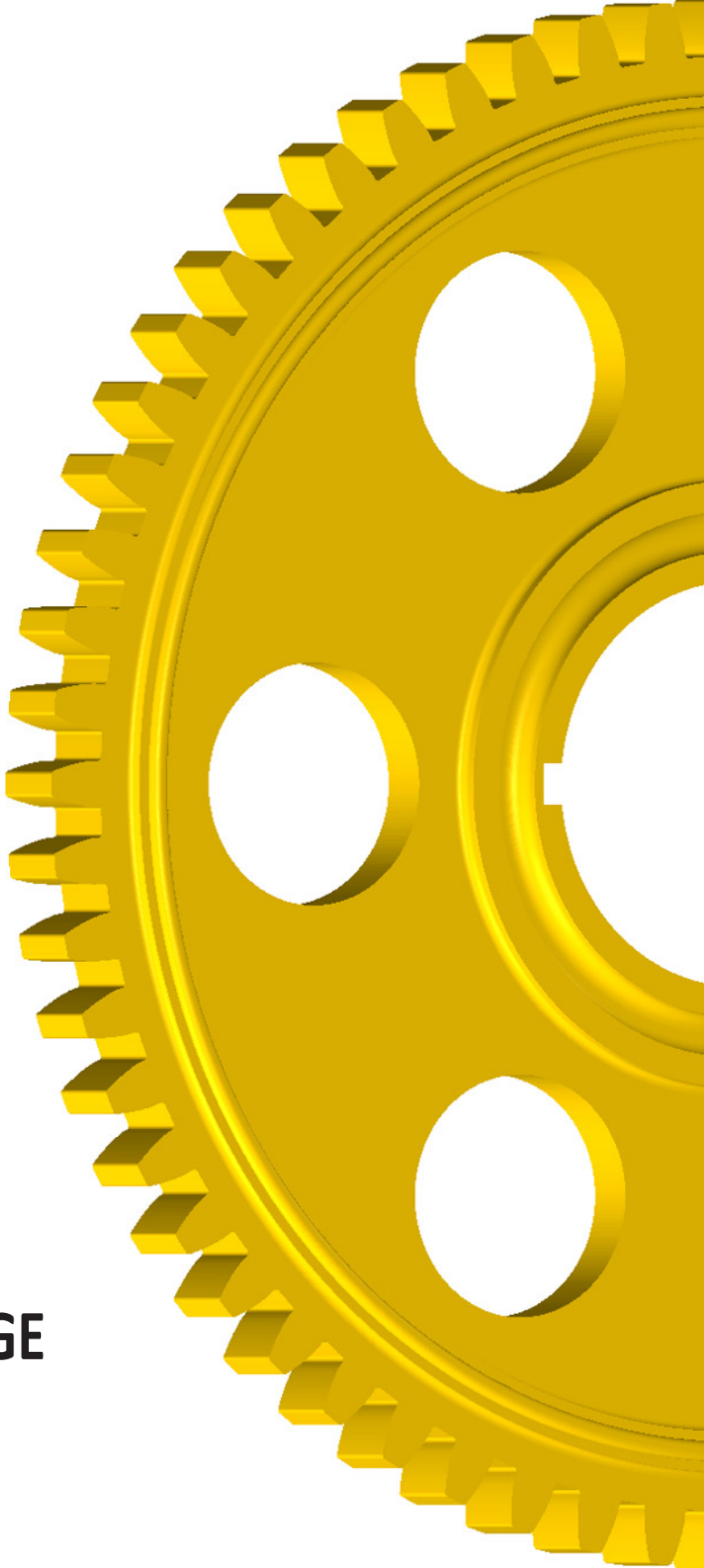
Table of Contents

MACRO LANGUAGE REFERENCE



CHAPTER 1: *Macro Language Reference*

THE GIBBSCAM MACRO LANGUAGE

ABOUT THE MACRO FUNCTION

GibbsCAM has a macro function that can be used to create geometry, import saved processes, and create operations. The intended use of the GibbsCAM Macros function is for family of parts programming. While you will likely find uses for the macros function beyond programming family of parts setups, and more functionality will be added as time goes on.

ABOUT THIS MANUAL

This document is a reference guide for the macro language and its use. This document is designed with the assumption that you are proficient with GibbsCAM and have some familiarity with programming macros or have general programming experience.

ABOUT THE MACRO LANGUAGE

GibbsCAM macros are text files which are interpreted at run-time. There is no compiler stage for GibbsCAM macros and the language used is similar to Basic. All macro files must be plain ASCII text files. You may use Notepad or the text editor of your choice for creating macros. The macro files may be saved anywhere as they are run from a menu in GibbsCAM. You add macros to the menu using the macro plug-in.

User input is available through a simple prompt for a single value, or a user defined dialog (a .DLG file) that may contain input boxes, check boxes and radio buttons.

Following is some general information about the GibbsCAM Macros and the syntax of the macros.

- GibbsCAM Macros are not case dependent.
- Tabs are not recognized, indenting must be done with spaces.
- There must be a comma and a space between arguments in a command.
- Angle values may be in either radians or degrees.
- Macros can call other macros.

LANGUAGE DESCRIPTION

VARIABLES

GibbsCAM Macro variables do not need to be, but can be declared. If a variable is not declared then it is automatically defined when it is used and is global. All numeric variables are double precision floating point numbers ("floats").

GLOBAL <VARIABLE NAME>, <VAR NAME> ... , <VAR NAME>: Global variables are available to all macros from the first run in an instance. Global variables do not need to be declared unless they are arrays. If a variable is not declared, i.e. specifically stated as "global" or "local" then it is considered global.

```
GLOBAL XVAL(10), YVAL(5), ABC, DEF$
```

In this example of an array "XVAL" is an array with 10 values, "YVAL" holds 5 values, "ABC" is a single numeric value and "DEF$" holds a single string of text characters.

LOCAL <VARIABLE NAME>, <VAR NAME> ... , <VAR NAME>: A local variable is only available to the particular macro in which it was defined. Variables that are local to a macros must be declared before they are used. When you exit the macro the variable is lost.

