') ==> 1230
tonum('-12E-1') ==> -1.2
```

◆ **tostr** (number [:format])**Parameters:**

number - any numeric expression.
format - format definition (optional)

Description:

This function converts the number into string notation according to the format specification 'format'. If 'format' is omitted, the string is converted according to a default format. See chapter C.5.10 for a detailed description of 'format' options.

Examples:

```
tostr(12.345)    ==> '12.345'
tostr(12.3456)   ==> '12.346' (default format is '5.3').
tostr(12.3456:'4.0') ==> '12.'
```

◆ **trim** (string)**Parameters:**

string - any string expression.

Description:

This function returns the same string as in the string, but removes all its trailing blanks.

Examples:

```
trim('abcd')    ==> 'abcd' (nothing to trim)
trim('abcd ')   ==> 'abcd'
```

◆ **upper** (string)

Parameters:

string - any string expression.

Description:

This function returns the same string, where all the lower case letters (a-z) are replaced by their upper case (CAPITAL) equivalent (A-Z). Please note that the original string is left unchanged. If you want to change a certain string into upper case, you must use an assignment statement (see examples below).

Examples:

upper('TITLE') ==> 'TITLE'

upper('Title no. 1') ==> 'TITLE NO. 1'

line = upper(line) ==> converts string variable line into upper case.

5.6.3. Logical functions

◆ **active** (variable)

Parameters:

variable - any local/global/system variable.

Description:

This function returns the value TRUE/FALSE (1/0) according to the current status of the active attribute of the variable. The active attribute is set to TRUE in the following cases:

- If the variable can be used in the current procedure.
- When the variable is assigned a value.
- When the user explicitly uses the 'active' attribute assignment statement.

This function is most useful when checking for conditional execution ('if' statement), or when generating modal items (see section C.5.10). See also the description of function 'change' later in this chapter.

Examples:

- active(xpos) - TRUE/FALSE as active flag of xpos
- active(123) - Error. Expressions does not have an active attribute.
- active(xpos)=active(xnext) - Sets the active attribute of variable xpos to be the same as the active attribute of variable xnext.

◆ **change** (variable)

Parameters:

variable - any local/global/system variable.

Description:

This function returns the value TRUE/FALSE (1/0) according to the current status of the change attribute of the variable. The change attribute is set to TRUE in the following cases:

- When the variable is assigned a new value.
- When the user explicitly uses the 'change' attribute assignment statement.

This function is most useful when checking for conditional execution ('if' statement), or when generating modal items (see section C.5.10). See also the description of function 'active' earlier in this chapter.

Examples:

- change(xpos) - TRUE/FALSE as change flag of xpos.
- change(123) - Error. Expressions does not have an change attribute.
- change(xpos) =change(xnext) - Sets the change attribute of variable xpos to be the same as the change attribute of variable

◆ **even (number)**

Parameters:

number - any numeric expression.

Description:

This function is TRUE only if the number is even.

Examples:

- even(8) ==> TRUE
- even(13) ==> FALSE
- even(4.2) ==> FALSE

◆ **odd (number)**

Parameters:

number - any numeric expression.

Description:

This function is TRUE only if the number is odd.

Examples:

- odd(7) ==> TRUE
- odd(14) ==> FALSE
- odd(3.2) ==> FALSE

CHAPTER 6

6. GPPtool System variables

There are three types of system variables that are available for use in the GPP tool procedures that define how each **SolidCAM** tool-path command is translated into G-code (see Chapter 7). These three types of variables are:

1. Variables that are defined in **SolidCAM** and passed to GPPtool.
2. Variables that are defined at the start of the [machine.gpp] file (in the @init_post section).
3. Variables which have special values and cannot be assigned new values.

6.1. Variables that are defined in SolidCAM and passed to GPPtool

- ◆ `gcode` type: integer
The number of the last-developed G-Code.
- ◆ `mcode` type: integer
The number of the last-developed M-Code.
- ◆ `rotated` type: logical
TRUE if the current JOB is a rotated one.
- ◆ `mirrored` type: logical
TRUE if the current JOB is a mirrored one.
- ◆ `prev_command` type: string
The name of the previous **SolidCAM** tool-path command that was executed.
- ◆ `current_command` type: string
The name of the current **SolidCAM** tool-path command that is being executed.

- ◆ next_command type: string

The name of the next **SolidCAM** tool-path command that is going to be executed.

- ◆ first_user_proc type: integer

Number of the first user-defined G-Code procedure. These procedures are normally included in the '@end_of_file' procedure and are appended at the end of the G-Code file that is generated from the translation of the **SolidCAM** tool-path commands.

6.2. Variables that are defined at the start of the [machine.gpp] file

G-Code format

- ◆ `remove_blanks` type: logical
{TRUE/FALSE}
If TRUE, all blanks (spaces) are removed out of the generated G-Code file. This leads to more compact G-code files (in terms of memory), but to a less readable G-code.
- ◆ `gcode_space` type: logical
{TRUE/FALSE}
If TRUE, G-field that was omitted due to modality is filled with spaces to keep the field after in fixed column. See description in chapter 5.5.10.
- ◆ `gcode_f` type: string
Display format for G field.
- ◆ `mcode_f` type: string
Display format for M field.
- ◆ `numeric_def_f` type: string
Default display format for numeric values.
- ◆ `integer_def_f` type: string
Default display format for integer values.
- ◆ `xpos_f` type: string
Display format for X values.
- ◆ `ypos_f` type: string
Display format for Y values.
- ◆ `zpos_f` type: string
Display format for Z values.

◆ apos_f type: string

Display format for angle of 4th axis.

◆ feed_f type: string

Display format for the feed field.

◆ end_block_text type: string

String appended to the end of each block.

Block Numbers

◆ blknum_letter type: string

The character used for the numbering field (usually the letter 'N').

◆ blknum_exist type: logical
{TRUE/FALSE}

Decides whether the G-Code of the machine can have Block numbering or not.

◆ blknum_gen type: logical
{TRUE/FALSE}

Decides whether to generate block numbering (note that the blocks will be numbered ONLY if both 'blknum_exist' and 'blknum_gen' are true.

◆ blknum_f type: string

Display format for Block-Numbering field.

◆ blknum_delta type: integer

Difference between the numbering of two consecutive blocks.

◆ blknum type: integer

Initial value: number of the first block. During execution of the GPP program, this variable holds the number of the next block to be generated.

◆ blknum_max type: integer

The maximum allowed block number.

◆ skipline type: logical
{TRUE/FALSE}

Decide whether or not to generate new line at the start of the next block

It effect if the next block start with 'nb' command and not 'nl' command.

◆ num_user_procs type: integer

Number of user procedures in the G-code. This is defined in '@end_of_file.'

Example:

For detailed explanation of display format options see section 5.5.10 in this manual.

@init_post

```

numeric_def_f = '5.3' ; point will always appear (5. and not 5)
integer_def_f = '5.0(p)'
gcode_f       = '2.0(p)'
mcode_f       = '2.0(p)'
xpos_f        = '5.3(p)' ; point will not always appear (5 and not 5.1)
ypos_f        = '5.3(p)'
zpos_f        = '5.3(p)'
apos_f        = '5.3(p)'
feed_f        = '4.0(p)'
end_block_text = "      ; do not append anything
blknum_letter = 'N' ; use 'N' as block numbering letter
blknum_gen    = FALSE ; do not generate block numbers
blknum_exist  = TRUE  ; blocks might be numbered in this
                ; particular machine
blknum_f      = '5.0(p)' ; use up to 5 digits for block numbers
                ; (no dec point)
blknum        = 5      ; start block numbers from 5
blknum_delta  = 5      ; block will be numbered as 5, 10, ...
blknum_max    = 32000 ; block numbers can not exceed this limit
num_user_procs = 1     ; one user procedure is defined in
                ; '@end_of_file' procedure.
remove_blanks = FALSE  ; leave blanks in generated G-Code file

endp

```

6.3. Variables which have special values

- ◆ **PI** type: numeric
value: 3.1415926535... (15 digits)
The value of the mathematical constant pi.

- ◆ **FALSE** type: logical
value: 0
The logical value of false condition.

- ◆ **TRUE** type: logical
value: 1
The logical value of true condition.

- ◆ **inch_system**
if TRUE, inch units; otherwise mm units.

CHAPTER 7

7. GPPtool commands

7.1. SolidCAM Tool-Path commands

These are the **SolidCAM** internal tool-path commands. All the tool-paths calculated in **SolidCAM**, by the various jobs, are saved in this language. When the user asks for the generation of G-code, then the saved tool-path commands file is translated to a G-code file according to the [MACHINE.MAC] and [MACHINE.GPP] files.

Each tool-path command has parameters that are assigned values by **SolidCAM**; the parameter list is given with the tool-path command description. Some parameters are optional i.e. they are not available for some of the CNC controllers. You can check if a parameter is active or not, for the particular CNC controller you are working with, by using the logical function 'active'.

Upon entry to a procedure, GPPtool assigns values only to the procedure's parameters. GPPtool does not change other system/global variables or parameters of other procedures; they retain their previous value. Therefore, it is possible to use inside the current executed procedure all the parameters of a previously executed procedure.

Note :

Whenever a tool-path command variable has a pre-defined value (e.g. variable `arc_direction` in '@arc' might have CW or CCW), you should test that variable against these constants.

Following is a complete description of all the **SolidCAM** tool-path commands together with their parameters:

@absolute_mode

Parameters:

None

Description:

This command enters the program into absolute coordinate position mode.

Examples:

```
@absolute_mode    ; FANUC
  gcode = 90
  {nb, 'G'gcode}
  skipline = FALSE ; generate next command on same line
endp
```

@arc**Parameters:**

xpos, ypos, zpos	type: numeric	Tool-position at end of arc.
feed	type: numeric	Feed-rate of movement.
xcenter, ycenter	type: numeric	Absolute center of arc.
xcenter_rel, ycenter_rel	type: numeric	Center of arc relative to the start point.
radius	type: numeric	Arc-radius.
start_angle, end_angle	type: numeric	Start and end angles of arc (in degrees).
arc_direction	type: integer	Arc-direction {CW, CCW}.
arc_plane	type: integer	Arc-plane {XY, YZ, ZX}.
arc_size	type: numeric	Arc size (in degrees).
next_direction	type: numeric	Direction of the next XY block (if the next block is not an XY block).
zstart	type: numeric	Z value at start of arc.

Description:

This command generates the arc feed-positioning G-code.

Examples:

```
@arc      ; FANUC
  if arc_direction eq CCW then
    gcode = 3
  else
    ; CW
    gcode = 2
  endif
  {nb, ['G'gcode], [' X'xpos], [' Y'ypos], [' Z'zpos]}
  if arc_size eq 360 then
    {' I'xcenter_rel, ' J'ycenter_rel}
  else
    if arc_size gt 180 then
      radius = -radius
    endif
    {' R'radius}
  endif
  {' F'feed]}
endp
```

```
@arc      ; MAHO-432
  if arc_direction eq CCW then
    gcode = 3
  else
    ; CW
    gcode = 2
  endif
  if machine_plane eq ZX then
    xpos = -xpos
  endif
  {nb, ['G'gcode], [' X'xpos], [' Y'ypos], [' Z'zpos]}
  if arc_size gt 180 then
    {' I'xcenter_rel], [' J'ycenter_rel]}
  else
    {' R'radius}
  endif
  {' F'feed]}
endp
```

@arc4x_cartesian**Parameters:**

xpos, cpos, zpos	type : numeric	Tool-position at end of arc.
feed	type : numeric	Feed-rate of movement.
xcenter, ycenter	type : numeric	Absolute center of arc.
xcenter_rel, ycenter_rel	type : numeric	Center of arc relative to the start point.
radius	type : numeric	Arc-radius.
start_angle, end_angle	type : numeric	Start and end angles of arc (in degrees).
arc_direction	type : integer	Arc-direction {CW, CCW}
arc_plane	type : integer	Arc-plane {XY, YZ, ZX}
arc_size	type : numeric	Arc-size (in degrees).
zstart	type : numeric	Z value at start of arc.
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the arc feed-positioning G-code. End position is defined by cartesian coordinates (xpos, cpos).

@arc4x_polar

Parameters:

xpos, cpos, zpos	type : numeric	Tool-position at end of arc in polar coordinates.
feed	type : numeric	Feed-rate of movement.
xcenter, ycenter	type : numeric	Absoloute center of arc.
xcenter_rel, ycenter_rel	type : numeric	Center of arc relative to the start point.
radius	type : numeric	Arc-radius.
start_angle, end_angle	type : numeric	Start and end angles of arc (in degrees).
arc_direction	type : integer	Arc-direction {CW, CCW}
arc_plane	type : integer	Arc-plane {XY, YZ, ZX}
arc_size	type : numeric	Arc-size (in degrees).
zstart	type : numeric	Z value at start of arc.
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the arc feed-positioning G-code arc in polar coordinates.

@call_prms**Parameters:**

label	type : integer	Name (number) of subroutine to be called.
start_line	type : function	Block number of subroutine start.
end_line	type : function	Block number of subroutine start.
call_prms_num	type : integer	Number of parameters to pass to the subroutine.
call_prms array	type : numeric array	The parameters to pass to the subroutine.

Description:

This command generates a subroutine call with parameters.

@call_proc**Parameters:**

label	type: integer	Name (number) of subroutine to be called.
proc_count	type: integer	Number of times to execute the subroutine.
parm1,parm2,parm3	type: numeric	Procedure parameters (optional).
message	type: string	Optional text that describes the subroutine.
start_line	type: function	Block number of subroutine start.
end_line	type: function	Block number of subroutine end.

For first level procedure the following parameters can be used:

Parameters for all job type

job_type	type: string	Can be one of the following: { "pocket", "profile", "drill", "slot", "trans_surf", "3-d model", "multy-drill", "3-d drill", "3-d engraving", "5x_advanced", "5x_face", "5x_pen_trace", "5x_port_mach", "5x_swarf", "5x_turbine", "5x_2_5_axis", "5x_3axis", "5x_4axis" "turn", "drill", "thread", "groove" "profile", "4_axis", "constant_angle", "macro", "variable_angle" } };
job_machine_type	type: integer	The type of the current job {milling, turning, wire_cut}.

Parameters for Miling jobs

mac_number	type: integer	Machine home to be used.
rot_axis_type	type: integer	The type of a milling job when using simultaneous 4th axis. {AXIS4_NONE, AXIS4_FACE, AXIS4_TOP, AXIS4_RADIAL}.
tool_side	type: integer	Side location of the tool in relation to the profile . {tool_side_left, tool_side_right, tool_side_middle}

Parameters for Drill jobs

drill_type	type: integer	Drill type, as defined in 'machine.mac' file.
d_drill_type	type: integer	Internal drill type {D_Drilling, D_F_Drill, D_Peck, D_Tapping, D_Boring, D_R_Boring, D_F_Boring}.

Parameters for Turning jobs

work_type	type: integer	For Grooving job {groove_rough, groove_prof, cut, ang_groove}. For Turning job ROUGH, COPY, PROFILE}. For Threading Job {ONCE, MULTIPLE}.
semi_finish	type: integer	TRUE if semi finish process is needed.
finish	type: logical	TRUE if finish process is needed.
process_type	type: integer	{LONG, FACE}.
turning_mode	type: integer	for LONG: {INTERNAL,EXTERNAL} for FACE: {BACK, FRONT}.
rough_offset	type: integer	Type of rough offset.
rough_offset_dist, rough_offset_x, rough_offset_z	type: numeric	offsets for ROUGH/COPY process.
semi_offset_x, semi_offset_z	type: numeric	offsets for semi_finish process.F
down_step	type: numeric	distance between two consecutive turning movements.
down_step_type	type: integer	Determines whether the thread should be done in one step or by multi step {DS_VALUE, DS_LIST}.
num_down_steps	type: integer	Number of down steps (if it is a multi step thread).
lead_unit	type: integer	{MM, PITCH_INCH}.
lead	type: numeric	lead of thread (pick to pick).
side_step	type: numeric	Distance between each two groove-cut steps.
min_diameter	type: numeric	Minimal diameter of thread.
start_pnt	type: numeric	Start position point of the job.
end_pnt	type: numeric	End position point of the job.

External cycle parameters

Parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type', 'drill_type', 'groove_type' and 'thread_type' types. For each type

defines the parameters required for it. The value of the parameters of each type set on the current job.

Parameters for Wire_cut jobs

wire_inserted; type: integer 1 if wire inserted, 0 otherwise

Description:

This command generates a subroutine call. Usage of the functions 'start_line' and 'end_line' is limited to generate statements only; for further description of their behaviour see section 5.6.4 ('generation functions').

Examples:

```
@call_proc      ; FANUC
  if active(parm1) then
    gcode = 65      ; generate G65 macro call type
    {nb, 'G'gcode, ' P'label}
    {' A'parm1, [' B'parm2], [' C'parm3]}
  else
    {nb, 'M98 P'label} ; generate normal M98 subroutine call
                        ; type
  endif
  if proc_count gt 1 then
    {' L'proc_count} ; repetition factor
  endif
  {' ('message, ')} ; optional descriptive text
endp
```

@change_ref_point

Parameters:

xhome,yhome,zhome	type: numeric	Position of new home location.
absolute	type: logical	TRUE if the change ref point coordinates are absolute; FALSE if relative.
ref_point_init	type: logical	TRUE if changed back to the original value

Description:

This command generates the G-Code block required to change the HOME location.

Examples:

```
@change_ref_point
; Given in absolute mode
gcode = 92
{nb, 'G'gcode, ' X'xhome, ' Y'yhome, ' Z'zhome }
endp
```

@change_tool**Parameters:**

tool_number	type: integer	Number of tool to be changed to.
tool_diameter	type: numeric	Diameter of tool to be changed to.
tool_length	type: numeric	The length of the tool relative to the length of the first tool (as defined in milling tools table).
corner_radius	type: numeric	The corner radius of the tool.
tool_direction	type: integer	Direction of spin of tool {CW, CCW}.
first_tool	type: logical	TRUE if this is the first tool in the program.
last_tool	type: logical	TRUE if this is the last tool change in the program.
next_tool_number	type: integer	Number of next tool.
next_tool_machine_type	type: integer	Machine type of next tool.
xnext,ynext,znext	type: numeric	Position to move to after tool change.
spin	type: numeric	Spin rate.
tool_id_number	type: integer	Identification number of the tool.
next_tool_id_number	type: integer	Identification number of the next tool.
xtool,ytool,ztool	type: numeric	Coordinates of tool-change position. (Optional).
tool_name	type: string	Name of the tool.
group_tool_name	type: string	Name of the group tool.
holder_name	type: string	Name of the holder of the tool.
group_holder_name	type: string	Name of the group holder of the tool.
tool_description	type: string	Description of the tool.
holder_description	type: string	Description of the holder.
total_tool_length	type: numeric	The length of the tool.

cutting_tool_length	type: numeric	The length of the cutting tool.
work_material	type: string	Kind of the stock material. Used for feed calculation.
tool_material	type: string	Kind of the tool material. Used for feed calculation.
d_offset	type: integer	The d offset address
h_offset	type: integer	The h offset address
hlength	type: numeric	
number_of_jobs_used_tool	type: integer	Number of jobs that use this tool number

Description:

This command generates the G-code required to change the tool.

Examples:

```
@change_tool      ; MAHO-432
  {nb, '(* TOOL 'tool_number' - DIA 'tool_diameter '*')}
  {nb, 'M9'}
  ; The tool is stopped by M6
  {nb, 'T'tool_number, ' M6'}
  xpos = xnext
  ypos = xnext
  zpos = tool_start_plane
  call @rapid_move      ; generate G0 block
  direction = CCW
  call @start_tool      ; generate G4 block
endp
```

@chng_tool_cnext

Parameters:

cnext type : numeric Normalized cpos

Description:

This command normalizes cpos.

@compensation

Parameters:

side	type: integer	Side of compensation {COMP_LEFT, COMP_RIGHT} or no compensation{COMP_OFF}.
offset_number	type: integer	The tool number in tool table (it may be used here or in '@change_tool').
tool_offset offset_radius	type: numeric	The tool offset radius.

Description:

This command decides whether tool-compensation is active and if yes, at what side.

Examples:

```
compensation    ; FANUC , MAHO-432
  if side eq COMP_LEFT then
    gcode = 41
  endif
  if side eq COMP_RIGHT then
    gcode = 42
  endif
  if side eq COMP_OFF then
    gcode = 40
  endif
  {nb, 'G'gcode, ''}
  skipline = FALSE ; next G-Code will be generated on the
                    same block, not on a new one.
endp
```

@def_tool**Parameters:**

tool_number	type: integer	The number of the tool.
tool_offset	type: numeric	The tool radius.
tool_id_number	type: integer	Identification number of the tool.
tool_name	type: string	Name of the tool.
group_tool_name	type: string	Name of the group tool.
holder_name	type: string	Name of the holder of the tool.
group_holder_name	type: string	Name of the group holder of the tool.
tool_description	type: string	Description of the tool.
holder_description	type: string	Description of the holder.
total_tool_length	type: numeric	The length of the tool.
cutting_tool_length	type: numeric	The length of the cutting tool.
work_material	type: string	Kind of the stock material. Used for feed calculation.
tool_material	type: string	Kind of the tool material. Used for feed calculation.
d_offset	type: integer	The d offset address
h_offset	type: integer	The h offset address
hlength	type: numeric	

Description:

This command defines the tool characteristics.

Examples:

```
@def_tool      ; FANUC
  {nb, '(G10 L12 P', (tool_number+50), ' R'tool_offset, ')}
endp
```

@def_turn_tool

Parameters:

tool_number	type: integer	The number of the tool.
tool_id_number	type: integer	Identification number of the tool.
message	type: string	descriptive message for tool change.
tool_direction	type: integer	{CW/CCW}.
tool_mode	type: integer	left/right tool.
tool_origin	type: integer	Tool reference point {T_TANGENT, T_CENTER, T_DEFINE}.
tool_type	type: integer	tool family {EXT_ROUGH, TURN_DRILLING, EXT_THREAD, EXT_GROOVE, EXT_CONTOUR, INT_FACE_BACK, INT_THREAD, INT_GROOVE, INT_CONTOUR, INT_ROUGH}.

Description:

This command defines the tool characteristics.

Examples:

```
@def_turn_tool      ; FANUC
  {nb, '(G10 L12 P', (tool_number+50) ')}
endp
```

@delay**Parameters:**

delay_period type: numeric Delay amount.

Description:

This command generates the G-Code block required to cause a delay.

Examples:

```
@delay      ; FANUC
  gcode = 4
  {nb, 'G'gcode, ' P'delay_period:integer_def_f}
                ; 'delay_period' is numeric, and FANUC
                ; requires it to be generated
                ; without the decimal point.
endp

@delay      ; MAHO-432
  gcode = 4
  {nb, 'G'gcode, ' X'delay_period:'5.1(p)'}
                ; no more than 1 digit after the
                ; (optional) decimal point.
endp
```

@drill

Parameters:

xpos, ypos, zpos	type: numeric	Position of point to be drilled.
drill_type	type: integer	Drill type, as defined in 'machine.mac' file.
d_drill_type	type: integer	Internal drill type {D_Drilling, D_F_Drill, D_Peck, D_Tapping, D_Boring, D_R_Boring, D_F_Boring}.
drill_lower_z	type: numeric	Drill lower-level.
drill_upper_z	type: numeric	Drill upper-level (including the safety distance).
drill_clearance_z	type: numeric	The level the tool rises to when moving from one hole to another.
drill_depth	type: numeric	(drill_upper_z-drill_lower_z).
down_step	type: numeric	Depth of down-step.
num_down_steps	type: integer	Number of down-steps (last step can be shorter than others).
feed	type: numeric	Feed-rate.
spin	type: numeric	Spin rate.

In addition to the above parameters, more parameters may be defined by the user in the 'machine.mac' file. These parameters define the 'drill_type', and for each type, defines the parameters required for it. See examples below.

Examples:

FANUC:

(here is a portion from the FANUC.MAC file):

```

drill_type    = Drilling Y          ; G81
drill_type    = F_Drill  Y Delay    ; G82
drill_type    = Peck    Y Delay    ; G83
drill_type    = Tapping  Y Delay    ; G84
drill_type    = Boring   Y Delay    ; G85
drill_type    = R_Boring Y Delay    ; G86
drill_type    = F_Boring Y Delay    ; G89

```

Seven 'drill_type' are defined here: 'Drilling', 'F_Drill', 'Peck', 'Tapping', 'Boring', 'R_Boring' and 'F_Boring'. Each drill type has its own parameters. For example: if 'drill_type' is 'Peck', then parameter 'Delay' has a valid value.

(here is the GPPL procedure for drilling):

```

@drill
  call @rapid_move    ; generate G0 block to (xpos,ypos,zpos)
  gcode = 98
  {nb, 'G'gcode, ' '}
  if drill_type eq drilling then
    gcode = 81
  endif
  if drill_type eq f_drill then
    gcode = 82
  endif
  if drill_type eq peck then
    gcode = 83
  endif
  if drill_type eq tapping then
    gcode = 84
  endif
  if drill_type eq boring then
    gcode = 85
  endif
  if drill_type eq r_boring then
    gcode = 86
  endif
  if drill_type eq f_boring then
    gcode = 89
  endif
  {'G'gcode, 'Z'drill_lower_z, 'R'drill_upper_z}
  if drill_type eq peck then
    {'Q'down_step}
  endif

  if drill_type eq f_drill or drill_type eq tapping then
    {'P'delay:integer_def_f}
  endif
  {'F'feed}
endp

```

@drill4x_cartesian

Parameters:

xpos, cpos	type : numeric	Coordinate of the drill-point.
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}
first_drill	type : logical	(see @drill_point)

Description:

This command generates the G-Code block required to define a drill-point. Drill position is defined by cartesian coordinates (xpos, cpos).

@drill_point**Parameters:**

first_drill	type: logical	Defines whether this is the first drill-point of a cycle or not. In some machines, the first drill is executed by '@drill'.
xpos, ypos, zpos	type: numeric	Coordinates of the drill-point. Only two of them have to be used according to the machine plane.

Description:

This command generates the G-Code block required to define a drill-point.

Examples:

```
@drill_point      ; FANUC
  if not first_drill then
    {nb, ' ', [' X'xpos], [' Y'ypos], [' Z'zpos]}
  endif
endp
```

```
@drill_point      ; MAHO-432
  gcode = 79
  {nb, 'G'gcode, [' X'xpos], [' Y'ypos], [' Z'zpos]}
endp
```

@drill4x_pnt

Parameters:

xpos, ypos, zpos cpos	type:numeric	Coordinates of the drill-point. Only two of them have to be used according to the machine plane ('cpos' is always used).
first_drill	type:logical	Defines whether this is the 1st drill-point of a cycle or not. In some machines, the 1st drill is executed by '@drill'.

Description:

This command generates the G-code block required to define a drill-point when using simultaneous 4th axis.

Examples:

```
@drill4x_pnt ; FAN_OTC
  if not first_drill then
    {nb,' ',[' X'xpos],[' Z'zpos ],[' C'cpos]}
  endif
endp
```

@drill4x_polar

Parameters:

xpos, cpos	type : numeric	Coordinate of the drill-point.
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}
first_drill	type : logical	(see @drill_point)

Description:

This command generates the G-Code block required to define a drill-point. Drill position is defined by polar coordinates ($xpos * \cos(cpos)$, $xpos * \sin(cpos)$).

@end_drill

Parameters:

None.

Description:

This command generates the G-Code block required to terminate the drill-cycle.

Examples:

```
@end_drill      ; FANUC
  gcode = 80
  {nb, 'G'gcode}
endp
```

```
@end_drill      ; MAHO-432
  ; No end_drill - always active
  ; Note that the procedure MUST present, although it contains
  ; no executable code generation.
endp
```

@end_job_procs

Parameters:

none

Description:

This command generates the required Gcode for end of job procedures.

Example:

```
@end_job_procs ;FANUC  
  {nb, 'M99'}  
endp
```

@end_loop

Parameters:

label	type: integer	Name (number) of the loop.
loop_level	type: integer	Loop nesting.
loop_count	type: integer	Number of times the loop is executed.
start_line	type: function	Block number of loop start.
end_line	type: function	Block number of loop end.

Description:

This command generates the G-code needed to end a loop.

Examples:

```
@end_loop    ; FANUC
  local integer var_num

  var_num = loop_level + 20
  {nb '#', var_num, '= #', var_num, ' + 1'}
  {nb 'END ', loop_level}
endp

@end_loop    ; MAHO
  gcode=14
  loop_count = loop_count - 1
  {nb, 'G'gcode, ' N1='start_line, ' N2='end_line, ' J'loop_count}
endp
```

@end_of_file**Parameters:**

None

Description:

Executed at the end of the G-Code file generation. It may contain user defined procedures. Use the following properties for automatic numbering of these procedures:

- Specify their number in 'num_user_procs' (in '@init_post').
- Use variable 'first_user_proc' as the first procedure number.
- Use 'first_user_proc'+1 for the second one etc...

Examples:

```
@end_of_file      ; FANUC
  label = first_user_proc
  call @proc      ; generate 'Onnnn' block
  {nb, '(------)'}
  {nb, '(- CHANGE TOOL -)'}
  {nb, '(------)'}
  {nb, 'G80 G49 G40 M9'}
  {nb, 'G91 G28 Z0.')}
  call @stop_tool ; generate 'M5' block
  {nb, 'G90 M1'}
  {nb, 'M6'}
  call @end_proc  ; generate 'M99' block
  {nl, '%'}
endp
```

@end_proc

Parameters:

label type: integer Name (number) of the subroutine.
end_of_job type: logical TRUE for end of job procedure.
start_line type: function Block number of subroutine start.
end_line type: function Block number of subroutine end.

Description:

This command generates G-Code for subroutine termination.

Examples:

```
@end_proc        ; FANUC  
  {nb, 'M99'}  
endp
```

```
@end_proc        ; MAHO-432  
                 ; Since block numbers are used to refer to the  
                 ; subroutine, there is no need to generate any  
                 ; G-Code blocks.  
                 ; Note that the procedure MUST be resent, although  
                 ; it contains no executable code generation  
endp
```

@end_program**Parameters:**

xpos, ypos, zpos type: numeric End position of the tool.

Description:

This command is executed at the end of the of the main program.

Examples:

```
@end_program        ; FANUC  
  {nb, 'M9'}  
  call @rapid_move    ; move to (xpos,ypos,zpos)  
  {nb, 'M30'}  
endp
```

@feed_spin

Parameters:

feed_unit type: integer Feed units {MM_MIN / MM_REV}
feed type: numeric Feed rate
spin_unit type: integer Spin units {RPM/CSS}
spin type: numeric Spin rate
spin_direction type: integer Spin direction {CW/CCW}

Description:

This command generates a G-code which defines the feed and spin rates for turning jobs.

Example:

```
@feed_spin
  if feed_unit eq MM_MIN then
    gcode = 94
  else
    gcode = 95
  endif
  {nb, 'G' gcode, [ S'spin ] }
endp
```

@fourth_axis**Parameters:**

angle type: numeric rotation angle (in degrees).
direction_4x type: integer direction of rotation {CW,CCW}.
xnext, ynext type: numeric next (X,Y) position.

Description:

This command generates the G-code block required to control the fourth-axis rotation.

Examples:

```
@fourth_axis     ; FANUC  
  gcode = 0  
  {nb, 'G'gcode, ' A'angle}  
endp
```

```
@fourth_axis     ; MAHO-432  
  gcode = 0  
  {nb, 'G'gcode, ' B'angle}  
endp
```

@groove

Parameters:

work_type	type: integer	{ groove_rough, groove_prof, cut, ang_groove }.
process_type	type: integer	{ LONG, FACE }.
turning_mode	type: integer	for LONG: { INTERNAL, EXTERNAL } for FACE: { BACK, FRONT }.
first_pos_x, first_pos_z	type: numeric	coordinates of first point of geometry.
last_pos_x, last_pos_z	type: numeric	coordinates of last point of geometry.
second_offset	type: integer	offset number of the other side of the tool.
down_step	type: numeric	down step of the groove.
side_step	type: numeric	distance between each two groove-cut steps.
release_dist	type: numeric	safety release distance of the groove-cut.

In addition to the above parameters, more parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type' and for each type defines the parameters required for it. See the example below.

Description:

This command generates a groove cycle block.

Examples:

```
@groove ; FANUC (simplified)

  if process_type eq FACE then
    gcode = 74
  else
    gcode = 75
  endif
  {nb, 'G' gcode, 'R'release_dist}
  {nb, 'G' gcode, 'X'first_pos_x, 'Z'last_pos_z}
  {P'down_step, 'Q'side_step, 'F'feed}
endp
```

Note:

SolidCAM splits a multiple-lines geometry into single line entities.

@home_data**Parameters:**

```
home_number:1
clearance_plane:50.000 tool_start_plane:70.000
work_upper_plane:0.000 zero_plane:-40.000
rotate_angle_x:0.000T rotate_angle_y:0.000T rotate_angle_z:0.000T
rotate_angle_x_dir:cw rotate_angle_y_dir:cw rotate_angle_z_dir:cw
x_angle_const_z:0.000T y_angle_const_z:0.000T dev_angle_z:0.000T
x_angle_const_z_dir:cw y_angle_const_z_dir:cw dev_angle_z_dir:cw
x_angle_const_y:0.000T z_angle_const_y:0.000T dev_angle_y:0.000T
x_angle_const_y_dir:cw z_angle_const_y_dir:cw dev_angle_y_dir:cw
y_angle_const_x:0.000T z_angle_const_x:0.000T dev_angle_x:0.000T
y_angle_const_x_dir:cw z_angle_const_x_dir:cw dev_angle_x_dir:cw
angle_4x_around_x:0.000T angle_4x_around_y:0.000T
angle_4x_around_x_dir:cw angle_4x_around_y_dir:cw
shift_x:0.000T shift_y:0.000T shift_z:0.000T
part_home_number:1 tool_z_level:500.000
tmatrix_I_1:1.000T tmatrix_I_2:0.000T tmatrix_I_3:0.000T tmatrix_I_4:0.000T
tmatrix_I_5:0.000T tmatrix_I_6:1.000T tmatrix_I_7:0.000T tmatrix_I_8:0.000T
tmatrix_I_9:0.000T tmatrix_I_10:0.000T tmatrix_I_11:1.000T tmatrix_I_12:0.000T
. tmatrix_I_13:0.000T tmatrix_I_14:0.000T tmatrix_I_15:0.000T tmatrix_I_16:1.000T
. x = cosy*cosz*x - sinz*cosy*y + siny*z
.. y = (-sinx*siny*cosz + cosx*sinz)*x + (sinx*siny*sinz + cosx*cosz)*y - sinx*cosy*z
z = (cosx*cosz*siny + sinx*sinz)*x + (-sinz*cosx*siny + sinx*cosz)*y - cosx*cosy*z
around Z
x = x*cos(dev_angle) - y*sin(dev_angle)
y = x*sin(dev_angle) + y*cos(dev_angle)
around Y
z = z*cos(dev_angle) - x*sin(dev_angle)
x = z*sin(dev_angle) + x*cos(dev_angle)
around X
y = y*cos(dev_angle) - z*sin(dev_angle)
z = y*sin(dev_angle) + z*cos(dev_angle)
```

This contains the same information as in **@tmatrix** .

This command is used at the end of the program to generate a sub-program for each home; we can later call this sub-program from the main program whenever we have to change the home position. If at some later point, the operator needs to change something, he can easily change only in one place (in the home sub-program).

Description:

@home_data contains information about the part home number, machine home number, planes, angles, shifts and transformation matrix.

@home_data location depends on mac variable home_data_at_start:
if “Y” appears after @def_tool
if “N” appears after @end_program.

Example of Deckel-Maho:

```
@home_data

{nl, '%MM' (6000+part_home_number)}
  {nl, 'N' (6000+part_home_number)}
  {nl, 'N5 (X) E101=' shift_x ' (Y) E102='shift_y ' (Z) E103=' shift_z }
  {' (A) E111=' (-rotate_angle_x) }
  {' (B) E112=' (rotate_angle_y) }
  {' (C) E113='(-rotate_angle_z)}
  {nl}

endp
```

@home_number

Parameters:

home_number type: integer The number of machine home to be activated.

Description:

This command generates the G-Code required to activate a new machine home. A new home can be set in the Job data-screen.

Examples:

```
@home_number ; FANUC
  gcode = 53 + home_number
  {nb, 'G'gcode}
endp
```

```
@home_number ; MAHO-432
  ; Note that the procedure MUST present, although it
  ; contains no executable code generation

endp
```

@init_cpos

Parameters:

prev_cpos	type: numeric	Rotation angle of the last block in the previous job. Used for milling jobs when using simultaneous 4th axis and when the value of the 'init_cpos' parameter in the .MAC file is 'Y'.
best_cpos	type: numeric	The best rotation angle to set the machine before the 1st movement block in a new job. See explanation for 'init_cpos' parameter in the machine parameters (chapter 3).

Description:

This command is used before the 1st movement block in a new milling job when using simultaneous 4th axis. It is used to set a new rotation angle for the machine home. It is used only if the 'init_cpos' parameter in the .MAC file is 'Y'. This parameter is used for avoiding unnecessary rotations of the material.

Examples:

```
@init_cpos ; FAN_0TC
  gcode = 0
  {nb,'G'gcode ' C'prev_cpos}
  gcode = 50
  {nb,'G'gcode ' C'best_cpos}
endp
```

@line

Parameters:

xpos, ypos, zpos	type: numeric	Tool-position at end of movement.
feed	type: numeric	Feed-rate of movement.
next_direction	type: numeric	Direction of the next XY block (if the next block is not an XY block).

Description:

This command generates the line feed-positioning G-code.

Examples:

```
@line      ; FANUC
  gcode = 1
  {nb, ['G'gcode],[ 'X'xpos],[ 'Y'ypos],[ 'Z'zpos],[ 'F'feed]}
endp
```

@line_4x**Parameters:**

xpos, ypos, zpos, cpos	type: numeric	Tool position at the end of the movement.
feed	type: numeric	Feed-rate of movement. If 'cpos' is changed in the block, the feed is in Angle/Min , otherwise it is in MM/Min.
next_direction	type: numeric	Direction of the next XY block (if the next block is not an XY block).

Description:

This command generates the line feed-positioning G-code when using simultaneous 4th axis.

Examples:

```
@line_4x ; FAN_0TC  
gcode = 1  
{nb, ['G'gcode],[' X'xpos],[' Z'zpos ],[' C'cpos],[' F'feed]}  
endp
```

@line_5x

Parameters:

xpos, ypos, zpos, apos, bpos	type: numeric	Tool position at the end of the movement.
feed	type: numeric	Feed-rate of movement.
next_direction	type: numeric	Direction of the next XY block (if the next block is not an XY block).

Description:

This command generates the line feed-positioning G-code when using 5 axis.

Examples:

```
@line_5x  
gcode = 1  
{nb, '[G'gcode],[ ' X'xpos] [ ' Y'ypos] [ ' Z'zpos] [ ' A'apos] [ ' B'bpos][ ' F'feed]}
```

endp

@line4x_cartesian**Parameters:**

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
feed	type : numeric	Feed-rate of movement.
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the line feed-positioning G-Code. End position is defined by cartesian coordinates.

@line4x_dir

Parameters:

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
feed	type : numeric	Feed-rate of movement.
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the line feed-positioining G-Code when using simultaneous 4th axis. 'dir4x' parameter sets the machine rotate direction.

This command is generated if the MAC file parameter set_dir = Y.

@line4x_polar**Parameters:**

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
feed	type : numeric	Feed-rate of movement.
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the line feed-positioning G-Code. End position is defined by polar coordinates.

@line_on

Parameters:

xpos, ypos, zpos	type: numeric	Tool-position at end of movement.
feed	type: numeric	Feed-rate of movement.
before	type: logical	The move ends a radius before or after the point specified.

Description:

A line feed-positioning command which is shorter or longer in a tool radius length. This command is generated if the MAC file parameter comp_x_start = Y.

Example:

```
@line_on      ; MAHO-432
  if before then
    gcode = 43
  else
    gcode = 44
  endif
  {nb, 'G'gcode, ''}
  skipline = FALSE
  call @line
endp
```

@loop**Parameters:**

label	type: integer	Name (number) of the loop.
loop_level	type: integer	Loop nesting.
loop_count	type: integer	Number of times the loop is executed.
start_line	type: function	Block number of loop start.
end_line	type: function	Block number of loop end.

Description:

This command generates the G-code needed to start a loop.

Examples:

```
@loop      ; FANUC
  local integer var_num

  var_num = loop_level + 20
  {nb, '#var_num, ' = 0'}
  {nb, 'WHILE [#var_num, ' LT ',loop_count,'] DO ',loop_level}
endp
```

@m_feed_spin

Parameters:

feed_unit type: integer Feed units {MM_MIN / MM_REV}
feed type: numeric Feed rate
spin_unit type: integer Spin units {RPM/CSS}
spin type: numeric Spin rate
spin_direction type: integer Spin direction {CW/CCW}

Description:

This command generates a G-code which defines the feed and spin rates for milling.

Example:

```
@feed_spin
  if feed_unit eq MM_MIN then
    gcode = 94
  else
    gcode = 95
  endif
  {nb, 'G' gcode, [ S'spin ] }
endp
```

@machine_plane**Parameters:**

machine_plane type: integer {XY, YZ, ZX}

Description:

This command defines the machining plane. This procedure is called only once, because the machine plane is fixed during program generation.

Examples:

```
@machine_plane    ;FANUC
  if machine_plane eq XY
    gcode = 17
  endif
  if machine_plane eq YZ
    gcode = 18
  endif
  if machine_plane eq ZX
    gcode = 19
  endif
  {nb, 'G'gcode}
endp
```

@message

Description:

This command generate the G-Code block required to print a message.

Parameters:

message type: string The message to be printed.

Examples:

```
@message            ; FANUC, MAHO-432  
  {nb, '(', message, ')'  
endp
```

@mirror**Parameters:**

mirror_type type: integer {MIRROR_OFF, MIRROR_X, MIRROR_Y,
MIRROR_XY}

Description:

This command generates the G-Code block required to activate the mirror facility.

Examples:

```
@mirror      ; FANUC
if mirror_type eq MIRROR_OFF then
  {nb, 'G50.1 X0 Y0'}
else
  {nb, 'G51.1 '}
  if mirror_type eq MIRROR_X then
    {'X1 Y0'}
  endif
  if mirror_type eq MIRROR_Y then
    {'X0 Y1'}
  endif
  if mirror_type eq MIRROR_XY then
    {'X1 Y1'}
  endif
endif
endp
```

```
@mirror      ; MAHO-432
if mirror_type eq MIRROR_OFF then
  gcode = 72
else
  gcode = 73
endif
{nb, 'G'gcode}

if mirror_type eq MIRROR_X then
  {' X1'}
endif
if mirror_type eq MIRROR_Y then
  {' Y1'}
endif
if mirror_type eq MIRROR_XY then
  {' X1 Y1'}
endif
endp
```

@move4x_cartesian

Parameters:

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the rapid-positioning G-Code.
End position is defined by cartesian coordinates.

@move4x_dir**Parameters:**

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the rapid-positioning G-Code when using simultaneous 4th axis. 'dir4x' parameter sets the machine rotate direction.

This command is generated if the MAC file parameter set_dir = Y.

@move4x_polar

Parameters:

xpos, ypos, zpos	type : numeric	Tool-position at end of movement.
cpos	type : numeric	C positioning of the machine
next_direction	type : numeric	Direction of next XY block (if the next block is not an XY block).
dc	type : numeric	Delta of cpos
dir4x	type : integer	Direction of C axis {CW, CCW}

Description:

This command generates the rapid-positioning G-Code.
End position is defined by polar coordinates.

@move_4x**Parameters:**

xpos, ypos, zpos, cpos type:numeric Tool position at end of movement.

Description:

This command generates the rapid-positioning G-code when using simultaneous 4th axis.

Examples:

```
@move_4x ; FAN_0TC
gcode = 0
{nb,'G'gcode,[' X'xpos],[' Z'zpos ],[' C'cpos]}
endp
```

@move_5x

Parameters:

xpos, ypos, zpos, apos, bpos type:numeric Tool position at end of movement.

Description:

This command generates the rapid-positioning G-code when using 5 axis.

Examples:

```
@move_5x
gcode = 0
{nb,'G'gcode,[' X'xpos],[' Z'zpos ],[' A'apos] ,[' B'bpos]}
endp
```

@proc**Parameters:**

label type: integer Name (number) of the subroutine.

start_line type: function Block number of subroutine start.

end_line type: function Block number of subroutine end.

All the parameters that available in @start_of_job available also in @proc.

Description:

This command generates the subroutine header. Usage of the functions 'start_line' and 'end_line' is limited to generate statements only; for further description of their behaviour see section 5.6.4 ('generation functions').

Examples:

```
@proc            ; FANUC  
  {nl, 'O'label}   ; Note the use of 'nl' and not 'nb'. This  
                   ; block should not be numbered.  
endp
```


@relative_mode**Parameters:**

None

Description:

This command enters the program into relative coordinate positioning mode.

Examples:

```
@relative_mode      ; FANUC,  
  gcode = 91  
  {nb, 'G'gcode}  
  skipline = FALSE ; generate next command on same line  
endp
```

@rotary_info

Parameters:

- rot_axis_type type: integer The type of a milling job when using simultaneous 4th axis. {`AXIS4_NONE`, `AXIS4_FACE`, `AXIS4_TOP`, `AXIS4_RADIAL`}.
- radial_diameter type: numeric Material diameter for TOP or WRAP milling jobs, when using simultaneous 4th axis.
- center_pos type: numeric Level of material rotation center position. This parameter is relevant for TOP or WRAP milling jobs, when using simultaneous 4th axis.
- rot_axis_coord type: integer The type of rot_axis_coordinate (`axis4_split`, `axis4_polar`, `axis4_cartesian`)

Description:

This command is used at the beginning of every job when using simultaneous 4th axis. This function provides the parameters of the job type, and of the material diameter and rotation center position.

Examples:

```
@rotary_info        ; FAN_0TC
  if rot_axis_type eq axis4_radial
    machine_plane = YZ
  endif
endp
```

@rotate**Description:**

This command generates the G-code block required to start/cancel rotation.

Parameters:

rotate_cancel type: logical TRUE if rotation should be cancelled.

angle type: numeric Rotation angle.

Examples:

```
@rotate ; FANUC
; Not exist in FANUC 6M
if rotate_cancel then
  gcode = 69
  {nb, 'G'gcode}
else
  gcode = 68
  {nb, 'G'gcode, ' X0 Y0 G91 R'angle}
  {nb, 'G90'}
endif
endp
```

```
@rotate ; MAHO-432
gcode = 92
{nb, 'G'gcode, ' B4='angle}
endp
```

@start_of_file

Parameters:

home_number	type: integer	Machine home to be used.
g_file_name	type: string	The G-Code file name to be generated.
full_g_file_name	type: string	The full G-Code file name to be generated.
part_name	type: string	The name of the part.

Description:

This command is executed once at the start of the G-Code file (before the tool definition section).

Example 1:

```
@start_of_file ;FANUC
  {'%'}
  {nl, 'O'program_number, '(', g_file_name, ')'}
  if rotate_used then
    gcode = 69
    {nb, 'G'gcode}
  endif
  if mirror_used then
    {nb, 'G50.1 X0 Y0'}
  endif
  {nb, '(SUBROUTINES: O'first_proc_number}
  { ' .. O' last_proc_number, ')'}
endp
```

Example 2:

Get the file size:

First Step:

set 'print_file_size' = 1 in '@init_post'.

Second Step:

Refer to the value of variable 'file_size' as following

```
@start_of_file
  {nl, '$$file_size$$'}
endp
```

@start_of_job**Parameters:**

job_name	type: string	The name of the job.
job_type	type: string	Can be one of the following: { "pocket", "profile", "drill", "slot", "trans_surf", "3-d model", "multy-drill", "3-d drill", "3-d engraving", "5x_advanced", "5x_face", "5x_pen_trace", "5x_port_mach", "5x_swarf", "5x_turbine", "5x_2_5_axis", "5x_3axis", "5x_4axis" "turn", "drill", "thread", "groove" "profile", "4_axis", "constant_angle", "macro", "variable_angle" }
job_machine_type	type: integer	The type of the current job { milling, turning, wire_cut }.
drill_type	type: integer	The type of drill operation. { drilling, peck, boring etc.. }
prev_job_mac_type	type: integer	The type of the previous job { milling, turning, wire_cut }.
depth	type: numeric	Job depth. Valid for 2.5D milling jobs.
down_step	type: numeric	Depth of down step used in the job.
safety	type: numeric	Safety distance of the job
compensation	type: logical	Defines whether machine tool radius compensation is used in the job.
max_spin	type: numeric	Maximum spin rate allowed for the machine used for CSS spin rate type in turning.
msg	type: numeric	The comment given for the job.
tool_number	type: integer	The number of tool used by the job.
label	type: integer	Name (number) of the subroutine.
start_line	type: function	Block number of subroutine start.
end_line	type: function	Block number of subroutine end.
finish_feed	type: numeric	Finish feed rate
feed_rate	type: numeric	Feed rate
spin_unit	type: integer	Spin units {RPM/CSS}
spin_rate	type: numeric	Spin rate
finish_spin	type: numeric	Finish spin rate

Parameters for Milling jobs

Job_clearance_plane	type: numeric	Z of clearance plane.
Job_upper_plane	type : numeric	Z of upper plane.
machine_plane	type: integer	{XY, YZ, ZX} The job working plane. Valid for milling jobs.
tool_length	type: numeric	The length of the tool used by the job.
z_feed	type: numeric	Feed rate in Z direction

Parameters for Profile job

cutting_diameter	type : numeric	Cutting diameter of drill tool for chamfer
tool_side	type: integer	Side location of the tool in relation to the profile. {tool_side_left,tool_side_right,tool_side_middle}
wall_offset	type: numeric	Wall offsets of the profile job.
clear_offset	type: numeric	Clear offsets of the job
clear_type	type: integer	Can be one of the following: { "for_and_back" "forward" }
depth_type	type: integer	Can be one of the following: { "constant_depth" }, {"slot_with_section" , "defined_depth" }
profile_app_type	type: integer	Can be one of the following: { "normal" , "arc" , "tangent" , "direct" , "point" }

Parameters for Pocket job

wall_offset	type: numeric	Wall offsets of the Pocket Job
floor_offset	type: numeric	Floor offsets of the Pocket Job
island_offset	type: numeric	Island offsets of the Pocket Job
pocket_app_type	type: integer	Can be one of the following: { "direct_approach", "vertical_approach" ,"diagonal_approach" ,"helical_approach", "linear_approach" }

Parameters for 4_Axis job

cartesian_mode	type: integer	1 for Cartesian mode
polar_mode	type: integer	1 for Polar mode

Parameters for '3-d model' job

msc_rough	type: integer	Can be one of the following: {none, hatch, contour, plunging}
msc_semi_finish, msc_finish	type: integer	Can be one of the following:

{none, linear, offset_cutting, spiral,
circular_pocket, constant_z, pencil}

Surface_offset	type: numeric	Surface offsets
semi_finish_surface_offset	type: numeric	Surface offsets for semi_finish process
semi_finish_wall_offset	type: numeric	Wall offsets for semi_finish process
semi_finish_floor_offset	type: numeric	Floor offsets for semi_finish process
msc_rough_down_step	type: numeric	Depth of down step used in rough process
msc_semi_finish_down_step	type : numeric	Depth of down step used in semi_finish process
msc_finish_down_step	type: numeric	Depth of down step used in finish process
job_lower_plane	type : numeric	Z of lower plane

Parameters for Turning jobs

use_cycle	type: logical	Defines wheather machine turning cycle is used in the job.
-----------	---------------	---------------------------------------------------------------

Description:

This command generates G-code at the beginning of the job. Normally, no code is required here. But if 'gen_procs' equals FALSE (meaning that machine does not have procedures) GPPtool executes the command '@start_of_job', but does not execute the command '@proc'.

Examples:

```
@start_of_job

{nb,('job_name,')}
if msg ne "
  {nb,('msg,')}
endif
if G64 ne 0
  {nb,'G64'}
endif
{nb, 'D'tool_number}
{' G'gcode}
if job_type ne 'drill'
  {' G40 G90 '}
  skipline = false
else
  skipline = true
endif
endp
```

@start_program

Parameters:

xpos, ypos, zpos type: numeric Start position of the tool

Description:

This command is executed once at the start of the main program.

Examples:

```
@start_program ;FANUC
; after tools definition
{nb, 'G80 G49 G40'}
call @home_number ; see procedure below
endp
@home_number ; set current home
gcode = 53 + home_number ; 'home_number' retains its
; previous value assigned in
; '@start_of_file'
{nb, 'G'gcode}
endp
```

@start_tool**Parameters:**

tool_direction type:integer Direction of tool's rotation {CW, CCW}.

spin type: numeric Spin rate.

Description:

This command generates the G-Code block required to start the tool rotation.

Examples:

```
@start_tool      ; FANUC
  if direction eq CCW then
    mcode = 3
  else            ; CW
    mcode = 4
  endif
  {'S'spin:integer_def_f, 'M'mcode} ; 'spin' is numeric, and
                                     ; FANUC requires it to be
                                     ; generated without the
                                     ; decimal point.

endp

@start_tool      ; MAHO-432
  ; start tool and coolant together
  if direction eq CCW then
    mcode = 13
  else            ; CW
    mcode = 14
  endif
  {'S'spin:integer_def_f, 'M'mcode} ; see note for FANUC above

endp
```

@stop_tool

Parameters:

None

Description:

This command generates the G-Code required to stop the tool rotation.

Examples:

```
@stop_tool      ; FANUC, MAHO-432  
  {'M5'}        ; no 'nb' is required. 'M5' appended  
                ; to the previous line.  
endp
```

@thread**Parameters:**

work_type	type: integer	{ONCE, MULTIPLE}.
process_type	type: integer	{LONG, FACE}.
turning_mode	type: integer	for LONG: {INTERNAL,EXTERNAL} for FACE: {BACK, FRONT}.
is_line	type: logical	TRUE if geometry is single line.
num_points	type: integer	number of geometry points.
first_pos_x, first_pos_z	type: numeric	coordinate of first point of geometry.
last_pos_x, last_pos_z	type: numeric	coordinate of last point of geometry.
depth	type: numeric	full depth of thread.
down_step	type: numeric	down step for threading.
down_step_type	type: integer	Determines whether the thread should be done in one step or by multi step {DS_VALUE, DS_LIST}.
num_down_steps	type: integer	Number of down steps (if it is a multi step thread).
safety	type: numeric	distance to keep between geometry and final movement.
lead_unit	type: integer	{MM, PITCH_INCH}.
lead	type: numeric	lead of thread (pick to pick).
label	type: integer	Name (number) of geometry procedures.
start_line	type: function	Block number of geometry subroutine start.
end_line	type: function	Block number of geometry subroutine end.

In addition to the above parameters, more parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type' and for each type defines the parameters required for it. See the example below.

Description:

This command generates a thread cycle block.

Examples:

```
@thread ; FANUC (simplified)
  if work_type eq MULTIPLE then
    gcode = 76
    {nb,'G'gcode,'P'no_last_cut, phase,tool_alfa, 'R'last_cut}
    {nb,'G'gcode, 'X'last_pos_x, 'Z'last_pos_z, 'P'depth}
    { 'Q'down_step, 'F'lead}
  else
    gcode = 92
    {nb, 'G'code, 'X'last_pos_x, 'Z'last_pos_z, 'F'feed}
  endif
endp
```

@tmatrix

Description:

@tmatrix contains information about the part home number, angles, shifts and transformation matrix for multi-sided milling. In this command you can find all the needed information to develop GPP for the various 4 and 5 axis machines.

@tmatrix appears:

- 1) Before: @home_number.
- 2) After: @job_info.

Parameters:

```
rotate_angle_x:0.000T rotate_angle_y:90.000T rotate_angle_z:0.000T
rotate_angle_x_dir:cw rotate_angle_y_dir:cw rotate_angle_z_dir:cw
x_angle_const_z:0.000T y_angle_const_z:90.000T dev_angle_z:0.000T
x_angle_const_z_dir:cw y_angle_const_z_dir:cw dev_angle_z_dir:cw
x_angle_const_y:-90.000T z_angle_const_y:-90.000T dev_angle_y:-90.000T
x_angle_const_y_dir:ccw z_angle_const_y_dir:ccw dev_angle_y_dir:ccw
y_angle_const_x:-90.000T z_angle_const_x:-180.000T dev_angle_x:-180.000T
y_angle_const_x_dir:ccw z_angle_const_x_dir:ccw dev_angle_x_dir:ccw
angle_4x_around_x:0.000T angle_4x_around_y:0.000T
angle_4x_around_x_dir:cw angle_4x_around_y_dir:cw
shift_x:120.000T shift_y:0.000T shift_z:-40.000T
part_home_number:4 tool_z_level:500.000
tmatrix_l_1:0.000T tmatrix_l_2:0.000T tmatrix_l_3:-1.000T tmatrix_l_4:-40.000T
tmatrix_l_5:0.000T tmatrix_l_6:1.000T tmatrix_l_7:0.000T tmatrix_l_8:0.000T
tmatrix_l_9:1.000T tmatrix_l_10:0.000T tmatrix_l_11:0.000T tmatrix_l_12:-120.000T
tmatrix_l_13:0.000T tmatrix_l_14:0.000T tmatrix_l_15:0.000T tmatrix_l_16:1.000T
x = cosy*cosz*x - sinz*cosy*y + siny*z
y = (-sinx*siny*cosz + cosx*sinz)*x + (sinx*siny*sinz + cosx*cosz)*y - sinx*cosy*z
z = (cosx*cosz*siny + sinx*sinz)*x + (-sinz*cosx*siny + sinx*cosz)*y - cosx*cosy*z
around Z
x = x*cos(dev_angle) - y*sin(dev_angle)
y = x*sin(dev_angle) + y*cos(dev_angle)
around Y
z = z*cos(dev_angle) - x*sin(dev_angle)
x = z*sin(dev_angle) + x*cos(dev_angle)
around X
y = y*cos(dev_angle) - z*sin(dev_angle)
z = y*sin(dev_angle) + z*cos(dev_angle)
```

Explanation of @tmatrix variables:

```
> rotate_angle_x:0.000T rotate_angle_y:0.000T rotate_angle_z:0.000T  
..> rotate_angle_x_dir:cw rotate_angle_y_dir:cw rotate_angle_z_dir:cw
```

Use this set of parameters to get the 3 rotation angles A, B, C that the controller needs for 5-axis machines (deckel with shop mill controller with tilt and rotting table):

A = rotate_angle_x
B = rotate_angle_y
C = rotate_angle_z

The angles are calculated in the following order:

1. Rotation around Z.
2. Rotation around Y.
3. Rotation around X.

This is the base information of the rotation and it is the same value you can see in the home data screen and the rotation angles of the drawing for the relevant part home number.

rotate_angle_x_dir = cw / ccw
rotate_angle_y_dir = cw / ccw
rotate_angle_z_dir = cw / ccw

This defines the shortest rotation direction to the next home (angle). It is needed for OKUMA (M15, M16).

```
..> x_angle_const_z:0.000T y_angle_const_z:0.000T dev_angle_z:0.000T  
..> x_angle_const_z_dir:cw y_angle_const_z_dir:cw dev_angle_z_dir:cw
```

For 5-axis machines that the controller can get 2 rotation angles A, B:

A = x_angle_const_z
B = y_angle_const_z

The angles are calculated in the following order:

- Rotation around Y.
- Rotation around X.

With 2 rotation angles the system can calculate only one position to get the defined plane in the correct angles. The position of X-Y in the screen of the system is not the position of the

coordinates X – Y on the machine. This was made in order to transfer the program to different type of 5_axis controllers. To compensate this error we have the parameter:

dev_angle_z

In this parameter we will find the rotation angle (G69 in fanuc G11 R in okuma) we have to set by rotation command or calculate in the GPP the new values for X and Y coordinates.

```
x_angle_const_z_dir = cw / ccw
y_angle_const_z_dir = cw / ccw
dev_angle_z_dir = cw / ccw
```

This defines the shortest rotation direction to the next home (angle). It is needed for OKUMA (M15,M16).

EXAMPLE:

```
HIDNHAIN 426
1020 'CYCL DEF 10.0 ROTATION
1021 'CYCL DEF 10.1 ROT+Q56 ; Q56 is the dev_angle_z
```

In machines that do not have the rotate GCODE you can use this calculation @
@calc_rotate_x_y

```
local numeric x y s_x s_y
local logical save_ch_x save_ch_y

; for @line @rapid_move @arc
; Saving the xpos and ypos value to local parameters
s_x = xpos
s_y = ypos

;rotating the poit X Y to the new position.
x = xpos*cos(dev_angle_x) - ypos*sin(dev_angle_x)
y = xpos*sin(dev_angle_x) + ypos*cos(dev_angle_x)

; setting the rotated value back to XPOS and YPOS
xpos = x
ypos = y

; set the correct change bit to xpos and ypos
if s_x <> xpos
change(xpos) = true
else
change(xpos) = false
```

```
endif

if s_y <> ypos
    change(ypos) = true
else
    change(ypos) = false
endif

; for @arc

;Calculating the rotated position to xcenter and ycenter
x = xcenter*cos(dev_angle_x) - ycenter*sin(dev_angle_x)
y = xcenter*sin(dev_angle_x) + ycenter*cos(dev_angle_x)

; setting the rotated value back to XCENTR and YCENTER
ycenter = y
xcenter = x

endp
```

In the @line @rapid_move @arc, the following lines have to be added:

```
if change(xpos) eq false and change(ypos) eq false

else
    call @calc_rotate_x_y
endif
```

```
..> x_angle_const_y:0.000T z_angle_const_y:0.000T dev_angle_y:0.000T
..> x_angle_const_y_dir:cw z_angle_const_y_dir:cw dev_angle_y_dir:cw
```

Use this set of parameters for 5 axis machines that the controller can get 2 rotation angles A, C

A = x_angle_const_y
C = z_angle_const_y

The angles are calculated in the following order:

- Rotation around Z.
- Rotation around X.

All other parameters are the same as above.

..> part_home_number:1

The defined number in the job dialog screen.

tool_z_level:250.000

You can use this parameter to move the Z level in order to avoid crashes during the rotation. This value is relative to the home position. Normally it is better to move to the reference point.

**..> tmatrix_I_1:1.000T tmatrix_I_2:0.000T tmatrix_I_3:0.000T tmatrix_I_4:0.000T
..> tmatrix_I_5:0.000T tmatrix_I_6:1.000T tmatrix_I_7:0.000T tmatrix_I_8:0.000T
..> tmatrix_I_9:0.000T tmatrix_I_10:0.000T tmatrix_I_11:1.000T tmatrix_I_12:0.000T
..> tmatrix_I_13:0.000T tmatrix_I_14:0.000T tmatrix_I_15:0.000T tmatrix_I_16:1.000T**

If the prepared rotation values are not good you can use this rotation matrix to make your own position calculations.

**..> x = cosy*cosz*x - sinz*cosy*y + siny*z
..> y = (-sinx*siny*cosz + cosx*sinz)*x + (sinx*siny*sinz + cosx*cosz)*y - sinx*cosy*z
..> z = (cosx*cosz*siny + sinx*sinz)*x + (-sinz*cosx*siny + sinx*cosz)*y - cosx*cosy*z**

You have to use this formula to rotate a point around the 3 rotation axis.

**..> around Z
..> x = x*cos(dev_angle) - y*sin(dev_angle)
..> y = x*sin(dev_angle) + y*cos(dev_angle)**

Use this formula to rotate a point around Z to get to the new X and Y coordinates.

**..> around Y
..> z = z*cos(dev_angle) - x*sin(dev_angle)
..> x = z*sin(dev_angle) + x*cos(dev_angle)**

Use this formula to rotate a point around Y to get to the new Z and X coordinates.

**..> around X
..> y = y*cos(dev_angle) - z*sin(dev_angle)
..> z = y*sin(dev_angle) + z*cos(dev_angle)**

Use this formula to rotate a point around X to get to the new Y and Z coordinates.

@tool_path_info

Parameters:

tool_path_type
approach_type.

The following value types are available for **tool_path_type**:

- start_approach
- end_approach
- start_retreat
- end_retreat
- start_finish
- end_finish

The following value types are available for **approach_type**:

- None
- vertical_approach
- diagonal_approach
- helical_approach
- linear_approach

In order to create parametric Gcode, this additional information about tool_path_type and approach_type is needed in order to eliminate some lines and to create the parametric Gcode. For example:

Example 1

Checking if the job is 2D job (not trans_surf and not 3-D model) and the tool_path_type is 'start_approach':

```
if job_type <> '3-d model' and job_type <> 'trans_surf'  
  if tool_path_type eq 'start_approach'  
    {nb,'G0 G80 G90 G40 G54.1P#3 B#2'}  
    {nb,'S#19 M3'}  
    {nb,'M8'}  
    flag_zpos = 1  
    flag_g90_g91 = 0  
  endif
```

Example 2

Checking if the tool_path_type is 'start_approach' and approach_type is either diagonal, helical or linear; we should then create G91 with the original movement from the Pcode.

```
if tool_path_type eq 'start_approach'  
  if approach_type eq 'diagonal_approach' or approach_type eq 'helical_approach'
```

```

                                or approach_type eq 'linear_approach'
        flag_g90_g91 = 2
    endif
endif

```

Example 3

Checking if tool_path_type is 'end_approach' and if we are under incremental movement; if yes we call the absolute mode G90.

```

if tool_path_type eq 'end_approach'
    if flag_g90_g91 eq 7
        flag_g90_g91 = 1
        call @absolute_mode
        skipline = FALSE
    endif
    flag_g90_g91 = 1
endif

```

Example 4

Checking if tool_path_type is 'start_retreat' and we are under G91; if yes, we generate G90.

```

if tool_path_type eq 'start_retreat'
    if flag_g90_g91 eq 91
        flag_g90_g91 = 1
        call @absolute_mode
        flag_g90_g91 = 0
    endif
    flag_g90_g91 = 0
    flag_zpos = 4
endif

```

Example 5

Checking if tool_path_type is 'end_retreat', Z is already at the clearance plane and the job is not trans_surf or 3-d model; we calculate the next Z level.

```

if tool_path_type eq 'end_retreat'
    if flag_zpos eq 5
        if job_type <> '3-d model' and job_type <> 'trans_surf'
            {nb, '#26 = #26 - #5 '}
        endif
        flag_zpos = 0
    endif
endif
endif
endp

```

@turn_change_tool**Parameters:**

tool_number	type: integer	Number of tool to be changed to.
tool_offset_long	type: integer	Number of tool in machine tool offset table.
first_tool	type: logical	TRUE if this is the first tool in the program.
last_tool	type: logical	TRUE if this is the last tool in the program.
next_tool_number	type: integer	Number of next tool.
xnext, znext	type: numeric	Position to move to after tool change.
spin	type: numeric	Spin rate.
spin_unit	type: integer	Spin units {RPM/CSS}.
spin_direction	type: integer	Spin direction {CW, CCW}.
tool_id_number	type: integer	Identification number of the tool.
next_tool_id_number	type: integer	Identification number of the next tool.
spin_limit	type: numeric	Machine spin limit.
xtool, ztool	type: numeric	Coordinates of tool_change position (optional).
message	type: string	descriptive message for tool change.
tool_direction	type: integer	{CW/CCW}.
tool_mode	type: integer	left/right tool.
tool_origin	type: integer	Tool reference point {T_TANGENT, T_CENTER, T_DEFINE}.
tool_type	type: integer	tool family {EXT_ROUGH, TURN_DRILLING, EXT_THREAD, EXT_GROOVE, EXT_CONTOUR, INT_FACE_BACK, INT_THREAD, INT_GROOVE, INT_CONTOUR, INT_ROUGH}.
number_of_jobs_used_tool	type: integer	Number of jobs that use this tool number
next_tool_machine_type	type: integer	Machine type of next tool.

tool_A	type: numeric	tool parameter (in tool table).
tool_B	type: numeric	tool parameter (in tool table).
tool_C	type: numeric	tool parameter (in tool table).
tool_D	type: numeric	tool parameter (in tool table).
tool_D1	type: numeric	tool parameter (in tool table).
tool_D2	type: numeric	tool parameter (in tool table).
tool_E	type: numeric	tool parameter (in tool table).
tool_F	type: numeric	tool parameter (in tool table).
tool_H	type: numeric	tool parameter (in tool table).
tool_K	type: numeric	tool parameter (in tool table).
tool_ALFA	type: numeric	tool parameter (in tool table).
tool_BETA	type: numeric	tool parameter (in tool table).
tool_G	type: numeric	tool parameter (in tool table).
tool_RADIUS_ALFA	type: numeric	tool parameter (in tool table).
tool_RADIUS_BETA	type: numeric	tool parameter (in tool table).

Description:

This command generates the G-Code required to change the tool.

Examples:

```
@turn_change_tool ;OKUMA (simplified)
{nb, 'GO', 'X'xtool, 'Y'ytool}
{nb, 'T'tool_number, tool_number, tool_number, 'M8'}
if spin_unit eq MM_MIN
  {nb, 'G97'}
else
  {nb, 'G96'}
endif
{'G1', 'X'xnext, 'Y'ynext}
endp
```

@turn_drill**Parameters:**

drill_type	type: integer	Drill type, as defined in [machine.mac] file.
d_drill_type	type: integer	Internal drill type {D_Drilling, D_F_Drill, D_Peck, D_Tapping, D_Boring, D_R_Boring, D_F_Boring}.
drill_upper_z	type: numeric	start position of the drill.
drill_lower_z	type: numeric	end position of the drill.
drill_depth	type: numeric	(drill_upper_z - drill_lower_z).
num_down_step	type: integer	Number of down steps (last step can be shorter than others).
safety	type: integer	distance between the tool and material before/after drilling.
down_step	type: numeric	Depth of down step.

In addition to the above parameters, more parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type' and for each type defines the parameters required for it. See the example below.

Description:

This command generates a turning drill cycle block.

Example:

```
@turn_drill ; FANUC (simplified)
  gcode = 74
  if drill_type eq drilling
    r = 0
    q = drill_lower_z
  endif
  if drill_type eq peck
    q = down_step
  endif
  {nb, 'G' gcode, 'R'r}
  {nb, 'G' gcode, 'Z' drill_lower_z, 'Q'q, 'F'feed}
endp
```

Note:

'r' and 'q' values are defined in the job screen.

@turn_endproc**Parameters:**

label type: integer Name (number) of the turn subroutine.
end_of_job type: logical TRUE for end of job procedure.
start_line type: function Block number of subroutine start.
end_line type: function Block number of subroutine end.

Description:

This command generates G-Code for turning subroutine termination.

Examples:

```
@turn_endproc ; OKUMA  
  gcode = 80  
  {nb, 'G' gcode}  
endp
```

@turn_opt_parms

Parameters:

None. Parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type' and for each type defines the parameters required for it.

Description:

This command gives you access to the user_defined parameters from the job screen. Usually there is no need to use this command for cycle generation.

@turn_proc**Parameters:**

process_type type: integer Turning process type {LONG/FACE}.
label type: integer Name (number) of the turn subroutine.
start_line type: function Block number of subroutine start.
end_line type: function Block number of subroutine end.

Description:

This command generates the subroutine header for turning geometry description. Usage of the functions 'start_line' and 'end_line' is limited to the generation of statements only; for further information, see section 5.6.4. ('generation functions').

Examples:

```
@turn-proc ;OKUMA  
  {nl, 'NLP' label}  
  if process_type eq LONG  
    gcode = 81  
  else  
    gcode = 82  
  endif  
  {'G' gcode}  
endp
```

@turning**Parameters:**

work_type	type: integer	{ROUGH, COPY, PROFILE}.
process_type	type: integer	{LONG, FACE}.
turning_mode	type: integer	for LONG: {INTERNAL, EXTERNAL} for FACE: {BACK, FRONT}.
semi_finish	type: integer	TRUE if semi finish process is needed.
finish	type: logical	TRUE if finish process is needed.
is_line	type: logical	TRUE if geometry is a single line.
num_points	type: integer	number of points in geometry.
rough_offset_x, rough_offset_z	type: numeric	offsets for ROUGH/COPY process.
semi_offset_x, semi_offset_z	type: numeric	offsets for semi_finish process.
first_pos_x, first_pos_z	type: numeric	coordinates of first point of geometry
last_pos_x, last_pos_z	type: numeric	coordinates of last point of geometry.
down_step	type: numeric	distance between two consecutive turning movements.
safety	type: numeric	distance to keep between the geometry and the final movement.
label	type: integer	Name (number) of geometry procedures.
start_line	type: function	Block number of geometry subroutine start.
end_line	type: function	Block number of geometry subroutine end.

In addition to the above parameters, more parameters may be defined by the user in the [machine.mac] file. These parameters define the 'turn_type' and for each type defines the parameters required for it. See the example below.

Description:

This command generates a turning cycle block.

Examples:

```
@turning ; OKUMA (simplified)
  if work_type eq ROUGH then
    {nb, 'G85', 'NLP'label}
    {'D' (z*down_step), 'U'rough_offset_x, 'W'rough_offset_z}
    {'F' feed}
  endif
  if work_type eq COPY then
    {nb, 'G86', 'NLP'label}
    {'D' (z*down_step), 'U'rough_offset_x, 'W'rough_offset_z}
    {'F' feed}
  endif
  if semi_finish then
    {nb, 'G87', 'NLP'label, 'U' semi_offset_x, 'W'semi_offset_z}
  endif
  if finish then
    {nb, 'G87', 'NLP'label}
  endif
endif
```

@wc_arc**Parameters:**

xpos, ypos, zpos	type : numeric	Tool-position at end of arc.
feed	type : numeric	Feed-rate of movement.
xcenter, ycenter	type : numeric	Absolute center of arc.
xcenter_rel, ycenter_rel	type : numeric	Center of arc relative to the start point.
radius	type : numeric	Arc-radius.
start_angle, end_angle	type : numeric	Start and end angles of arc (in degrees).
const_angle	type : numeric	Inclination angle
arc_direction	type : integer	Arc-direction {CW, CCW}
arc_plane	type : integer	Arc-plane {XY, YZ, ZX}
arc_size	type : numeric	Arc-size (in degrees).
next_const_angle	type : numeric	next const angle (in degrees).

Description:

This command generates the arc feed-positioning G-code for EDM.

Examples:

```
@wc_arc
if arc_direction eq CCW then
  gcode = 3
else ; CW
  gcode = 2
endif
if change(gcode) then
  {NB' G'gcode}
else
  {NB}
endif
if arc_size le 360
  {' X'xpos] [' Y'ypos], ' I'xcenter_rel, ' J'ycenter_rel }
else
  if arc_size eq 360 then
    {' I'xcenter_rel, ' J'ycenter_rel}
  endif
endif
endp
```

@wc_chng_condition**Parameters:**

```

chng_cond_num type : integer Number of chng_cond parameters
chng_cond     type : string array
                chng_cond<<1,i>> - name
                chng_cond<<2,i>> - type
                    {INTEGER, NUMERIC, STRING}
                chng_cond<<3,i>> - value

```

Description:

This command sets EMD conditions.

Examples:

```

@wc_chng_condition
  local integer i, a
  local numeric d
  ;
  ;1 = name
  ;2 = type (numeric , integer , string)
  ;3 = value

  i = 1
  while i <= chng_cond_num
    if change(chng_cond<<3,i>>) then
      if chng_cond<<2,i>> eq "NUMERIC"
        d = tonum(chng_cond<<3,i>>)
        ; {nb, ' numeric = ' d:'5.3(*1000p)'}
      endif
      if chng_cond<<2,i>> eq "INTEGER"
        a = tonum(chng_cond<<3,i>>)
        ; {nb, ' integer = ' a:'5.0(n)'}
      endif
      gcode = 10
      {nb,'G'gcode ' L10 R'i ' P'chng_cond<<3,i>>}
    endif
    i = i + 1
  endw
endp

```

@wc_chng_e_group

Parameters:

e_group_name	type : string	Name of conditions group.
e_group_changed	type : logical	Indicates change of 'e_group_name'.

Description:

This command sets EMD group of conditions.

Examples:

```
@wc_chng_e_group
; e_group_name:"
if e_group_changed then
;   {nb,'E'e_group_name ,' D'offset_radius}
   {nb,'E'e_group_name}
endif
endp
```

@wc_cut_wire

Parameters:

None

Description:

Cut EDM wire.

Examples:

```
@wc_cut_wire
  {nb,'M7'}
endp
```

@wc_finish_info

Parameters:

tool_diameter	type : numeric	Diameter of wire.
feed	type : numeric	Feed rate.
zpos	type : numeric	Z positioning.
angle	type : numeric	Inclination angle
e_group_name	type : string	Name of conditions group.

Description:

This command defines cut conditions of finish process.

Examples:

```
@wc_finish_info
  local numeric aa1

; tool_diameter:0.200 feed:3.000 zpos:32.000 angle:0.000
; e_group_name:'727'
{nb}
if save_z1 <> zpos
  {nb,'Z1 = 'zpos}
  save_z1 = zpos
endif

if save_z2 <> program_z2
  {nb,'Z2 = 'PROGRAM_Z2}
  save_z2 = program_z2
endif
if save_z5 <> zero_plane
  {nb,'Z5 = 'zero_plane}
  save_z5 = zero_plane
endif
aa1 = tonum(e_group_name)
{nb,'E'aa1:integer_def_f}
{nb,'H'tool_number ' = 'tool_diameter }
{nb 'H'tool_number ' F'feed}
if parm_angle <> 0
  if save_angle <> angle
    {nb,'H100 = 'angle}
    save_angle = angle
  endif
endif
endp
```

@wc_g49_arc

Parameters:

None

Description:

This command defines that the next arc will be in ISO form (not CONIC).

Examples:

```
@wc_g49_arc  
  {nb, 'G49'}  
endp
```

@wc_insert_wire

Parameters:

None

Description:

Insert EDM wire.

Examples:

```
@wc_insert_wire
  {'M6'}
  {nb, 'M17'}
endp
```

@wc_line**Parameters:**

xpos, ypos	type : numeric	Tool-position of lower guide at end of movement.
upos, vpos	type : numeric	Tool-position of upper guide at end of movement.
feed	type : numeric	Feed-rate of movement.
zero_plane	type : numeric	Top Z of material
upper_plane	type : numeric	Bottom Z of material
const_angle	type : numeric	Inclination angle
upos_inc	type : numeric	For future use.
vpos_inc	type : numeric	For future use.
u_angle	type : numeric	Wire Inclination angle in U direction.
v_angle	type : numeric	Wire Inclination angle in V direction

If appears before @wc_angle_status has new parameter next_const_angle.

Description:

This command generates the line feed-positioning G-Code for EDM.

Examples:

```
@wc_line
{nb}
gcode = 1
if change(gcode) then
  {' G'gcode}
else
  {' '}
endif
{' X'xpos}, {' Y'ypos}, {' U'upos}, {' V'vpos}}
endp
```

@wc_move

Parameters:

xpos, ypos type : numeric Tool-position of lower guide
at end of movement.
upos, vpos type : numeric Tool-position of upper guide
at end of movement.
zero_plane type : numeric Top Z of material
upper_plane type : numeric Bottom Z of material
upos_inc type : numeric For future use.
vpos_inc type : numeric For future use.
u_angle type : numeric Wire Inclination angle in U direction.
v_angle type : numeric Wire Inclination angle in V direction

Description:

This command generates the rapid-positioning G-Code for EDM.

Examples:

```
@wc_move
{nb}
gcode = 0
if change(gcode) then
  {'G'gcode}
else
  {' '}
endif
{' X'xpos}, {' Y'ypos}
if job_type ne 'profile' then
  {' U'upos}, {' V'vpos}
endif
endp
```

@wc_program_plane

Parameters:

program_plane type : numeric Z of program plane.

Description:

This command defines Z of program plane.

7.2. User-Defined Commands

These are commands that are defined by the user and that can be activated by calls from the **SolidCAM**-generated tool-path commands. Following are 2 examples of such user-defined commands:

@start_tool1

This command starts the tool rotation. It is called from @change_tool.

Parameters

tool_direction type: integer Direction of rotation of the tool.

spin type: numeric Spin rate.

Examples:

```
@start_tool1 ; FANUC
  if direction = CCW then
    mcode = 3;
  else ; CW
    mcode = 4;
  endif
  {'S'spin:'5.0(p)', 'M'mcode}
endp
```

```
@start_tool1 ; MAHO-432
; start tool and coolant together
  if direction = CCW then
    mcode = 13
  else ; CW
    mcode = 14
  endif
  {'S'spin:'5.0(p)', 'M'mcode}
endp
```

@stop_tool1

This command stops the tool rotation;it is called from @change_tool.

Examples:

```
@stop_tool1    ; FANUC, MAHO-432  
  {'M5'}  
endp
```

```
@change_tool  
  call @start_tool1  
  call @stop_tool1  
endp
```

APPENDIX A

A.1. Example 1: User-defined Pre and Post-processor for a Fanuc controller

A.1.1 Pre-Processor file: [fanuc.mac]

```
;FANUC
```

```
@pre_processor
```

```
;  
;Internal parms  
;
```

```
machine_type      = MILLING  
post_processor    = fanuc  
doc_processor     = fanuc  
gpp_file_ext     = TAP  
mac_holder       = holder  
tool_table_name  = table  
max_g_name_length = 0  
max_tool_numbers = 1000  
default_lang     = DEFAULT
```

```
;  
;Machine Initialize  
;
```

```
machine_plane     = XY  
z_with_xy        = Y  
mac_axes         = XYZ  
num_axes         = 4  
num_simult_axes  = 4  
abs_coord       = N  
rotate          = Y  
mirror          = Y  
variables       = Y  
loops          = Y  
_4th_axes_around = X  
first_rotation_angle = Z  
_5th_axes_around =
```

APPENDIX A

```
;  
;Program numbers  
;  
prog_num_min      = 5000  
prog_num_max      = 8999  
prog_num_dflt     = 5000  
get_prog_num      = Y  
proc_num_min      = 1  
proc_num_max      = 8999  
proc_num_dflt     = 1  
get_proc_num      = Y  
  
;  
;Procedures control  
;  
full_gcode        = N  
gen_procs         = N  
drill_proc        = N  
turn_proc         = Y  
thread_proc       = Y  
gen_internal_proc = N  
turn_common_proc  = Y  
gen_1_line_proc   = Y  
optimize_jobs_loop = Y  
G_in_2_cols       = Y  
seq_sub_number    = N  
loop_exist        = Y  
same_sub_numbers  = N  
init_var_after_split = Y  
  
;  
;Home  
;  
num_homes         = 6  
dflt_home         = 1  
get_job_home      = N  
abs_zero_chng     = Y  
home_data_at_start = N  
  
;  
;Positioning  
;  
dflt_start        = 0.0000 200.0000 100.0000, 0.0000 7.8740 3.9370  
dflt_end          = 0.0000 200.0000 0.0000, 0.0000 7.8740 0.0000  
set_xy_chng       = N N  
set_z_chng        = Y N  
dflt_tool_chng    = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000  
  
;
```

;Compensation

;

comp_exist = Y
comp_arc_arc = Y
comp_arc_line = Y
comp_line_line = Y
next_angle = N
comp_x_start = N
comp_by_arcs = N
chnge_tool_table = N
look_forward = 2
delta_for_TOOL_H = 50
comp_by_zero_tool = N

;

;Arc definitions

;

arc_exist = Y
arc_3d = Y
arc_quadrants = N
arc_gt_180 = Y
arc_max_chord = 30.0000, 1.1811
arc_max_angle = 10.0000
arc_max_radius = 2000.0000, 78.7402
arc_min_length = 0.0000, 0.0000
arc_zx_yz = N

;

;Epsilon values

;

eps_angle = 0.0020, 0.0001
eps_line = 0.0010, 0.0000
zero_value = 0.0010, 0.0000
movement_precision = 0.0010, 0.0000
min_delt_arc_rad = 0.0100, 0.0004
safety_dist = 2.0000, 0.0787

;

;Feed-Spin

;

rapid_feed = 5000.0000, 196.8503
max_spin = 6000.0000
max_feed = 6000.0000, 236.2205
spin_direction = CW

;

;Timing

;

APPENDIX A

```
time_factor          = 1.0000
block_time           = 0.2000
change_tool_time     = 15.0000

;
;Part options
;

options              = COLENT INTEGER

;
;Job options
;

job_opt_type         = OPT1 Y DELY FEAD

;
;Drill cycles
;

drill_type           = Drilling Drilling Y
drill_type           = F_Drill F_Drill Y Delay
drill_type           = Peck Peck Y Delay
drill_type           = Tapping Tapping Y
drill_type           = Boring Boring Y Delay
drill_type           = R_Boring R_Boring Y Delay
drill_type           = F_Boring F_Boring Y Delay

;
;Fourth axis
;

indexial_4th_axis    = N
indexial_increment   = 0.0000, 0.0000
init_cpos            = Y
polar_4x             = N
cartez_4x            = N
set_dir              = N
fourth_axis_letter   = C

;
;Sim Five axis
;

kinematic_type       = HEAD_HEAD
spindle_direction    = 0.0000 0.0000 1.0000
rotate_axis_dir1     = 0.0000 0.0000 -1.0000
rotate_axis_dir2     = 0.0000 -1.0000 0.0000
rot_axis_base_pnt1   = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
rot_axis_base_pnt2   = 0.0000 0.0000 0.0000, 0.0000 0.0000 0.0000
abs_machine_coord    = N
pivot_length         = 0.0000, 0.0000
```

```
first_rot_axis_name      = C
second_rot_axis_name    = B
machine_simulation      = HeadHead
rot_axis_min_limit0     = -100000.0000
rot_axis_min_limit1     = -100000.0000
rot_axis_max_limit0     = 100000.0000
rot_axis_max_limit1     = 100000.0000
auto_angle_pair         = Y
other_angle_pair        = Y
angle_change_limit      = 30.0000
interplat_angle_step    = 3.0000
interplat_for_dist      = N
interplat_distance      = 5.0000, 0.1969
retract_distance        = 100.0000, 3.9370
center_rot_mac_num      = 20
min_inverse_feed        = 100.0000, 3.9370
enable_mx_edit          = N

;
endp
```

A.1.2. Post-Processor file: [fanuc.gpp]

```
; FANUC
; -----
;
;
@init_post
  global string tool_diameter_f

  num_user_procs = 1

  numeric_def_f = '5.3'
  integer_def_f = '5.0(p)'
  gcode_f      = '2.0(p)'
  mcode_f      = '2.0(p)'
  xpos_f       = '5.3'
  ypos_f       = '5.3'
  zpos_f       = '5.3'
  feed_f       = '4.3(p)'
  tool_diameter_f = '5.3/1'
  blknum_f     = '5.0(p)'
; blknum_gen   = FALSE
  blknum_exist = TRUE
  blknum       = 5
  blknum_delta = 5
  blknum_max   = 32000
endp

;-----

@start_of_file
; before tools definition
{'%' }
{nl, 'O'program_number, '(', g_file_name, ')'}
if rotate_used then
  gcode = 69
  {nb, 'G'gcode}
endif
if mirror_used then
  {nb, 'G50.1 X0 Y0'}
endif
{nb, '(SUBROUTINES: O'first_proc_number, '.. O'last_proc_number, ')'}
endp

;-----

@start_program
; after tools definition
{nb, 'G80 G49 G40'}
call @home_number
endp

;-----
```

```
@end_program
  {nb, 'M9'}
  call @rapid_move
  {nb, 'M30'}
endp

;-----

@end_of_file
label = first_user_proc
call @proc
  {nb, '(------)'}
  {nb, '(- CHANGE TOOL -)'}
  {nb, '(------)'}
  {nb, 'G80 G49 G40 M9'}
  {nb, 'G91 G28 Z0.')}
call @stop_tool
  {nb, 'G90 M1'}
  {nb, 'M6'}
call @end_proc
  {nl, '%'}
endp

;-----

@relative_mode
gcode = 91
  {nb, 'G'gcode, ''}
skipline = FALSE
endp

;-----

@absolute_mode
gcode = 90
  {nb, 'G'gcode, ''}
skipline = FALSE
endp

;-----
```

```
@machine_plane
  if machine_plane eq XY
    gcode = 17
  endif
  if machine_plane eq YZ
    gcode = 18
  endif
  if machine_plane eq ZX
    gcode = 19
  endif
  {nb, 'G'gcode}
endp
```

```
;-----
```

```
@call_proc
  if active(parm1) then
    gcode = 65
    {nb, 'G'gcode, ' P'label}
    {' A'parm1, [' B'parm2], [' C'parm3]}
  else
    {nb, 'M98 P'label}
  endif
  if proc_count gt 1 then
    {' L'proc_count}
  endif
  {' ('message, ')}
endp
```

```
;-----
```

```
@proc
  {nl, 'O'label}
endp
```

```
;-----
```

```
@end_proc
  {nb, 'M99'}
endp
```

```
;-----
```

```
@loop
  local integer var_num

  var_num = loop_level + 20
  {nb, '#', var_num, '= 0'}
  {nb, 'WHILE ['#', var_num, ' LT ', loop_count, '] DO ', loop_level}
endp
```

```
;-----
```

```
@end_loop
  local integer var_num

  var_num = loop_level + 20
  {nb '#', var_num, '= #', var_num, '+ 1'}
  {nb 'END ', loop_level}
endp

;-----

@def_tool
  {nb, '(G10 L12 P', (tool_number+50), ' R'tool_offset, ')'}
endp

;-----

@rapid_move
  {nb}
  gcode = 0
  if change(gcode) then
    {'G'gcode}
  else
    {' '}
  endif
  {' X'xpos, [' Y'ypos], [' Z'zpos]}
endp

;-----

@line
  {nb}
  gcode = 1
  if change(gcode) then
    {'G'gcode}
  else
    {' '}
  endif
  {' X'xpos, [' Y'ypos], [' Z'zpos], [' F'feed]}
endp

;-----
```

@arc

```
if arc_direction eq CCW then
  gcode = 3
else ; CW
  gcode = 2
endif
if change(gcode) then
  {'G'gcode}
else
  {' '}
endif
{[' X'xpos] [' Y'ypos] [' Z'zpos]}
```

```
if arc_size eq 360 then
else
  if arc_size >= 180 then
    radius = -radius
  endif
  {' R'radius}
endif
```

```
{[' F'feed]}
endp
```

;------

@compensation

```
if side eq COMP_LEFT then
  gcode = 41
endif
if side eq COMP_RIGHT then
  gcode = 42
endif
if side eq COMP_OFF then
  gcode = 40
endif
{nb, 'G'gcode, ' '}
skipline = FALSE
```

endp

;------

@delay

```
gcode = 4
{nb 'G'gcode, ' P'delay_period:integer_def_f}
endp
```

;------

```
@change_ref_point
; Given in absolute mode
gcode = 92
{nb, 'G'gcode, ' X'xhome, ' Y'yhome, ' Z'zhome}
endp
```

```
;-----
```

```
@home_number
gcode = 53 + home_number
{nb, 'G'gcode}
endp
```

```
;-----
```

```
@rotate
; Not exist in FANUC 6M
if rotate_cancel then
gcode = 69
{nb, 'G'gcode}
else
gcode = 68
{nb, 'G'gcode, ' X0 Y0 G91 R'angle}
{nb, 'G90'}
endif
endp
```

```
;-----
```

```
@fourth_axis
gcode = 0
{nb, 'G'gcode, ' A'angle}
endp
```

```
;-----
```

```
@change_tool
  local logical save_blknum_gen

  {nb, '(*TOOL 'tool_number, ' - DIA 'tool_diameter, '*')}
  {nb, 'T'tool_number}
  label = first_user_proc
  call @call_simple_proc
  save_blknum_gen = blknum_gen
  blknum_gen = true ; gen block number for next block
  gcode = 43
  {nb, 'G'gcode, ' H'tool_number, ' D'(tool_number+50), ' '}
  blknum_gen = save_blknum_gen
  xpos = xnext
  ypos = ynext
  zpos = znext
  skipline = FALSE
  call @rapid_move
  tool_direction = CCW
  call @start_tool
  {nb, 'M8'}
endp
```