ARGS <VARIABLE NAME>, <VAR NAME> ... , <VAR NAME>: When you call another macro, you can pass values or variables to that macro. Inside the macro that is called, you must declare these values using the ARGS command. This will define variables that are local to the called macro that will hold the values passed to that macro.

```
CALL "example.mac", 1, ABC
```

Inside the "example.mac" macro:

```
ARGS NUM, COUNTER
```

In the "example.mac" macro, the local variable NUM will start with the value 1 (the first value passed to it) and the local variable COUNTER will hold the value of the ABC variable in the calling macro. When the "example.mac" macro is finished, it will pass back the current value of COUNTER into the variable ABC.

FLOW CONTROL

GOTO <LABEL NAME>: Goto instructs the computer to jump to another point in the macro, specified by a label name.

CALL <MACRO FILENAME>: Call instructs the macro to refer to another macro, specified by the name of the macro to be referenced.

FUNCTIONS, OPERATORS & EXPRESSIONS

Math

By default all trig functions work in radians, not degrees. You can change this during a macro by using the command "degrees", the command "radians" will switch the values back to radians. In the description of the functions angled brackets (< >) indicate any valid expression that evaluates to a number.

The GibbsCAM Macros use the standard mathematical operators, i.e. +, -, /, *, <, > and =. Expressions follow the standard order of operations. Parentheses are optional but are highly recommended to ensure calculation is correct and to maximize readability.

****:** This operator will raise a value to a power.

```
a = x**3
```

<>: This operator specifies "not equal to".

```
if dir<>1 then message "Rotate"
```

INT <NUMBER>: When this command is placed in front of a value or numeric variable the number will be truncated to an integer.

SQRT <NUMBER>: Return the square root of the given number.

ABS <NUMBER>: Return the absolute value of the given number.

SIN <NUMBER>: Return the sine of the given number. By default the result is in radians.

```
x= r1 * sin(a)
```

COS <NUMBER>: Return the cosine of the given number. By default the result is in radians.

```
x= r1 * cos(a)
```

TAN <NUMBER>: Return the tangent of the given number. By default the result is in radians.

```
x= r1 / tan(a)
```

ASIN <NUMBER>: Compute the arc sine of the given number. By default the result is in radians.

ACOS <NUMBER>: Compute the arc cosine of the given number. By default the result is in radians.

ATAN <NUMBER>: Compute the arc tangent of the given number. By default the result is in radians.

DEGREES: This command will switch all trig functions to work in degrees.

RADIANS: This command will switch all trig functions to work in radians.

Strings

About Strings

Concatenation

Most macro operators (-, *, /, etc) don't operate on strings. The exception is +, which acts as a concatenation operator.

```
var$ = "a little bit of text"
msg$ = "This string and " + var$
```

In this example the msg will be set to "`This string and a little bit of text`".

Embedding Variables

Variables can be embedded in strings at runtime by using the "%variable" syntax.

```
a = 3
b = 4
s = a * a + b * b
c = sqrt(s)
Message "The hypotenuse of a right triangle with sides of length %a and %b
is %c."
```

In this example the message will be set to "`The hypotenuse of a right triangle with sides of length 3 and 4 is 5.`"

Special Characters

Linebreaks in strings may be represented with the "`\n`" special character sequence.

```
msg$ = "this\nand that"
```

In this example the message will be set to:

```
this
and that
```

Certain characters may not be included in string literals. At this point, the following characters will cause errors: percent sign (%), double quote mark ("), and exclamation point (!).

Commands With Strings

Commands that handle strings follow.

LEN <STRING>: Return the length of a string.

```
Ilen = len(“abc”)
```

ilen will be set to 3

```
Ilen = len(a$)
```

ilen will be set to the number of characters in the variable a$

LEFT$ <STRING>, <NUMBER OF CHARACTERS>: Return the leftmost number of characters from a string.

```
B$ = left$(“ABCDE,” 3)
```

B$ will be set to “ABC”

RIGHT$ <STRING>, <NUMBER OF CHARACTERS>: Return the rightmost number of characters from a string.

```
B$ = right$(“ABCDE,” 3)
```

B$ will be set to “CDE”

MID$ <STRING>, <FIRST CHARACTER>, <NUMBER OF CHARACTERS>: Return the characters from the middle of a string.

```
B$ = mid$(“ABCDE,” 3, 2)
```

B$ will be set to “CD”

LTRIM$ <STRING>: Return the string with any leading spaces removed

RTRIM$ <STRING>: Return the string with any trailing spaces removed

TRIM$ <STRING>: Return the string with any leading and trailing spaces removed

UCASE$ <STRING>: Return the string with any characters converted to upper case

LCASE$ <STRING>: Return the string with any characters converted to lower case

FMT$ <STRING>, <FORMAT>: Return a numeric value, formatted as a text string. The format may contain:

0	A digit
#	A digit if it is not a leading or trailing zero
.	A decimal point
+	A plus sign if the number is positive

```
B$ = fmt(123.45, ”0000.000”)
```

B$ will be set to “0123.450”

CONDITIONAL LOGIC

IF <EXPRESSION> THEN <EXPRESSION>: The if/then loop is fairly standard. A basic logic command has the following syntax.

```
IF <value 1> <condition> <value 2> THEN <macro command>
```

Several examples of logic commands follow.

```
IF a = b THEN c = d

IF (a+2) > (b * cos(c+d/2)) THEN call macro2

IF abc <> 3 THEN goto label10

IF messages=1 THEN IF dir<>1 THEN message "Rotate"
```

FOR <VARIABLE> = <START> TO <END>, [STEP <INTEGER>]: For loops are very straight forward. The STEP parameter has a default value of 1 so you do not need to declare the STEP if the increment is 1. Several examples of FOR loops follow.

```
FOR I = 1 TO 10
   A = A + 1
NEXT I

FOR I=2 TO 6 STEP 2
   FOR J = (I+1) TO 2
      A = I + J
```

```
   NEXT J
NEXT I
```

The first example increments "A" by 1 ten times. The second example has a FOR in a FOR.

NEXT <VARIABLE NAME>: The next command modifies the FOR variable by the specified step and executes the contents of the loop again.

CONTINUE: The continue command, when placed inside a loop, stops executing the current iteration of a loop and returns to the top of the next iteration of the loop.

```
for i = 1 to 3
   message "This message will appear three times."
   if i = 2 then continue
   message "This message will appear twice, for i = 1 and 3; it is bypassed
   for 2"
next i
```

COMMANDS

Commands that require arguments use a comma as a delimiter between each argument. In the description of the commands angled brackets (< >) indicate any valid expression that evaluates to a number. Square brackets ([ ]) indicate optional arguments. The pipe character (|) indicates individual valid parameter entries such as "CW | CCW", which indicates clockwise or counter clockwise and "0 | 1" which indicates either "true or false" or "on or off". Quite a few commands described here have the parameters "Left" and "Top". The "Left" parameter describes the distance from the left edge of GibbsCAM or the dialog the parameter is in. The "Top" parameter describes the distance from the top edge of GibbsCAM or the dialog the parameter is in.

Functions With String Return Values

Most macro commands have no return value or return a floating-point number. There are, however, a number of commands which return a string. These functions are indicated by a $ after the command name. These commands are documented in the "Commands With Strings" section of this manual. These functions must be called with parentheses around the parameters.

```
msg$ = " AB CD "
msg2$ = rtrim$(msg$)
```

In this example the variable msg2 will be set to "AB CD"

User Input Commands

INPUT "PROMPT", <VARIABLE>, <DEFAULT>: This command will display a dialog with an input box for the user to enter a value and is called from within the macro file. Please note that this command is different than the `input` command used within a dialog, which is documented on page 11.

```
INPUT "Enter Width", w1, 10
```

The results of this code is a dialog with an input box that prompts the user to input a value with a default of "10". This value will be assigned to the variable `w1`.

YESNO "PROMPT TEXT", <VARIABLE>, ["CAPTION"]: This command calls a dalog with "Yes" and "No" buttons. The variable gets set to "0" for "No" and "1" for "Yes". If a caption is not set then the top of the dialog will simply say "Custom Macro".

```
YESNO "Do you want to do this?", doit1, "You need to make a choice."
```

This will display a dialog with the caption "You need to make a choice" and the message "Do you want to do this?". There will be YES and NO buttons. If you press "yes", the variable doit1 will be set to 1, if you press "no" or close the dialog, it will be set to 0.

DIALOG "FILENAME": You can set up a technique to get input through a custom dialog. When run this command will open the specified dialog.

```
DIALOG "MyMacro.dlg"
```

Dialog Creation Commands

As stated in the "dialog "filename"" section, you may collect input from a user by creating a custom dialog. The dialog must be defined as a separate text file. A dialog text file must start with the "dialog "caption", <left>, <top>, <width>, <height>" command. A dialog may then contain any combination of text labels, input boxes, check boxes and radio buttons. There may be up to 10 each of the controls. A dialog must also contain an "Okay" and "Cancel" button. A dialog may also contain an optional bitmap image.

DIALOG "CAPTION", <LEFT>, <TOP>, <WIDTH>, <HEIGHT>: This command defines the left and top position of a dialog followed by its width and height. This command must be the first command in a dialog text file.

```
DIALOG "Diamond Insert" 30, 50, 530, 390
```

This creates a dialog called Diamond Insert that is 530x390 pixels which will open 30 pixels from the left and 50 pixels from the top of the GibbsCAM window.

FONT <NAME>, <SIZE>: By default the Macros function uses the default system font for all text in a macro. This command lets you specify the font used. If the font specified is not installed on the user's system then the macro will use the system default font.

FRAME <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, "TEXT": This command lets you place a frame within a dialog that can include a text label. A frame is useful for organizing groups of controls.

```
FRAME 20, 240, 250, 100 "Angle Definition"
```

This creates a frame within a dialog that has a label that says Angle Definition. The frame is 250x100 pixels and is offset by 20 pixels from the left and 240 pixels from the top of the dialog.

LABEL "TEXT", <LEFT>, <TOP>, <WIDTH>, <HEIGHT>: This command lets you place a text label in a dialog. You define the left and top position within the dialog, followed by the size of the text, followed by what the text is. Each label you create is automatically given an incrementing name, i.e. the first label in the dialog is referred to as "label1", the second label is "label2", etc.

```
LABEL "Enter a value" 310, 39, 100, 24
```

This creates text that says Enter a value. The text is offset by 310 pixels from the left and 39 pixels from the top of the dialog. The text has a maximum area of 100x24 pixels. If the text takes up more space than this it will be clipped.

INPUT <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, "CAPTION", <VARIABLE>, [<DEFAULT>]: This command lets you place a text input box in a dialog. You define the left and top position within the dialog, followed by the size of the box, followed by a variable name and, optionally, a default value. Each input you create is automatically given an incrementing name, i.e. the first input in the dialog is referred to as "input1", the second input is "input2", etc.

```
INPUT 420, 35, 70, 24, "# of Degrees", a1, 10
```

This creates a text entry box that is offset by 420 pixels from the left and 35 pixels from the top of the dialog. The box is 70x24 pixels. Whatever the user types into this box will be assigned to the variable "a1". This box has a default value of "10".

CHECK <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, "LABEL", <VARIABLE NAME>, [<0|1>]: This command lets you place a checkbox within a dialog. You define the left and top position within the dialog, followed by the size of the checkbox, followed by the text that will appear, followed by a variable name and lastly by the default state. "0" states that the checkbox is off, "1" states that the checkbox is on. If a default setting is not used the

checkbox s automatically set off. Each check box you create is automatically given an incrementing name, i.e. the first check box in the dialog is referred to as “check1”, the second check box is “check2”, etc.

```
CHECK 420, 35, 20, 20 “Ooh! Ooh! Pick me! Pick me!” check1, 1
```

RADIO <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, “LABEL”, <VAR NAME>, [<0|1>, <GROUP #>]: This command lets you place a radio button within a dialog and you can group them together. You define the left and top position within the dialog, followed by the size of the button, followed by the text that will appear, followed by a variable name. Optionally you can specify whether the item is off (`0`, which is the default) or on (`1`) and assign the button to a group, which is any integer value. Each radio button you create is automatically given an incrementing name, i.e. the first radio button in the dialog is referred to as “radio1”, the second radio button is “radio2”, etc.

```
RADIO 40, 260, 200, 24 "Face Relief", opt1, 1, 1
RADIO 40, 285, 200, 24 "Diameter Relief", opt2, 0, 1
```

This code creates two radio buttons, the first is selected the second is not. The buttons are inset from the dialog by 40 pixels and are 260 and 285 pixels from the top of the dialog. both occupy a 200x24 area, which includes label text. The first button sets the variable “opt1” and the second button sets the variable “opt2”. Face Relief is enabled by default and both radio buttons are in group 1. See the “ON_EVENT” command for an example of toggling radio buttons.