```
;-----
```

```
@message
  {nb, '(', message, ')'}
endp
```

```
;-----
```

```
@drill
  call @rapid_move

  gcode = 98
  {nb, 'G'gcode, ' '}
  if drill_type eq drilling then
    gcode = 81
  endif
  if drill_type eq f_drill then
    gcode = 82
  endif
  if drill_type eq peck then
    gcode = 83
  endif
  if drill_type eq tapping then
    gcode = 84
  endif
  if drill_type eq boring then
    gcode = 85
  endif
endif
```

```
if drill_type eq r_boring then
  gcode = 86
endif
if drill_type eq f_boring then
  gcode = 89
endif
{'G'gcode, 'Z'drill_lower_z, 'R'drill_upper_z}
if drill_type eq peck then
  {'Q'down_step}
endif
if drill_type eq f_drill or drill_type eq tapping then
  {'P'delay:integer_def_f}
endif
{'F'feed}

endp

;-----

@drill_point
if not first_drill then
  {nb, ' ', ['X'xpos], ['Y'ypos], ['Z'zpos]}
endif
endp

;-----

@mirror
if mirror_type eq MIRROR_OFF then
  {nb, 'G50.1 X0 Y0'}
else
  {nb, 'G51.1 '}
  if mirror_type eq MIRROR_X then
    {'X1 Y0'}
  endif
  if mirror_type eq MIRROR_Y then
    {'X0 Y1'}
  endif
  if mirror_type eq MIRROR_XY then
    {'X1 Y1'}
  endif
endif
endp

;-----

@end_drill
gcode = 80
{nb, 'G'gcode}
endp

;-----
```

```
@halt_program
  {'M0'}
endp

;-----

@start_of_job
  ; NOP
endp

;-----

@end_of_job
  ; NOP
endp

; =====
; USER DEFINED PROCEDURES
; =====

@call_simple_proc
  active(message) = FALSE
  active(parm1)   = FALSE
  active(parm2)   = FALSE
  active(parm3)   = FALSE
  proc_count      = 1
  call @call_proc
endp

;-----

@start_tool
  if tool_direction eq CW then
    mcode = 4
  else      ; CCW
    mcode = 3
  endif
  {'S'spin:integer_def_f, 'M'mcode}
endp

;-----

@stop_tool
  {'M5'}
endp

-
```

A.2. Example 2: User-defined Pre and Post-processor for mill-Turn with XZYCB axis controller

A.2.1. Pre-Processor file: [Integrex-e-410h.mac] ; Integrex-e-410h

```

@pre_processor
;Internal parms
  gpp_file_ext      = EIA
  machine_type      = MILL_&_TURN
  post_processor    = Integrex-e-410h

_5th_axes_around = z
home_data_at_start = y
seq_sub_number = y

;Machine Initialize
machine_plane      = XY
z_with_xy          = Y
mac_axes           = XYZCB
num_axes           = 5
num_simult_axes    = 4
abs_coord          = N
rotate             = Y
mirror             = Y
variables          = Y
loops              = Y

;Program numbers
prog_num_min       = 0001
prog_num_max       = 8999
prog_num_dflt      = 5000
get_prog_num       = Y
proc_num_min       = 1
proc_num_max       = 8999
proc_num_dflt      = 1
get_proc_num       = Y

;Procedures control
full_gcode         = N
gen_procs          = Y
drill_proc         = Y
turn_proc          = N
thread_proc        = Y
gen_internal_proc  = Y
turn_common_proc   = N
gen_1_line_proc    = Y
optimize_jobs_loop = Y
G_in_2_cols        = Y

```

APPENDIX A

```
;Home
num_homes      = 6
dflt_home      = 1
get_job_home   = N
abs_zero_chng  = Y

;Positioning
dflt_start     = 200.0000 0.0000 200.0000, 7.8740 0.0000 7.8740
dflt_end       = 200.0000 0.0000 200.0000, 7.8740 0.0000 7.8740
set_z_chng     = Y
dflt_tool_chng = 100.0000 0.0000 200.0000, 3.9370 0.0000 7.8740

;Compensation
comp_exist     = Y Y
comp_arc_arc   = Y
comp_arc_line  = Y
comp_line_line = Y
next_angle     = N
comp_x_start   = N
comp_by_arcs   = N
chng_tool_table = N
look_forward   = 2

;Arc definitions
arc_exist      = Y
arc_3d         = Y
arc_quadrants  = N
arc_gt_180     = Y
;arc_max_chord = 10.0000, 0.3937
;arc_max_angle = 20.0000
arc_max_chord  = 30.0000, 1.1811
arc_max_angle  = 10.0000
arc_max_radius = 2000.0000, 78.7402

;Epsilon values
eps_angle      = 0.0020, 0.0001
eps_line       = 0.0010, 0.0001
zero_value     = 0.0010, 0.0001
min_delt_arc_rad = 0.0100, 0.0004
safety_dist    = 2.0000, 0.0787

;Feed-Spin
rapid_feed     = 5000.0000, 196.8504
max_spin       = 12000.0000
max_feed       = 9000.0000, 236.2205
spin_direction = CW

;Timing
time_factor    = 1.0000
block_time     = 0.2000
change_tool_time = 15.0000

;Part options
```

```

options      = BACK_SPIN LOGICAL
options      = GREAPER LOGICAL
options      = COLENT INTEGER
options      = number_of_parts integer

;Job options

job_opt_type    = G_GCOD N G107 use_feed m7_m8

;Drill cycles
drill_type     = Peck Peck Y Delay ; G83
drill_type     = Tapping Tapping Y Delay ; G84
drill_type     = G84_2 Tapping Y ; G84.2
drill_type     = F_Boring F_Boring Y Delay ;G85
drill_type     = G87 Peck Y Delay ; G87
drill_type     = G88 Tapping Y Delay ; G88
drill_type     = G88_2 Tapping Y ; G88.2
drill_type     = G89 F_Boring Y Delay ;G89

;Turning cycles
turn_type      = T1 N m7_m8_1

;Threading cycles
thread_type    = THREAD1 Y phase no_last_cut last_cut min_down_step spin_dir1
T_angle m7_m8_3

;Grooving cycles
groove_type    = GR N m7_m8_4

;Wire Cut cycles

;Turning definitions
turning_cycle  = Y N
groove_cycle   = Y N
combined_cycles = N
optimize_cycle = N
finish_retreat = N
semi_finish_retreat = N
fanuc_cycle    = Y

;Fourth axis
indexial_4th_axis = n
indexial_increment = 0.0000, 0.0000
init_cpos       = Y
polar_4x        = y
cartez_4x       = y
set_dir         = N

;Wire Cut parameters
lower_guide_level = 0.0000, 0.0000
group_def        = N
offset_group_name =
wc_pos_proc      = Y

```

```
u_max      = 100.0000, 3.9370
v_max      = 100.0000, 3.9370
xy_abs     = Y
uv_abs     = N

;GCTOOL
int_is_illegal = N
devison_factor = 1000.0000
gc_procs_file_name = NO_FILE
change_tool_at_ref = Y
endp
```

A.2.2. Post-Processor file: [Integrex-e-410h.gpp]

;Integrex-e-410h

```
@init_post

global string tool_diameter_f spin_f
global numeric m_feed_flag
global string f_name
GLOBAL INTEGER open_file_count pre_part_home_number
global logical main_program flag_g69 flag_start_of_job rapid_flag
global string tool_diameter_f tool_number_f spin_f xend_f
global numeric q ss
global numeric save_spin_unit save_x save_z save_tool ch_tool
global logical spin_dir_us spin_dir_us1 ben2 mill_tool flag_feed flag_plane
global logical gen_end_of_file cartez_flag

global integer prev_tool_number
  global numeric first_107 first_feed
  global integer plane split_sub
global numeric rot_x rot_y send_x send_y rot_angle delta_angle
delta_angle = 180

; Non GPPL variables
num_user_procs = 15
line_labels = false ; Jump to N...

; GPPL variables
pre_processor = 'Itegrex-e-410H'
numeric_def_f = '5.3'
integer_def_f = '5.1(p)'
gcode_f = '2.0(p)'
mcode_f = '2.0(p)'
xpos_f = '5.3(*2)'
xend_f = xpos_f
ypos_f = '5.3'
zpos_f = '5.3'
feed_f = '4.3(p)'
```

```

tool_diameter_f = '5.3/1'
blknum_f       = '5.0(p)'
tool_number_f  = 'z2.0(n)'
blknum_gen     = true
blknum_exist   = true
blknum        = 5
blknum_delta   = 5
blknum_max     = 3200000
gcode_space    = TRUE
spin_f         = '5.0(n)'

split_sub      = 1

; trace "all":1
; trace "all":5

endp

;-----

@start_of_file
; before tools definition

local integer st_length

st_length = strlen( g_file_name)
f_name = tostr(program_number)
if split_sub <> 0
  {nl, '!!open file=' f_name 'EIA!!'}
  {nl}
endif
{nl, 'O'program_number, '(', g_file_name, ')'}
if rotate_used then
  gcode = 69
  {nb, 'G'gcode}
endif

{nb, 'G91 G28 X0 Y0 Z0'}
{nb, 'G0 G90 G80 G40'}
{nb, 'G69'}
{NB, '#100=0'}
{NB, '#101=0'}

main_program   = true
pre_part_home_number = 0
endp

;-----

```

APPENDIX A

```
@start_program
; after tools definition
skipline = false
;ss = 6000
;{nb, 'G92 S'ss:'5.0(p)' }
;{nb,'M7'}
gen_end_of_file = true
endp

;-----

@end_program
xpos=xtool
zpos=ztool

if gen_end_of_file eq true
if !mill_tool
if save_spin_unit <> rpm
gcode = 97
{nb 'G'gcode}
endif

{nb,'G69'}
{nb, 'G28 G91 X0 Y0'}
{nb, 'G28 Z0'}
{nb,'G90'}
endif
if mill_tool
{nb,'G69'}
{nb, 'G28 G91 X0 Y0'}
{nb, 'G28 Z0'}
{nb,'M108'}
{nb,'M212'}
{nb,'G0 C0 B0'}
endif
{nb, 'G90'}
{nb, 'M30'}
endif
if split_sub <> 0
{nl,'!!close file=' f_name' ".EIA!!'}
{nl,'!!copy file=' f_name' ".EIA!!'}
endif
main_program = FALSE
endp

;-----

@end_of_file

endp

;-----
```

```
@relative_mode
  gcode = 91
  {nb, 'G'gcode, ''}
  skipline = FALSE
endp

;-----

@absolute_mode
  gcode = 90
  {nb, 'G'gcode, ''}
  skipline = true
endp

;-----

@machine_plane

if job_machine_type eq milling
  if machine_plane eq XY
    gcode = 17
  endif
  if machine_plane eq YZ
    gcode = 19
  endif
  if machine_plane eq ZX
    gcode = 18
  endif
else
  gcode = 18
endif
{nb, 'G'gcode}

endp

;-----

@call_proc

  {nb, 'M98 P'(label+10000)}
  ;{[' ('message, ')]}
  ;{[' ('upper(message), ')]}
endp

;-----

@call_prms

  gcode = 65
```

```
        {nb, 'G' gcode ' P'(label+10000) ' L'ncalls}
        {[ ' (upper(message), ')]}
endp

@proc
  blknum = 10

  if split_sub <> 0
    {nl, '!!open file=(label+10000).EIA!!'}
  endif
  cartez_flag = true

  rot_x = 99999999
  rot_y = 99999999

  {nl}
  {nl, 'O'(label+10000)}
endp

;-----

@end_proc

  {nb, 'M99'}
  if split_sub <> 0
    {nl, '!!close file=(label+10000).EIA!!'}
    {nl, '!!copy file=(label+10000).EIA!!'}
  endif
endp

;-----

@loop
  local integer var_num

  var_num = loop_level + 20
  {nb, '#', var_num, ' = 0'}
  {nb, 'WHILE [#', var_num, ' LT ', loop_count, '] DO ', loop_level}
  call @home_number
endp

;-----

@end_loop
  local integer var_num

  var_num = loop_level + 20
  {nb '#', var_num, ' = #', var_num, ' + 1'}
  {nb 'END ', loop_level}
endp
```

```

;-----
@def_tool
; {nb, '(G10 L12 P', (tool_number), ' D'(tool_offset*2), ')'}
; {nb, '(N0', tool_number '-T'(tool_number+0), ' - DIA ', TOOL_message ')'}
endp

;-----

@rapid_move

if job_machine_type eq milling
  if machine_plane <> xy      ;spatial format for xy plane
    xpos_f = '5.3'
  else
    zpos_f = '5.3'
    ;spatial format for zx plane
    xpos_f = '5.3'
  endif
  call @mill_rapid_move
  xpos_f = '5.3'      ;returne to normal formt
  zpos_f = '5.3'      ;returne to normal formt
else
  xpos_f = '5.3(*2)'
  gcode = 0
  {nb, ['G'gcode]}
  {' X'xpos, [' Z'zpos]}
  save_x = xpos
  save_z = zpos
endif

endp

@mill_rapid_move

{nb, ''}
gcode = 0
if change(gcode) then
  {'G'gcode}
else
  {' '}
endif

if flag_start_of_job
  {' X'xpos, ' Y'ypos }
  {nb, ' Z'zpos}
  {' M8'}
  flag_start_of_job = false
else
  {' X'xpos, [' Y'ypos] ,[' Z'zpos]}
endif

```

```
rapid_flag = true

endp
;-----

@move_4x

xpos_f = '5.3'

if next_command eq '@move4x_cartesian'
    {nb,'G17 G90 C0',[' X'xpos],[' Z'zpos ] }
else
    gcode = 0
    {nb,['G'gcode] [' X'xpos] [' Z'zpos ] [' C['cpos '+#100]] }
endif

endp

;-----

@line
if job_machine_type eq milling
    if machine_plane <> xy                ;spatial format for xy plane
        xpos_f = '5.3'
    else
        zpos_f = '5.3'
        xpos_f = '5.3'
    endif
    call @line_mill
    xpos_f = '5.3(*2)'                    ;returne to normal formt
    zpos_f = '5.3'                        ;returne to normal formt
else
    xpos_f = '5.3(*2)'
    gcode = 1
    {nb, [' G'gcode]}

    if ben2 and work_type eq ROUGH
        if process_type eq LONG then
            {' X'xpos}
        else
            {' Z'zpos}
        endif
    endif
    ben2 = false
else
    {' X'xpos}, [' Z'zpos]}
```

```
endif

if prev_command eq '@turn_proc' then
    change(feed) = TRUE
endif

    {' F'feed}}
endif

endp

@line_mill

gcode = 1
if change(gcode) then
    {nb,'G'gcode}
else
    {nb,' '}
endif

    {' X'xpos}, {' Y'ypos},{ ' Z'zpos}}

if m_feed_flag eq 1
    {' F'feed}
    m_feed_flag = 0
else
    if change(feed) eq true
        {' F'feed}}
    else
    endif
endif
endp

@line_4x
local numeric x1 z1

xpos_f = '5.3'

gcode = 1

if next_command eq '@start_cartesian'
    {nb,'G17 G90'}
    skipline = false
    if use_feed eq 0
        {nb,['G'gcode] [' X'xpos] [' Z'zpos ] [' C['cpos '+'#100]'}}
        if flag_feed eq true
            {' F'feed}
            flag_feed = false
        else
            {' F'feed}}
        endif
    endif
endif
```

```
    else
        {nb,['G'gcode] [' X'xpos] [' Z'zpos ] [' C['cpos '+#100]]}
        if first_feed eq 0 or first_feed eq 1
            first_feed = first_feed + 1
            if flag_feed eq true
                {' F'feed}
                flag_feed = false
            else
                {' F'feed}
            endif
        endif
    endif
endif

ch_tool = 0

if g107 eq 1
    if first_107 eq 0
        first_107 = 1
        {nb,'G1 G18 W0 H0'}
        {nb,'G107 C'(radial_diameter/2)}
    endif
endif

endp
```

```
;-----
```

```
@arc
```

```
if job_machine_type eq milling
    if machine_plane <> zx          ;spatial format for xy plane
        xpos_f = '5.3 '
    else
        zpos_f = '5.3(*2)'        ;spatial format for zx plane
    endif
    call @arc_mill
    xpos_f = '5.3'                ;returne to normal formt
    zpos_f = '5.3'                ;returne to normal formt
else
    xpos_f = '5.3(*2)'
    if arc_direction eq CCW then
        gcode = 3
    else ; CW
        gcode = 2
    endif
    {nb, ['G'gcode] ' X'xend ' Z'zend}

    if arc_size >= 180 then
```

```

        radius = -radius
    endif
    {' R'radius, [' F'feed]}
endif

endp

@arc_mill
;-- arc plane --
if change(arc_plane) then
    if arc_plane eq XY then
        gcode = 17
    endif
    if arc_plane eq YZ then
        gcode = 18
    endif
    if arc_plane eq ZX then
        gcode = 19
    endif
    {' nb,'G'gcode, ' '}
endif

if arc_direction eq CCW then
    gcode = 3
else ; CW
    gcode = 2
endif
if change(gcode) then
    {' nb,'G'gcode}
else
    {' '}
endif
if m_feed_flag eq 1
    {' F'feed}
    m_feed_flag = 0
else
    {' [' F'feed]}
endif

{' [' X'xpos] [' Y'ypos] [' Z'zpos]}

if arc_size eq 360 then
    if arc_plane eq XY then
        {' I'xcenter_rel, ' J'ycenter_rel}
    endif
    if arc_plane eq YZ then
        {' J'xcenter_rel, ' K'ycenter_rel}
    endif
    if arc_plane eq ZX then
        {' K'xcenter_rel, ' I'ycenter_rel}
    endif
else
    if arc_size >= 180 then

```

APPENDIX A

```
        radius = -radius
    endif
    {' R'radius}
endif

endp
;-----

@compensation
if side eq COMP_LEFT then
    gcode = 41
    {'nb 'G'gcode ' ' }
endif
if side eq COMP_RIGHT then
    gcode = 42
    {'nb 'G'gcode ' ' }
endif
if side eq COMP_OFF then
    gcode = 40
    {'nb 'G'gcode ' ' }
endif

    skipline = false

endp
;-----

@delay
    gcode = 4
    {'nb 'G'gcode, ' P'delay_period:integer_def_f}
endp

;-----

@change_ref_point
    {'nb,'G10 L10 P'home_number ' U'(-(xhome)), ' V'(-(yhome)), ' W'zhome}
    {'nb 'G90'}
endp

;-----

@home_number
    gcode = 53 + home_number
    ; {'nb, 'G'gcode}
endp

;-----

@rotate
; Not exist in FANUC 6M
if rotate_cancel then
```

```

    gcode = 69
    {nb, 'G'gcode}
else
    gcode = 68
    {nb, 'G'gcode, ' X0 Y0 G91 R'angle}
    {nb, 'G90'}
endif
endp