IMAGE <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, <FILENAME>: This command lets you place an image within a dialog. You define the left and top position within the dialog, followed by the size of the image, followed by the name of the image. The name may be simply the name of the file or it may be the entire path to the file. The image must be a bitmap (.BMP) file. Each image you create is automatically given an incrementing name, i.e. the first image in the dialog is referred to as “image1”, the second image is “image2”, etc.

```
IMAGE 20, 20, 250, 200, "UpLeft.bmp"
```

This code places the image “UpLeft.bmp” 20 pixels from the left and 20 pixels from the top of the dialog. The image is given an area of 250x200 pixels.

OK <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, [“CAPTION”]: This command lets you place an “Okay” button within a dialog. You define the left and top position within the dialog, followed by the size of the button. The `CAPTION` option lets you change the text on the button.

```
OK 420, 316, 70, 24
```

This code creates an "OK" button that is 70x24pixels. The button is 420 pixels from the left and 316 pixels from the top of the dialog.

CANCEL <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, ["CAPTION"]: This command lets you place a "Cancel" button within a dialog. You define the left and top position within the dialog, followed by the size of the button. The `CAPTION` option lets you change the text on the button.

```
CANCEL 310, 316, 70, 24
```

This code creates an "Cancel" button that is 70x24pixels. The button is 310 pixels from the left and 316 pixels from the top of the dialog.

BUTTON <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, "LABEL", <VALUE>: This command lets you place a custom button within a dialog. You define the left and top position within the dialog, followed by the size of the button, followed by the text that will appear. This command defines a variable called "button" that can be used to pass the value.

```
BUTTON 250, 100, 40, 20, "Do it" 1
```

This creates a 40x20 pixel button labelled "Do it" that sets a value of `1` when the button is pressed.

ON_EVENT <CONTROL NAME>, <CONTROL NAME>, <"ENABLE" | "DISABLE" | "SHOW">: This command is used to control the enabling and disabling of controls. For example when a radio button is selected the others should deselect. Following are two examples of this command's use. The first example illustrates displaying an image when a radio button is selected. The second example shows the toggling action between radio buttons.

```
ON_EVENT radio3, image3, SHOW

ON_EVENT radio1, input1, ENABLE
ON_EVENT radio1, input2, DISABLE
ON_EVENT radio2, input1, DISABLE
ON_EVENT radio2, input2, ENABLE
```

We see two different uses of the on_event command. The first ("on_event radio3, image3, show") will display image 3 when radio button 3 is clicked. The second example shows interactivity of radio buttons. When radio button 1 is clicked "input1" will be enabled and "input2" will be disabled. When radio button 2 is clicked "input1" will be disabled and "input2" will be enabled.

DROPDOWN_NEW <DROPDOWN #>, <LEFT>, <TOP>, <WIDTH>, <HEIGHT>, <VARIABLE>: This command lets you create a dropdown menu within a dialog. The arguments <left> and

<top> specify the where the top left corner of the menu will be placed in the dialog. The <height> argument is the total height of the menu when dropped down, which is different than other uses <height>, therefore the value should be fairly large to accommodate multiple entries in the menu. The <variable> argument sets the name the dropdown menu's value. You may define up to twenty menus in a dialog.

```
DROPDOWN_NEW 1, 20, 20, 250, 100, drop1
```

This creates dropdown menu #1 20 pixels in from the top and left of a dialog. The menu is 250 pixels wide and 100 pixels tall when deployed. The selection will be assigned to the variable `drop1`.

DROPDOWN_ADD <DROPDOWN #>, <CAPTION>, <VALUE>: This command puts an entry into a dropdown menu.

```
DROPDOWN_ADD 1, "Create a Point", 10
DROPDOWN_ADD 1, "Create a Line", 20
DROPDOWN_ADD 1, "Create a Circle", 30
```

This dropdown menu has three entries. If the "Create a Circle" item is selected from the menu, the variable is set to "`30`".

DROPDOWN_VAL <DROPDOWN #>, <VALUE>: This is an optional command that lets you specify a pre-selected entry in the dropdown list. If this command is not used then the menu will automatically display the first item defined by "dropdown_add".

```
DROPDOWN_VAL 1, 20
```

The dropdown we are defining will, by default, display the entry with the value "20".

DROPDOWN_EXCEL <DROPDOWN #>, <RANGE NUMBER>: This command maps an Excel range, which is set by using the "excel_get_range" command. The "excel_get_range" command is used in the macro which calls the dialog. See the Excel macro sample file for an example of this command's use.

LOAD_DEFAUTS: This command has no arguments. When the dialog is closed, it will save all of the dialog values in a file that has the same name as the macro filename, but ending with "_in.ini" or "_mm.ini" depending on whether the current units are inch or metric.

SAVE_DEFAULTS: This command has no arguments. It will set all of the dialog variables to the values that they contained when the dialog was last used. The values will be read from the file previously saved using the SAVE_DEFAULTS command.

COMMENTS

! Comments are noted by a leading exclamation point. To make a multi-line comment be sure to place an exclamation point at the start of each line. Comments can be placed in-line with commands but anything following the exclamation point will not be read by the macro.

```
RADIO 40, 260, 200, 24, "Face Relief", opt1, 1 ! 1 = default
```

Here we see a radio button being defined. At the end of the line there is a comment.

LABELS

LABELS: A label is used as a pointer for references. A label is designated as a line of text with a colon (:) at the start of the line.

```
if <value 1>=<value 1> goto option2
(...)
:option2        ! dia relief angle
a1 = 90-a2-a3
a4 = a1+(a2/2)
```

Here we see a logic test and if the test is true the macro will go to “option2”. Further down the macro we find “:option2” which is the label for that section of the code. There is a comment to help us keep track of what the code does. The last two lines are the actual code for option2.

GIBBS PART MANIPULATION COMMANDS

File Handling Commands

NEW_PART: This command will start a new file. This command has no arguments.

OPEN_PART “FILENAME”: This command will open an existing file. If the file resides in the same directory as the macro then the name does not need to be in quotes. Alternatively the filename can be a variable.

```
open_part “c:\Gibbs Parts\My Cool Macro-generated Part.vnc”
```

SAVE_PART: This command will save the current part. This command has no arguments.

SAVE_PART_AS <“FILENAME”>: This command will save the current part with a new filename specified or data from a variable. The name of the file must be specified within quotation marks.

FILE_DIALOG_NEW <CAPTION>: This command creates a new Windows Common Dialog with the given caption.

FILE_DIALOG_EXTENSION <DESCRIPTION>, <EXTENSION>: This command adds a value to the filename filter part of the dialog box.

FILE_DIALOG_SHOW OPEN|SAVE <VARIABLE NAME>: This command shows the created dialog in "Open" or "Save" mode.

```
FILE_DIALOG_NEW "Open GibbsCAM File"
FILE_DIALOG_EXTENSION "GibbsCAM Part Files (*.vnc)", "*.vnc"
FILE_DIALOG_EXTENSION "All Files (*.*)", "*.*"
FILE_DIALOG_SHOW OPEN, filename$
OPEN_PART filename$
```

Part Data

GET_PART_DATA <PARAMETER>, <VARIABLE NAME>: This command is used to get part specific data. A list of the available parameters is provided in the section "Part Data" on page 40.

SET_PART_DATA <PARAMETER>, <VALUE>: This command is used to set part specific data. A list of the available parameters is provided in the section "Part Data" on page 40.

GET_SPINDLE_NUM <VARIABLE NAME>: This command is used to get the number of the currently selected spindle.

SET_SPINDLE_NUM <SPINDLE NUMBER>: This command is used to set the number of the currently selected spindle.

GET_MTM_DATA <PARAMETER>, <VARIABLE NAME>: This command is used to get part specific data that is unique to MTM. A list of the available parameters is provided in the section "MTM Setup Data" on page 40.

SET_MTM_DATA <PARAMETER>, <VALUE>: This command is used to set part specific data that is unique to MTM. A list of the available parameters is provided in the section "MTM Setup Data" on page 40.

Post Commands

GET_POST_DATA <PARAMETER>, <VARIABLE NAME>: This command is used to get data from the posting dialog. A list of the available parameters is provided in the section "Post Data" on page 51.

SET_POST_DATA <PARAMETER>, <VALUE>: This command is used to set data in the posting dialog. A list of the available parameters is provided in the section "Post Data" on page 51.

RUN_POST [<POST FILENAME>, <OUTPUT FILENAME>]: This command is used to run a post.

Geometry Creation Commands

The GibbsCAM Macro function has the ability to create geometry. When creating a point, line or circle you should specify the feature's number. These numbers are created within the macro memory, not within GibbsCAM. This is done so that the feature may be selected. The system allows for up to 100 points, 100 lines and 100 circles.

A Note About The <Option> Argument

Quite a few of the geometry creation commands have the <option> argument. This argument is present in commands that can result in more than one solution, such as a line tangent to a circle at a specific angle, which has two solutions. The <option> argument provides for which solution will be chosen.

Each solution is calculated mathematically, so by trial and error you can see which one you will need. For example, intersecting a line and circle to get a point will have 2 options. option 1 will always be the first intersection as you look along the direction of the line. Intersecting 2 circles will have 2 solutions. Option 1 will always be the same side of the line from the center of circle 1 to the center of circle 2.

Points

POINT <X>, <Y>: This command will create a point at the given position.

```
POINT x1, y1
POINT 10, 15
```

We see two different examples here. The first uses variables; a point will be created at the position specified by variables from user input. The second example uses a hard-coded value; a point will be created at x10 y15. The values are in part units.

CREATE_POINT <POINT NUMBER>: Points, lines and circles can be created internally in a macro that are in memory but not drawn. These geometry features are used to calculate geometry, e.g. so you can create a point at the intersection of a line and circle. This command takes a point definition that is in memory and creates an actual point at that location.

POINT_XY <POINT NUMBER>, <X>, <Y>: This command will create a point at the given position.

POINT_CA <POINT NUMBER>, <CIRCLE>, <ANGLE>: This command will create a point on a circle at an angle.

POINT_2L <POINT NUMBER>, <LINE>, <LINE>: This command will create a point at the intersection of two lines.

POINT_LC <POINT NUMBER>, <LINE>, <CIRCLE>, <OPTION>: This command will create a point at the intersection of a line and circle.

POINT_2C <POINT NUMBER>, <CIRCLE>, <CIRCLE>, <OPTION>: This command will create a point at intersection of 2 circles.

POINT_2P <POINT NUMBER>, <POINT>, <POINT>: This command will create a point between two specified points.

POINT_GET_DATA <POINT NUMBER>, <X>, <Y>: This command acquires the data for the point specified.

```
point_get_data 10, xx, yy
```

This example shows us getting the X and Y coordinates for point #10. The coordinate data is put into the variables xx and yy.

POINT_COPY <POINT NUMBER>, <NEW POINT NUMBER>: This command will create a copy of the specified point. The copied point is given a number.

```
point_copy 1, 99
```

This example shows point 1 being duplicated and the new point being set as point 99.

POINT_TRANSLATE <POINT NUMBER>, <DX>, <DY>: This command will move the specified point by an amount in X and Y.

```
point_translate 1, 3, -5
```

This example shows point 1 being moved by 3 part units in X and -5 part units in Y.

POINT_ROTATE <POINT NUMBER>, <XC>, <YC>, <ANGLE>: This command will rotate the point about a specified location at an angle. Negative values are valid.

```
point_rotate 1, 0, 0, 45
point_rotate 2, -3, -3, -45
```

This example shows point 1 being rotated about XoYo by 45 degrees and point 2 being rotated about X-3Y-3 by -45 degrees.

POINT_MIRROR <POINT NUMBER>, <AXIS>, <VAL>: This command will reflect the specified point across the specified axis at a particular value.

```
point_mirror 1, x, 0
point_mirror 2, y, 5
```

This example shows point 1 being reflected across the X axis and point 2 being reflected across Y5, from its present position.