;-----

@fourth_axis
gcode = 0
IF rot_axis_type eq axis4_none
    {nb, 'M212'}
    {nb, 'G'gcode, ' C('angle ' + #101)'}
    {nb, 'M210'}
ELSE
    {NB, '#100 = ' angle }
ENDIF

endp

;-----

@change_tool
local numeric xx rr
{nb, 'M09'}
if mill_tool eq false
    {nb, 'M205'}
endif
if !first_tool
    {nb, 'G69'}
    {nb, 'G91 G28 X0'}
    {nb, 'G90'}
endif
local logical save_blknum_gen

gcode = 54
{nb, 'G'gcode}

{nb, '(-----)'}
{nl 'N0'tool_number ' ( TOOL ' tool_number ' DIAMETER ' tool_diameter)}
{nb, '(-----)'}
{nb, 'M200'}
{nb, 'G10.9 X0'}
{nb, 'T'tool_number:'2/2.0(p)" M06 (DIA ',TOOL_message ')}
{nb, 'M8'}
call @machine_plane
tool_direction = CW

```

```
job_machine_type = 0
call @start_tool_mill

call @my_tmatrix

mill_tool = true
prev_tool_number = tool_number
endp

; -----

@turn_change_tool

    local numeric tool_ang
    {nb, ''}
    {nb, 'G10.9 X1'}
    {nb, 'M212'}
    {nb, 'M202'}
        blknum=true
    local numeric xx rr
    {nb, 'M09'}
    if mill_tool eq false
        {nb, 'M5'}
    endif
    if !first_tool
        {nb, 'G69'}
        {nb, 'G91 G28 X0'}
        {nb, 'G90'}
    endif
    local logical save_blknum_gen

gcode = 53 + home_number
{nb, 'G'gcode}

{nb, '(-----)'}
{nl 'N0'tool_number ' ( TOOL ' tool_number ' DIAMETER ' tool_diameter)}
{nb, '(-----)'}

{nb, 'T'tool_number:'2/2.0(p)" M06 (DIA ',TOOL_message ')}

if tool_mode eq long
    tool_ang = 90.0
else
    tool_ang = 0.0
endif

    {nb, 'M107'}
    {nb, 'G0 B'tool_ang ' Y0'}
    {nb, ' M108'}
    ; {nb, call message}

job_machine_type = 1
call @start_tool
```

```

    {nb,'M51'}
    xpos = xnext
    zpos = znext
    gcode = 1

    save_spin_unit = spin_unit
    save_tool = tool_number
    spin_dir_us = false
    spin_dir_us1 = true
    mill_tool = false

    blknum = true
    gcode = 43
    {nb,'G0 G'gcode ' P1' ' X'xpos ' Z'zpos}
    mill_tool = false
    prev_tool_number = tool_number
endp

;-----

@message
    {nb,'(, message, ')'}
endp

;-----

@turn_drill

local string g_code
    gcode = 98
    {nb,'G'gcode, ''}
    if drill_type eq Peck then;(G83)
        g_code = '83'
    endif
    if drill_type eq Tapping then;(G84)
        g_code = '84'
    endif
    if drill_type eq G84_2 then;(G84.2)
        g_code = '84.2'
    endif
    if drill_type eq F_Boring then;(G85)
        g_code = '85'
    endif
    if drill_type eq G87 then;(G87)
        g_code = '87'
    endif
    if drill_type eq G88 then;(G88)
        g_code = '88'
    endif
    if drill_type eq G88_2 then;(G88.2)
        g_code = '88.2'
    endif
    if drill_type eq G89 then;(G89)

```

```
    g_code = '89'
  endif
  {'G',g_code, ' Z'drill_lower_z, ' R'drill_upper_z}
  if drill_type eq peck or drill_type eq G87 then
    {' Q'(down_step*1000)}
    {' P'delay:integer_def_f}
    {' L0'}

  endif
  if drill_type eq tapping or drill_type eq G88 or drill_type eq F_Boring or drill_type eq G89
then
    {' P'delay:integer_def_f}

  endif
  if drill_type eq G84_2 then

    endif
    if drill_type eq G88_2 then

      endif

    {' F'feed:feed_f}
  endp
```

@mill_drill1

```
  local numeric m
  m = 2

  local string g_code
  gcode = 98
  {'nb, 'G'gcode, ''}
  if drill_type eq Peck then;(G83)
    g_code = '83'
  endif
  if drill_type eq Tapping then;(G84)
    g_code = '84'
  endif
  if drill_type eq G84_2 then;(G84.2)
    g_code = '84.2'
  endif
  if drill_type eq F_Boring then;(G85)
    g_code = '85'
  endif
  if drill_type eq G87 then;(G87)
    g_code = '87'
  endif
  if drill_type eq G88 then;(G88)
    g_code = '88'
  endif
  if drill_type eq G88_2 then;(G88.2)
    g_code = '88.2'
```

```

endif
if drill_type eq G89 then;(G89)
  g_code = '89'
endif
{'G',g_code, 'Z'drill_lower_z, ' R'drill_upper_z}
if drill_type eq peck or drill_type eq G87 then
  if down_step == 0
    down_step = drill_depth
  endif
  {' Q'down_step}
  {' P'delay:integer_def_f}

endif
if drill_type eq tapping or drill_type eq G88 or drill_type eq F_Boring or drill_type eq G88
or drill_type eq G89 then
  {' P'delay:integer_def_f}

endif
if drill_type eq G84_2 then

  endif
if drill_type eq G88_2 then

  endif
  {' L0'}
  {' F'feed:feed_f}

endp

@drill
  call @mill_drill1
endp
;-----

@drill_point

  {nb, ' ', [' X'xpos], [' Y'ypos], [' Z'zpos]}

endp

@drill4x_pnt
  if first_drill
    {nb,'M212'}
  endif
  {nb, ' ', [' X'xpos], [' Z'zpos] , [' C'cpos]}
endp

;-----

```

```
;-----  
@mirror  
  if mirror_type eq MIRROR_OFF then  
    {nb, 'G50.1 X0 Y0'}  
  else  
    {nb, 'G51.1 '  
  if mirror_type eq MIRROR_X then  
    {'X1 Y0'}  
  endif  
  if mirror_type eq MIRROR_Y then  
    {'X0 Y1'}  
  endif  
  if mirror_type eq MIRROR_XY then  
    {'X1 Y1'}  
  endif  
endif  
endp
```

```
;-----  
@end_drill  
  ; initalization  
  
  gcode = 80  
  
  {nb, 'G'gcode}  
  
endp
```

```
;-----  
@halt_program  
  {'M0'}  
endp
```

```
;-----  
@round_comp  
  ; NOP  
endp
```

```
;-----  
@start_of_job  
  skipline = false  
  {' ('job_name,')'  
  skipline= true  
  
  flag_start_of_job = true
```

```

endp

;-----

@end_of_job

endp

;-----

@assign_axis

endp

; =====
; USER DEFINED PROCEDURES
; =====

@call_simple_proc
  active(message) = FALSE
  active(parm1)   = FALSE
  active(parm2)   = FALSE
  active(parm3)   = FALSE
  proc_count     = 1
  call @call_proc
endp

;-----

@start_tool_mill
  if tool_direction eq CW then
    mcode = 3
  else
    ; CCW
    mcode = 4
  endif

  {nb, 'S'spin:spin_f, ' M'mcode}

endp

@start_tool
  local numeric save_spin
  save_spin = spin
  if back_spin eq false
    if tool_direction eq CW then
      mcode = 204
    else
      ; CCW
      mcode = 203
    endif
  else
    if tool_direction eq CW then
      mcode = 304
    else
      ; CCW

```

```
        mcode = 303
    endif
endif

    {nb,' S'spin:spin_f, ' M'mcode}
endp
;-----

@stop_tool
    if back_spin eq false
        {nb ' M205'}
    else
        {nb ' M5'}
    endif
endp

;-----

@home_data2

    local numeric ang_x
    local numeric a c d new_x new_y

    if change (rotate_angle_x)
        if flag_g69 eq false
            skipline=true
            {nb, 'G69'}
            {nb, 'M9'}
            {nb, 'G28 G91 X0 Y0'}
            {nb, 'G28 Z0'}
            {nb, 'G90'}
            flag_g69 = true
        endif
    endif

    {nb,'T'tool_number}

        if job_machine_type eq milling

            {nb,'M212'}
            {nb,'G0 C'rotate_angle_z }
            {nb,'M210'}
        else
            {nb, 'M210'}
        endif

            {nb,'M108'}
            {nb,'G0 B' (rotate_angle_y):'3.3'}
            {nb,'M107'}
```

```

    send_x = shift_x
    send_y = shift_y
    rot_angle = rotate_angle_z
    call @Rorate_x_y

    {NB, 'G68 X'rot_x:'5.3' ' Y'rot_y:'5.3' ' Z'shift_z:'5.3'}
    {' I0 J1 K0 R'(rotate_angle_y):'3.3'}
    call @start_tool_mill

    {nb, '(===== end =====)'}
    flag_g69 = false

endp

@Rorate_x_y
    local numeric local_rot_angl

    rot_x = send_x * cos(rot_angle) - send_y * sin(rot_angle)
    rot_y = send_x * sin(rot_angle) + send_y * cos(rot_angle)

endp

@turning
    local numeric ww uu rr aa bb
    local logical ben1

    if semi_finish then
        rough_offset_x = semi_offset_x
        rough_offset_z = semi_offset_z
        work_type = rough
        if process_type eq long
            if turning_mode <> internal
                uu = save_x - first_pos_x
            else
                uu = first_pos_x - save_x
            endif
            down_step = (uu+10)
        else
            ww = save_z - first_pos_z
            down_step = (ww+10)
        endif
        ben1 = true
    endif

    if work_type eq ROUGH then
        if process_type eq LONG then
            gcode = 271
            {nb, 'G'gcode}
            {' U'down_step, ' R'(down_step/2) }
            {nb, 'G'gcode ' P'start_line ' Q'end_line}
        endif
    endif

```

```
    {' U'rough_offset_x ' W'rough_offset_z}
else
    gcode = 272
    {nb, 'G'gcode }
    {' W'down_step, ' R'(down_step/2) }
    {nb 'G'gcode ' P'start_line ' Q'end_line }
    {' U'rough_offset_x ' W'rough_offset_z}
endif
if feed_unit eq css
    {' F'feed }
else
    {' F'feed:'5.0(p)' }
endif
ben2 = true
endif

if work_type eq COPY then

    ww = save_z - first_pos_z
    uu = save_x - first_pos_x

    aa = (uu - rough_offset_x)
    bb = (ww - rough_offset_z)
    rr = sqrt(aa * aa + bb * bb)

    gcode = 273
    if ben1 eq false
        num_down_steps = rr / down_step + 1
    else
        num_down_steps = 1
    endif

    {nb, 'G'gcode,' U'uu, ' W'ww, ' R'num_down_steps}
    {nb, 'G'gcode ' P'start_line ' Q'end_line }
    {' U'rough_offset_x ' W'rough_offset_z}

    if feed_unit eq css
        {' F'feed }
    else
        {' F'feed:'5.0(p)' }
    endif

endif

if semi_finish then
    gcode = 150
endif

if finish then
    gcode = 151
endif

endp
```

```
@turn_proc
endp

@turn_endproc
endp

@end_job_procs
  {nb, 'M99'}

endp

@m_feed_spin

  call @start_tool_mill
endp

@feed_spin

if tool_number ne 99
  {nb}
  if feed_unit eq mm_min
    feed_f = '5.0(p)'
    {'G94'}
  else
    ; CSS
    {'G95'}
    feed_f = '4.3(p)'
  endif

  if spin_unit eq rpm ; Mm/Sec
    gcode = 97
    {' G'gcode}
  else
    gcode = 96
    {' G'gcode}
  endif

  if change(spin) then
    {' S'spin}
  endif
endif
endp

@thread
  local string for1 for2 for3 for4
  local numeric taper_angle
  for = '6.0(n*10000)'
  for1 = '2/2.0(P)'
  for2 = '2/2.0(P*10)'
  for3 = '5.0(P*1000)'
  for4 = '5.3'
```

```
taper_angle = (first_pos_x-last_pos_x)
if last_pos_x <> first_pos_x
  if turning_mode <> internal
    xpos = (last_pos_x + depth + safety )
  else
    xpos = (last_pos_x - depth - safety )
  endif
  gcode = 0
  {nb,'G'gcode ' X'xpos }
endif

if lead_unit <> mm
  lead = (25.4/lead)
endif

if spin_dir1 eq 1
  tool_direction = cw
  if spin_dir_us eq false
    call @stop_tool
    call @start_tool
  endif
  spin_dir_us = true
else
  tool_direction = ccw
  if spin_dir_us1 eq false
    call @stop_tool
    call @start_tool
    spin_dir_us = false
  endif
  spin_dir_us1 = true
endif

if work_type eq multiple
  gcode = 276
  ;tool_nose_engle = 0 60 29 55 30 80
  {nb, 'G'gcode, ' P'no_last_cut:for1, phase:for2 ,T_angle:for1 '
Q'min_down_step:numeric_def_f ' R'last_cut:numeric_def_f}
  {nb 'G'gcode ' X'last_pos_x:xpos_f ' Z'last_pos_z }
  if taper_angle <> 0
    { ' R'taper_angle }
  endif
  { ' P'depth:numeric_def_f ' Q'down_step:numeric_def_f ' F'lead }
else
  gcode = 92
  {nb 'G'gcode ' X'last_pos_x:xpos_f ' Z'last_pos_z }
  if taper_angle <> 0
    { ' R'taper_angle }
  endif
  { ' F'lead }
endif
endp
```

```

@groove
  local string for1 for2 for3 for4
  local numeric d1 d2

  for1 = '2/2.0(P)'
  for2 = '2/2.0(P*10)'
  for3 = '5.0(P*1000)'
  for4 = '5.3(P)'
  d1 = abs(side_step)
  d2 = abs(down_step)
  if process_type eq face

    gcode = 274
    {nb 'G'gcode, 'R'release_dist }
    {nb 'G'gcode,' X'last_pos_x:xpos_f ' Z'last_pos_z ' P'd1 :numeric_def_f ' Q'd2
:numeric_def_f }
  else

    gcode = 275
    {nb 'G'gcode, 'R'release_dist }
    {nb 'G'gcode,' X'last_pos_x:xpos_f ' Z'last_pos_z ' P'd2:numeric_def_f '
Q'd1:numeric_def_f}
  endif
  { ' F'feed }
endp

@turn_opt_parms
if tool_number ne 99
  if job_type eq 'turn'
    if m7_m8_1 eq 0
      {nb,'M7'}
    else
      {nb,'M8'}
    endif
  endif
endif
; if job_type eq 'drill'
; if m7_m8_2 eq 0
; {nb,'M7'}
; else
; {nb,'M8'}
; endif
; endif
; if job_type eq 'thread'
  if m7_m8_3 eq 0
    {nb,'M7'}
  else
    {nb,'M8'}
  endif
endif
endif
if job_type eq 'groove'
  if m7_m8_4 eq 0
    {nb,'M7'}
  else

```

```
        {nb,'M8'}
    endif
endif
endif

endp

@rotary_info

    if rot_axis_type eq axis4_radial
        plane = 19
        flag_plane = true
        machine_plane = YZ
    endif
    if rot_axis_type eq axis4_top
        plane = 19
        flag_plane = true
        machine_plane = YZ
    endif
    if rot_axis_type eq axis4_face
        plane = 17
        flag_plane = true
        machine_plane = XY
    endif

endp

@chng_tool_cnext
endp

@init_cpos
endp

@move4x_dir
endp

@line4x_dir
endp

@move4x_polar
endp

@line4x_polar
endp

@arc4x_polar
```

endp

```
@move4x_cartesian
  {nb}
  gcode = 0
  if change(gcode) then
    {'G'gcode}
  else
    {' '}
  endif
  {' X'xpos}, {' C'cpos}, {' Z' zpos}}
endp
@start_cartesian
```

```
  {nb,'G12.1'}
  {nb,'G0 C0',{' X'xpos] }
```

endp

```
@end_cartesian
  {nb,'G13.1'}
```

endp

```
@line4x_cartesian
  {nb}
  gcode = 1
  if change(gcode) then
    {'G'gcode}
  else
    {' '}
  endif
  {' X'xpos}, {' C'cpos}, {' Z'zpos}, {' F'feed}}
endp
```

```
@arc4x_cartesian
  {nb}
  if arc_direction eq CCW then
    gcode = 3
  else ; CW
    gcode = 2
  endif
  if change(gcode) then
    {'G'gcode}
  else
    {' '}
  endif
  {' X'xpos] [' C'cpos] }

  if arc_size eq 360 then
```

```
        {' I'xcenter_rel, ' K'ycenter_rel}
else
  if arc_size >= 180 then
    radius = -radius
  endif
  {' R'radius}
endif

  {' F'feed}}
endp

@drill4x_polar
endp

@drill4x_cartesian
endp

@tmatrix
  call @my_tmatrix

endp

@my_tmatrix
  if job_machine_type eq milling
  if main_program == true
  if part_home_number <> pre_part_home_number

    {'nb, '(===== CHANGE HOME =====)'}
    {'nb, '( HOME = ' mac_number ' POSITION = 'position )}

  if mac_number eq 1

    {' B0 C'rotate_angle_z ' ) ' }
    {'NB, '#101='rotate_angle_z}
    {'nb, '(=====)'}
    {'nb, 'G91 G28 X0'}
    {'nb, 'G90'}
    if rot_axis_type eq axis4_none
      {'nb, 'G69'}
      {'nb, 'M108'}
      {'nb, 'M212'}
      {'nb, 'G0 B0. C'rotate_angle_z ' ( FACE XYZ OPERATION )'}
      {'nb, 'M107'}
      {'nb, 'M210'}
    else
      {'nb, 'G69'}
      {'nb, 'M108'}
      {'nb, 'M212'}
      {'nb, 'G0 B0. C'rotate_angle_z ' Y0" ( FACE XZC OPERATION )'}
      {'nb, 'M107'}
```

```

endif
endif
if mac_number eq 2

  { ' B90 C'rotate_angle_x ' ) ' }
  { NB, '#101='rotate_angle_z }
  { nb, '(=====)' }
  { nb, ' G91 G28 X0' }
  { nb, 'G90' }
  if rot_axis_type eq axis4_none