POINTS: This command takes no arguments This command lets you can create a group of pre-selected points. Within brackets you define the various points in the group using any combination of the point creation commands.

```
POINTS [
POINT 1, 2
POINT 3, 2
POINT 3, 5
]
```

After these commands there will be 3 points, [at (1,2), (3,2) and (3,5)] and they are all selected. At this point you can load a drilling process to create a drilling operation.

Lines

LINE_2P <LINE NUMBER>, <POINT>, <POINT>: This command will create a line though 2 points.

```
line_2p 1, 3, 4
line_2p 2, varp1, varp2
```

The first example shows line #1 being created through point 3 and point 4. The second example shows line #2 being created through points that are variables.

LINE_HP <LINE NUMBER>, <POINT>: This command will create a horizontal line though a point.

```
line_hp 5, 1
```

This example shows horizontal line #5 being created through point #1.

LINE_VP <LINE NUMBER>, <POINT>: This command will create a vertical line though a point.

```
line_vp 5, 1
```

This example shows vertical line #5 being created through point #1.

LINE_PA <LINE NUMBER>, <POINT>, <ANGLE>: This command will create a line though a point at an angle.

```
line_pa 2, 1, 30
```

This example shows line #2 being created through point #1 at 30 degrees.

LINE_PC <LINE NUMBER>, <POINT>, <CIRCLE>, <OPTION>: This command will create a line though a point tangent to a circle.

LINE_CA <LINE NUMBER>, <CIRCLE>, <ANGLE>, <OPTION>: This command will create a line tangent to a circle at a specific angle.

LINE_2C <LINE NUMBER>, <CIRCLE>, <CIRCLE>, <OPTION>: This command will create a line tangent to two circles.

LINE_LD <LINE NUMBER>, <LINE>, <DISTANCE>: This command will create a line offset from another line at a specified distance.

LINE_COPY <LINE NUMBER>, <NEW LINE NUMBER>: This command will create a copy of the specified line. The copied line is given a number.

```
line_copy 1, 99
```

This example shows line 1 being duplicated and the new line being set as line 99.

CREATE_LINE <LINE NUMBER>: Points, lines and circles can be created internally in a macro that are in memory but not drawn. These geometry features are used to calculate geometry, e.g. so you can create a point at the intersection of a line and circle. This command takes a line definition that is in memory and creates an actual line at that location.

LINE <X1>, <Y1>, <X2>, <Y2>: This command has been deprecated. Use one of the other line creation commands instead.

Circles

CIRCLE <XC>, <YC>, <RADIUS>: This command will create a circle with a given center point and radius. Upon creation the circle is defined by CircleRef(+1).

```
CIRCLE cpx, cpy, rad1
```

```
CIRCLE x0, y0, 10
```

We see two different examples here. The first uses variables; a circle with a radius defined by rad1 will be created with its center at cpx, cpy. These variables will be determined at runtime from user input. The second example uses a hard-coded value; a 10 part unit radius circle will be created at the part origin.

CREATE_CIRCLE <CIRCLE NUMBER>: Points, lines and circles can be created internally in a macro that are in memory but not drawn. These geometry features are used to calculate geometry, e.g. so you can create a point at the intersection of a line and circle. This command takes a circle definition that is in memory and creates an actual circle at that location.

CIRCLE_CR <CIRCLE NUMBER>, <POINT #>, <RAD>: This command will create a circle using an existing point for the center point with a specified radius.

CIRCLE_CP <CIRCLE NUMBER>, <POINT1>, <POINT1>: This command will create a circle using two points, the first is the center point and the second specifies the radius.

CIRCLE_2P <CIRCLE NUMBER>, <POINT>, <POINT>: This command will create a circle using two points two define the diameter.

CIRCLE_2PR <CIRCLE NUMBER>, <POINT>, <POINT>, <RAD>, <OPTION>: This command will create a circle using two points that are on the radius with a stated radius value.

CIRCLE_2LR <CIRCLE NUMBER>, <LINE>, <LINE>, <RAD>, <OPTION>: This command will create a circle with a specified radius tangent to two lines.

CIRCLE_3P <CIRCLE NUMBER>, <POINT>, <POINT>, <POINT>: This command will create a circle through three points.

CIRCLE_PL <CIRCLE NUMBER>, <POINT>, <LINE>: This command will create a circle with the specified point at the center, that is tangent to a line.

CIRCLE_PLR <CIRCLE NUMBER>, <POINT>, <LINE>, <RAD>, <OPTION>: This command will create a circle with the specified point at the center, that is tangent to a line and has a specified radius.

CIRCLE_PC <CIRCLE NUMBER>, <POINT>, <CIRCLE>: This command will create a circle with the specified point at the center, that is tangent to a circle.

CIRCLE_PCR <CIRCLE NUMBER>, <POINT>, <CIRCLE>, <RAD>, <OPTION>: This command will create a circle with the specified point at the center, that is tangent to another circle and has a specified radius.

CIRCLE_LCR <CIRCLE NUMBER>, <LINE>, <CIRCLE>, <RAD>, <OPTION>: This command will create a circle with a specified radius that is tangent to a line and a circle.

CIRCLE_2CR <CIRCLE NUMBER>, <CIRCLE>, <CIRCLE>, <RAD>, <OPTION>: This command will create a circle with a specified radius that is tangent to two circles.

CIRCLE_GET_DATA <CIRCLE NUMBER>, <CX>, <CY>, <RAD>: This command acquires the data for the circle specified.

```
circle_get_data 10, xx, yy, rr
```

This example shows us getting the X and Y coordinates and the radius value for circle #10. The coordinate data is put into the variables `xx`, `yy` and `rr`.

CIRCLE_COPY <CIRCLE NUMBER>, <NEW CIRCLE NUMBER>: This command will create a copy of the specified circle. The copied circle is given a number.

```
circle_copy 1, 99
```

This example shows circle 1 being duplicated and the new circle being set as circle 99.

CIRCLE_TRANSLATE <CIRCLE NUMBER>, <DX>, <DY>: This command will move the specified circle by an amount in X and Y.

```
circle_translate 1, 3, -5
```

This example shows circle 1 being moved by 3 part units in X and -5 part units in Y.

CIRCLE_ROTATE <CIRCLE NUMBER>, <XC>, <YC>, <ANGLE>: This command will rotate the circle about a specified location at an angle. Negative values are valid.

```
circle_rotate 1, 0, 0, 45
circle_rotate 2, -3, -3, -45
```

This example shows circle 1 being rotated about X0Y0 by 45 degrees and circle 2 being rotated about X-3Y-3 by -45 degrees.

CIRCLE_MIRROR <CIRCLE NUMBER>, <AXIS>, <VAL>: This command will reflect the specified circle across the specified axis at a particular value.

```
circle_mirror 1, x, 0
circle_mirror 2, y, 5
```

This example shows circle 1 being reflected across the X axis and circle 2 being reflected across Y5, from its present position.

Other Geometry

CONTOUR [ ]: The macro function lets you can create a contour comprised of lines and/or circular arcs. Within brackets you define the starting position of the shape followed by the lines and arcs that define the shape. The contour definition is terminated by a].

```
contour [
start r1,    0
line  l1-r1, 0
line  l1,    h2-r2
line  l2+r2, h2
line  l2,    h1-r2
line  r1,    h1
line  0,     r1
]
```

<REF> = CONTOURREF: Once the contour definition is terminated, the variable ContourRef is set to the reference number of the contour.

```
myShape = ContourRef
```

START <X>, <Y>: This is the starting position of the contour you are defining.

```
START 0, 0
```

LINE <X>, <Y>: This is the starting position of a line in a contour shape.

```
LINE varx1, vary1

LINE 0, 0
```

We see two different examples here. The first uses variables; a line will be created that starts at the intersection of varx1 and vary1. These variables will be determined at runtime from user input. The second example uses a hard-coded value; a line will be created that starts at the part origin. The values are in part units.

ARC <XC>, <YC>, <X>, <Y>, "CW" | "CCW": This command states the centerpoint, radius defined by a position, and the direction of an arc.

```
ARC xc2, yc2, x2+tx2, y2+ty2, ccw
```

```
ARC 9, 1, 9, 0, ccw
```

We see two different examples here. The first uses variables; a counter clockwise arc whose centerpoint is the intersection of xc2, yc2, with an end point at (x2+tx2), (y2+ty2). These variables will be determined at runtime from user input. The second example uses a hard-coded value; a counter clockwise arc whose centerpoint is X9, Y1 with an end point at X9, Yo. The values are in part units.

FIT_CURVE <TOLERANCE>, <UNIFORM | CHORD_LEN | FOLEY | CENTRIPETAL>: This command will create a curve between points using one of four modes, either uniform, chord length, Foley's or centripetal. See the Geometry Creation manual for more information on the differences between these types of curves.

Geometry Selection & Transformation Commands

Geometry selection is controlled using the commands.

SELECT_GEO <REF NUM>: Select the given geometry feature, this will be added to the currently selected geometry. Once selected, geometry may be transformed using the Translate, Rotate, Scale or Mirror functions.

```
SELECT_GEO line1
```

SELECT_ALL_GEO: This command will select all geometry in the current workgroup. There are no arguments for this command. Once selected, geometry may be transformed using the Translate, Rotate, Scale or Mirror functions.

SELECT_SHAPE <REF>: This command will select a shape. The shape reference number may be the number of any feature on that shape, the GibbsCAM Macro function will automatically select all features that are connected to the given feature. Once selected, geometry may be transformed using the Translate, Rotate, Scale or Mirror functions.

```
SELECT_SHAPE myShape
```

DESELECT_ALL_GEO: This command will deselect all geometry. This is typically used before selecting shapes to ensure you do not get multiple/incorrect selections and at the end of a macro. There are no arguments for this command.

DESELECT_GEO <REF NUM>: This command will deselect the given geometry feature.

```
DESELECT_GEO line1
```

SELECT_REF <REF>: This command has been deprecated by "select_geo <ref num>".

GET_SELECTION_LIST <LIST NUMBER>: This command gets a list of all currently selected geometry and saves that list for later use. For example you may have a matrix of points selected. Creating a selection list lets you save this particular grouping for use later, such as after creating more geometry. The "get_selection_list" command saves the current selection. You may have up to 10 selection lists.

SET_SELECTION_LIST <LIST NUMBER>, [<OPTION>]: Select the geometry that was previously saved in the given selection list number. If you set the option to `1`, then any currently selected geometry will remain selected, setting it `0` (the default) will deselect all geometry before selecting the geometry in the list.

TRANSLATE_GEO <DX>, <DY>, [<DZ>, <# OF COPIES>]: This command will move the selected geometry by the given amount. Numbers may be positive or negative. Optionally you may also change the depth of the geometry and/or translate the geometry multiple times.

```
TRANSLATE_GEO 5, -2, 2
```

This command will move selected geometry by 5 part units in X, -2 units in Y and 2 units in Z. Typically this command will be used with variables whose values are set at runtime.

ROTATE_GEO <XC>, <YC>, <ANGLE IN DEGREES>, [<# OF COPIES>]: This command will rotate the selected geometry about the specified position by the specified number of degrees. Numbers may be positive or negative. Optionally you may also rotate the geometry multiple times.

```
ROTATE_GEO 5, 0, 45
```

This command will rotate the selected geometry by 45 degrees about X5, Y0. Typically this command will be used with variables whose values are set at runtime.

SCALE_GEO <FACTOR>: This command will scale the selected geometry.

```
SCALE_GEO 0.1

SCALE_GEO 25.4
```

The first example command will scale the selected geometry to 10% of its current size. The second example will scale the selected geometry to 25.4 times its current size. Typically this command will be used with variables whose values are set at runtime.

MIRROR_GEO <AXIS>, <AXIS VALUE>, [<# OF COPIES>]: This command will flip the selected geometry at the specified point along a specific axis. The "axis" parameter for the mirror

command is either X or Y. Optionally you may also create more than one copy of the mirrored geometry.

```
MIRROR_GEO X, 5

MIRROR_GEO Y, 0
```

The first example command will flip the selected shape about X5. The second example command will flip the geometry about Yo. Typically this command will be used with variables whose values are set at runtime.

DELETE_GEO <GEO REF>: Delete a geometry feature.

DELETE_SHAPE <SHAPE REF>: Delete a complete shape.

Geometry Information Commands

These commands let you acquire data from a feature.

GET_NUM_FEAT_SELECTED <VARIABLE NAME>: Get the number of features currently selected.