    { nb, 'G69' }
    { nb, 'M108' }
    { nb, 'M212' }
    { nb, 'G0 B90. C'rotate_angle_x ' ( SIDE XYZ OPERATION )' }
    { nb, 'M107' }
    { nb, 'M210' }
    zpos = znext
    ypos = ynext
    change(xpos) = false
    call @rapid_move
    change(ypos) = false
    change(zpos) = false
    xpos = xnext
    gcode = 43
    { nb, 'G'gcode }
    skipline = false
    call @rapid_move
  else
    { nb, 'G69' }
    { nb, 'M108' }
    { nb, 'G0 B90. C'rotate_angle_x ' Y0' ' ( SIDE XZC OPERATION )' }
    { nb, 'M107' }
    change(ypos) = false
    change(zpos) = false
    xpos = xnext
    gcode = 43
    { nb, 'G'gcode }
    skipline = false
    call @rapid_move
  endif
endif
if mac_number eq 3

  { ' B'rotate_angle_y ' C'rotate_angle_z ' PLANE ROTATION ) ' }
  { NB, '#101='rotate_angle_z }
  { nb, '(=====)' }
  { nb, ' G91 G28 X0' }
  { nb, 'G90' }
  call @home_data2
endif
{ nb, 'G94' }
zpos = xnext

```

```
        change(ypos) = false
        change(xpos) = false
        zpos = (tool_start_plane + 50)
        gcode = 43
        {nb,'G'gcode}
        skipline = false
        call @rapid_move
    pre_part_home_number = part_home_number
endif
endif
endif

endp
```

APPENDIX B

EXAMPLE: FANUC G-Code file

```
% ; @start_of_file
O5000 (OUT) ; "
(SUBROUTINES: O1 .. O5) ; "
(G10 L12 P51 R6.35) ; @def_tool
(G10 L12 P64 R5.) ; @def_tool
(G10 L12 P62 R6.) ; @def_tool
G17 ; @machine_plane
G80 G49 G40 ; @start_program
G54 ; "
(*TOOL 1 - DIA 12.7*) ; @change_tool
T1 ; "
M98 P5 ; "
N5 G43 H1 D51 G0 X-16.35 Y-53.75 Z50. S3000 M3 ; "
M8 ; "
M98 P1 (F-PR1-T1) ; @call_proc
(*TOOL 14 - DIA 10.0*) ; @change_tool
T14 ; "
M98 P5 ; "
N10 G43 H14 D64 G0 X93.036 Y-50. Z50. S2000 M3 ; "
M8 ; "
M98 P2 (P-PC1-T14) ; @call_proc
(*TOOL 12 - DIA 12.0*) ; @change_tool
T12 ; "
M98 P5 ; "
N15 G43 H12 D62 G0 X75. Y-19. Z50. S3000 M3 ; "
M8 ; "
M98 P3 (D-DR1-T12) ; @call_proc
M9 ; @end_program
G0 X0. Y0. Z60. ; "
M30 ; "
O1 ; @proc
```

```

(-----) ; @message
(F-PR1-T1 - PROFILE ) ; @message
(-----) ; @message
G0 X-16.35 Y-53.75 Z10. ; @rapid_move
  Z2. ; @rapid_move
G1 Z-30. F75 ; @line
  Y-37.832 F250 ; @line
G2 X-8.35 Y-24.565 R15. ; @arc
G1 Y-20.5 ; @line
G2 X20.5 Y8.35 R28.85 ; @arc
G1 X49.565 ; @line
G2 X62.832 Y16.35 R15. ; @arc
G1 X87.168 ; @line
G2 X100.435 Y8.35 R15. ; @arc
G1 X129.5 ; @line
G2 X158.35 Y-20.5 R28.85 ; @arc
G1 Y-24.565 ; @line
G2 X166.35 Y-37.832 R15. ; @arc
G1 Y-62.168 ; @line
G2 X158.35 Y-75.435 R15. ; @arc
G1 Y-79.5 ; @line
G2 X129.5 Y-108.35 R28.85 ; @arc
G1 X100.435 ; @line
G2 X87.168 Y-116.35 R15. ; @arc
G1 X62.832 ; @line
G2 X49.565 Y-108.35 R15. ; @arc
G1 X20.5 ; @line
G2 X-8.35 Y-79.5 R28.85 ; @arc
G1 Y-75.435 ; @line
G2 X-16.35 Y-62.168 R15. ; @arc
G1 Y-53.75 ; @line
  X-8.35 ; @line
  Y-37.832 ; @line
G2 X-3.438 Y-31.15 R7. ; @arc
G1 X-0.35 Y-30.185 ; @line
  Y-20.5 ; @line
G2 X20.5 Y0.35 R20.85 ; @arc
G1 X55.185 ; @line
  X56.15 Y3.438 ; @line
G2 X62.832 Y8.35 R7. ; @arc
G1 X87.168 ; @line
G2 X93.85 Y3.438 R7. ; @arc
G1 X94.815 Y0.35 ; @line
  X129.5 ; @line

```

G2 X150.35 Y-20.5 R20.85 ; @arc
G1 Y-30.185 ; @line
X153.438 Y-31.15 ; @line
G2 X158.35 Y-37.832 R7. ; @arc
G1 Y-62.168 ; @line
G2 X153.438 Y-68.85 R7. ; @arc
G1 X150.35 Y-69.815 ; @line
Y-79.5 ; @line
G2 X129.5 Y-100.35 R20.85 ; @arc
G1 X94.815 ; @line
X93.85 Y-103.438 ; @line
G2 X87.168 Y-108.35 R7. ; @arc
G1 X62.832 ; @line
G2 X56.15 Y-103.438 R7. ; @arc
G1 X55.185 Y-100.35 ; @line
X20.5 ; @line
G2 X-0.35 Y-79.5 R20.85 ; @arc
G1 Y-69.815 ; @line
X-3.438 Y-68.85 ; @line
G2 X-8.35 Y-62.168 R7. ; @arc
G1 Y-53.75 ; @line
G41 G1 X5. ; @compensation + @line
Y-42.5 ; @line
X13. Y-40. ; @line
Y-20.5 ; @line
G2 X20.5 Y-13. R7.5 ; @arc
G1 X65. ; @line
X67.5 Y-5. ; @line
X82.5 ; @line
X85. Y-13. ; @line
X129.5 ; @line
G2 X137. Y-20.5 R7.5 ; @arc
G1 Y-40. ; @line
X145. Y-42.5 ; @line
Y-57.5 ; @line
X137. Y-60. ; @line
Y-79.5 ; @line
G2 X129.5 Y-87. R7.5 ; @arc
G1 X85. ; @line
X82.5 Y-95. ; @line
X67.5 ; @line
X65. Y-87. ; @line
X20.5 ; @line
G2 X13. Y-79.5 R7.5 ; @arc

G1 Y-60.	; @line
X5. Y-57.5	; @line
Y-53.75	; @line
G40 G1 X-1.35 Y-53.75	; @compensation + @line
G0 Z10.	; @rapid_move
M99	; @end_proc
O2	; @proc
(-----)	; @message
(P-PC1-T14 - POCKET)	; @message
(-----)	; @message
G0 X93.036 Y-50. Z10.	; @rapid_move
Z2.	; @rapid_move
G1 Z-10. F105	; @line
G2 X97. Y-53.225 R20. F350	; @arc
G1 Y-46.775	; @line
G2 X93.036 Y-50. R20.	; @arc
G1 X56.964	; @line
G2 X53. Y-46.775 R20.	; @arc
G1 Y-53.225	; @line
G2 X56.964 Y-50. R20.	; @arc
G1 X59.598 Y-54.25	; @line
G2 X67.5 Y-52. R15.	; @arc
G1 X82.5	; @line
G2 X95.767 Y-60. R15.	; @arc
G1 X102.21	; @line
G2 X102. Y-57.5 R15.	; @arc
G1 Y-42.5	; @line
G2 X102.21 Y-40. R15.	; @arc
G1 X95.767	; @line
G2 X82.5 Y-48. R15.	; @arc
G1 X67.5	; @line
G2 X54.234 Y-40. R15.	; @arc
G1 X47.79	; @line
G2 X48. Y-42.5 R15.	; @arc
G1 Y-57.5	; @line
G2 X47.79 Y-60. R15.	; @arc
G1 X54.233	; @line
G2 X59.598 Y-54.25 R15.	; @arc
G1 X62.232 Y-58.5	; @line
G2 X67.5 Y-57. R10.	; @arc
G1 X82.5	; @line
G2 X92.045 Y-64.017 R10.	; @arc
G1 X92.352 Y-65.	; @line
X110.386	; @line

G2 X107. Y-57.5 R10.	; @arc
G1 Y-42.5	; @line
G2 X110.386 Y-35. R10.	; @arc
G1 X92.352	; @line
X92.045 Y-35.983	; @line
G2 X82.5 Y-43. R10.	; @arc
G1 X67.5	; @line
G2 X57.955 Y-35.983 R10.	; @arc
G1 X57.648 Y-35.	; @line
X39.614	; @line
G2 X43. Y-42.5 R10.	; @arc
G1 Y-57.5	; @line
G2 X39.614 Y-65. R10.	; @arc
G1 X57.648	; @line
X57.955 Y-64.017	; @line
G2 X62.232 Y-58.5 R10.	; @arc
G1 X67.65 Y-62.	; @line
G41 G1 Y-67.	; @compensation + @line
X82.5	; @line
X85. Y-75.	; @line
X125.	; @line
Y-60.	; @line
X117. Y-57.5	; @line
Y-42.5	; @line
X125. Y-40.	; @line
Y-25.	; @line
X85.	; @line
X82.5 Y-33.	; @line
X67.5	; @line
X65. Y-25.	; @line
X25.	; @line
Y-40.	; @line
X33. Y-42.5	; @line
Y-57.5	; @line
X25. Y-60.	; @line
Y-75.	; @line
X65.	; @line
X67.5 Y-67.	; @line
X67.65	; @line
G40 G1 X67.65 Y-62.	; @compensation + @line
G0 Z10.	; @rapid_move
M99	; @end_proc
O3	; @proc
(-----)	; @message

```

( D-DR1-T12 - DRILL )           ; @message
(-----)                       ; @message
G0 X75. Y-19. Z10.              ; @drill
G98 G83 Z-30. R2. Q30. F75     ; @drill
M98 P4 (DR1)                   ; @call_proc
M99                             ; @end_proc
O4                              ; @proc
(-----)                       ; @message
( DR1 - DRILLS )               ; @message
(-----)                       ; @message
    Y-81.                      ; @drill_point
    X131. Y-50.                ; @drill_point
    X19.                       ; @drill_point
G80                             ; @end_drill
M99                             ; @end_proc
O5                              ; @end_of_file
(-----)                       ; "
(- CHANGE TOOL -)              ; "
(-----)                       ; "
G80 G49 G40 M9                 ; "
G91 G28 Z0. M5                 ; "
G90 M1                          ; "
M6                              ; "
M99                             ; "
%                               ; "
-

```

APPENDIX C

C. GPPtool error messages

GPPtool generates error messages while translating **SolidCAM** tool-path commands into G-Code. If an error occurs, GPPtool displays the line number at which the error occurred, the error number and an explanatory text for that error. In this appendix you will find a complete list of GPPtool error messages together with explanations of how to recover from these errors.

- ◆ Error 193 : Not enough memory

Description: GPPtool runs out of memory.

Recovery : Quit, and re-run **SolidCAM**. Try to generate the G-Code now. If this fails, see Chapter 2 for generation of G-Code outside **SolidCAM**.

- ◆ Error 194 : ENDIF statement missing

Description: GPPtool's 'IF' statement started, but no respective 'ENDIF' statement found.

Recovery : Check [MACHINE.GPP] file for misspelling of the 'ENDIF' keyword. Check for balanced number of 'IF' and 'ENDIF' keywords.

- ◆ Error 383 : Variable 'xxx' is not defined

Description: Variable 'xxx' was used, but it is neither system variable nor was it explicitly declared as global or local variable.

Recovery : Check for misspelling of the variable name. Declare it as global/local variable if necessary.

◆ Error 384 : Program too complex

Description: The GPPL program is too complicated for GPPtool.

Recovery: Try to simplify your program: use less nested 'IF' statements, or reduce procedure CALLs level. Error 384 rarely occurs; if you get this error and you have other errors for that GPPL file, try to fix them and check the program again. In most cases this will solve the problem.

◆ Error 385 : Unrecognized statement

Description: GPPtool encountered a statement which could not be classified.

Recovery : Check for misspelling of line's keywords. Check also if a statement appears out of the procedure body.
E.g: an 'ENDIF' keyword without an earlier 'IF' keyword. A frequent mistake is to write statements out of a procedure body (i.e: after an 'ENDP' statement, and before the start of the next procedure).

◆ Error 386 : Variable 'xxx' is not numeric

Description: GPPtool requires a numeric (not integer or logical) value.

Recovery : Check the expression and convert it to numeric.

◆ Error 387 : Argument of function out of range

Description: One of the arguments for GPPL internal functions is out of its legal range.

Recovery : Check in chapter 5.6 for legal values of functions' arguments.

◆ Error 388 : ENDP statement missing

Description: GPPtool found the beginning of a procedure before it found the 'ENDP' statement of the previous one.

Recovery : Correct the problem and re-run.

◆ Error 389 : Invalid ASSIGNMENT statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 390 : Invalid GENERATE statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 391 : Invalid CALL statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 392 : Invalid IF statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 394 : Variable 'xxx' is not string variable

Description: GPPtool requires a string variable (not an integer or numeric one).

Recovery : Check the expression and convert it to string.

◆ Error 395 : Literals table overflow

Description: Too many literals (string constants) are defined for that statement.

Recovery : This error rarely appears. Try to simplify the line that caused the problem (i.e: split it for two lines if possible).

◆ Error 396 : Proc '@xxx' already defined

Description: You declared more than one GPPL procedure with the same name '@xxx'.

Recovery : Check for misspelling and rename one procedure.

◆ Error 397 : Local table 'xxx' cannot be recovered

Description: GPPtool internal error.

Recovery : Try to solve other error messages (if any) and re-run the program. If the error still exists, call CADTECH for technical support.

◆ Error 398 : Invalid seek

Description: GPPtool could not find the proper sector of a disk file.

Recovery : Probably you ran out of disk space;free some space and re-run the program.

◆ Error 399 : Cannot open file 'xxx'

Description: GPPtool could not find file 'xxx'.

Recovery : Check the existance of the file in the current working directory.

◆ Error 400 : Invalid GLOBAL statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 401 : Invalid LOCAL statement

Description: Check the syntax.

Recovery : Correct the problem and re-run.

◆ Error 402 : Symbol 'xxx' cannot be modified

Description: You tried to assign a value into a non-modifiable system variable.

Recovery : It is forbidden to assign a value into a GPPL non-modifiable variable. It makes no sense to change 'pi' value for example.

◆ Error 403 : Symbol table is full

Description: GPPL symbol table ran out of its size.

Recovery : Try to reduce the number of global/local variables. Sometimes the problem arises because GPPtool ran out of memory. Quit and re-run **SolidCAM**.

◆ Error 404 : Symbol 'xxx' does not exist

Description: You used an unrecognized variable named 'xxx' in your program.

Recovery : Check for misspelling. If OK, declare that variable as global or local.

◆ Error 405 : Stack overflow

Description: GPPtool run-time stack ran out of memory.

Recovery : Try to reduce stack requirements: reduce the number of nested 'IF' statements or reduce the number of CALLs levels. This error rarely occurs.

◆ Error 406 : Stack underflow

Description: GPPtool internal error.

Recovery : Try to solve other error messages (if any) and re-run the program. If the error still exists, call CADTECH for technical support.

◆ Error 407 : Zero divide

Description: You tried to divide by zero.

Recovery : Correct the problem.

◆ Error 408 : Format: without any digit

Description: A display format does not contain an <integer> part.

Recovery : See chapter 5.5.10 for proper display format and correct the problem.

◆ Error 409 : Format: scale factor invalid

Description: A display format scale factor is invalid.

Recovery : See chapter 5.5.10 for proper display format and correct the problem.

◆ Error 410 : Val: number out of range

Description: You tried to generate a number which is larger than 1.E100.

Recovery : Correct the problem.

◆ Error 411 : Global variable 'xxx' is multiply defined

Description: A global variable 'xxx' is multiply declared with different types.

Recovery : Check for misspelling. It is impossible to declare a variable with two different types.

◆ Error 412 : Local variable 'xxx' is multiply defined

Description: A local variable 'xxx' is multiply declared with different types.

Recovery : Check for misspelling. It is impossible to declare a variable with two different types.

- ◆ Error 415 : Variable 'xxx' is already declared as local
Description: You declared a local variable 'xxx' as global one.
Recovery : Check for misspelling. It is impossible to declare a variable as local and global together.

- ◆ Error 416 : Variable 'xxx' is already declared as global
Description: You declared a global variable 'xxx' as local one.
Recovery : Check for misspelling. It is impossible to declare a variable as local and global together.

- ◆ Error 418 : System variable 'xxx' cannot be declared
Description: You tried to declare system variable 'xxx' as global or local.
Recovery : Rename variable and re-run.

- ◆ Error 420 : Numeric expression is not of logical type
Description: A numeric expression does not yield a TRUE/FALSE (1/0) value.
Recovery : Correct the problem and re-run the program.

- ◆ Error 421 : Format: no decimal point
Description: A display format does not contain a decimal point.
Recovery : See chapter 5.5.10 for proper display format and correct the problem.

- ◆ Error 422 : Format: no fraction digits
Description: A display format does not contain a <fraction> part.
Recovery : See chapter 5.5.10 for proper display format and correct the problem.

- ◆ Error 423 : Format: Invalid format option

Description: A display format contains an invalid option.

Recovery : See chapter 5.5.10 for proper display format and correct the problem.

◆ Error 424 : Cannot initialize GPPtool environment

Description: An error occurred while initializing the GPPtool environment.

Recovery : Correct other error messages. When running GPPtool outside **SolidCAM**: check if 'CONFIG.NC' file contains proper machine name, and if [MACHINE.MAC] file contains proper 'post_processor' parameter value.

◆ Error 425 : Cannot terminate GPPtool environment

Description: Error occurred while terminating GPPtool environment.

Recovery : Internal error. Call CADTECH for technical support.

◆ Error 426 : Error while reading file 'xxx'

Description: I/O error occurred while reading file 'xxx'.

Recovery : Probably hardware error. Try to re-run the program.
If the problem still occurs, try to change the magnetic media (i.e: try to run it on another disk).

◆ Error 427 : Error while writing file 'xxx'

Description: I/O error occurred while writing file 'xxx'.

Recovery : Probably hardware error. Try to re-run the program.
If the problem still occurs, try to change the magnetic media (i.e: try to run it on another disk).
If working with diskettes, check that the diskette is not a 'read-only' one.

◆ Error 452 : Block number field too big

Description: While sequencing G-code blocks, 'blknum_max' value was reached.

Recovery : If possible, increase 'blknum_max' value (in procedure '@init_post'). Otherwise, split the Part into smaller parts and re-generate the G-Code.