GET_SELECTED_GEO_REF <NUMBER>, <VARIABLE NAME>: Get the GibbsCAM feature reference number for one of the currently selected features. <NUMBER> is a value between 1 and the number of selected features.

```
GET_NUM_FEAT_SELECTED inum
FOR I = 1 to inum
   GET_SELECTED_GEO_REF  i, iref
   DEBUG iref
NEXT i
```

In the above example, assume that there were 3 features currently selected. The variable inum would be set to 3 and the macro would loop 3 times displaying the GibbsCAM feature number for the first selected feature, then the second selected feature followed by the third.

GET_FEAT_TYPE <NUMBER>, <VARIABLE NAME>: Get the type of selected feature. The variable will be set to:

1 = Point	2 = Line	3 = Circle
4 = Arc	5 = Curve	

GET_FEAT_START <GEO_REF>, <CS>, <XS>, <YS>, [<ZS>]: Get the start position data of the selected feature. If <CS> is set to `0`, the position is in the local feature CS, any other value will return the positions in the world CS.

GET_FEAT_END <GEO REF>, <CS>, <XE>, <YE>, [<ZE>]: Get the end position data of the selected feature.

GET_CIRCLE_DATA <GEO REF>, <CS>, <RAD>, <XC>, <YC>, [<ZC>]: Get the center position data of the selected feature. If the feature is not a circle, the macro will stop processing.

GET_ARC_DATA <GEO REF>, <CS>, <RAD>, <DIR>, <XC>, <YC>, [<ZC>]: Get the center position data of the selected feature. If the feature is not an arc, the macro will stop processing.

Workgroup & Coordinate System Commands

NEW_CS [XY, <NAME>, <X>, <Y>, <Z>] | [XZ, <NAME>, <X>, <Y>, <Z>] | [YZ , <NAME>, <X>, <Y>, <Z>] | [3P, <NAME>, <X1>, <Y1>, <Z1>, <X2>, <Y2>, <Z2>, <X3>, <Y3>, <Z3>]: This command creates a new coordinate system. There are four ways the CS may be defined. In each case, you provide the name for the new CS as either text inside double quotes, or a string variable that contains the CS name. GibbsCAM will allocate the CS number and this number will be saved in the variable CsNumber.

1. Based on the XY plane `[xy, <name>, <x>, <y>, <z>]` This CS will be aligned to the XY plane and its origin will be at the XYZ coordinates specified.

2. Based on the XZ plane `[xz, <name>, <x>, <y>, <z>]`. This CS will be aligned to the XZ plane and its origin will be at the XYZ coordinates specified.

3. Based on the YZ plane`[yz, <name>, <x>, <y>, <z>]`. This CS will be aligned to the YZ plane and its origin will be at the XYZ coordinates specified.

4. Based on three points `[3p, <name>, <x1>, <y1>, <z1>, <x2>, <y2>, <z2>, <x3>, <y3>, <z3>]`. This option allows you to create a CS that is aligned to three arbitrary points. The new CS's origin will be placed at the first point. The second point defines any position along the H axis and the third point is any position along the V axis.

SET_CS <CS NUMBER>: Select the given CS number.

GET_CS <VARIABLE NAME>: Get the number of the currently selected CS.

GET_CS_LIST: This command is one of three commands used to iterate through the CS list. Once called it enables the use of the number_of_css and next_cs_number commands. See

the example macro “Get CS List Info” on page 62 for a good example of this and related commands.

NUMBER_OF_CSS: This command is the second of three commands used to iterate through the CS list.This command stores a variable for the number of coordinate systems in a part.

NEXT_CS_NUMBER: This command is the third of three commands used to iterate through the CS list. This command iterates to the next CS.

GET_CS_NAME <CS NUMBER>, <VARIABLE>: This command is used to acquire the name of a particular CS. The first variable is the CS number and the second variable is the name of the CS.

SET_CS_NAME <CS NUMBER>, <VARIABLE NAME>: Get the name of the given CS number.

GET_GEO_CS <GEO REF>, <VARIABLE NAME>: Get the CS number associated with the given feature number and return the CS number in the variable.

SET_GEO_CS <GEO REF>, <VALUE>: Change the CS of a geometry feature.

GET_GEO_AIR <GEO REF>, <VARIABLE NAME>: Get the Air/Wall attribute of a geometry feature. If it is Air, the variable will be set to `1`, if not it will be set to `0`.

SET_GEO_AIR <GEO REF>, <VALUE>: Change the Air/Wall attribute of a geometry feature. 1 will set it to Air, 0 to Wall.

NEW_WG <WG NUMBER>: This command creates a new workgroup. Typically a variable will be used for the workgroup number to increment to the next WG.

SET_WG <WG NUMBER>: Set the current Workgroup to the given workgroup number.

GET_WG <VARIABLE NAME>: Get the number of the currently selected Workgroup.

GET_WG_NAME <WG NUMBER>, <VARIABLE NAME>: Get the name of the given Workgroup number.

GET_CS_SPINDLE <CS>, <VARIABLE>: This command retrieves the spindle number associated with a given CS.

Solids Commands

EXTRUDE <ZS>, <ZE>: This command will create an extruded solid from a selected closed shape. Define the starting and ending Z values, relative to the current coordinate system. The order of the values does not matter, i.e. the +Z or the -Z can be first or last.

```
extrude 10, -10
extrude -10, 10
```

The examples shown here will create identical solids, extruded by 10 part units.

REVOLVE H <VALUE>, <ANGLE> | V <VALUE>, <ANGLE>: This command will create a revolved solid from selected geometry about either the horizontal or vertical axis. The <value> parameter states the position about which the shape will be revolved and the <angle> specifies how many degrees the revolution should be, with 360 being the maximum value.

```
revolve v, 0, 360
revolve h, 1, 180
```

These examples will create a shape that is revolved 360 degrees about the vertical axis and a shape that is revolved 180 degrees about H+1.

TRANSLATE_SOLID <DX>, <DY>, [<DZ>, <NUM COPIES>]: This command will move the selected solid by the given amount. Numbers may be positive or negative. Optionally you may also move the solid along the depth axis and/or translate the solid multiple times.

ROTATE_SOLID <XC>, <YC>, <ANGLE>, [<NUM COPIES>]: This command will rotate the selected solid about the specified position by the specified number of degrees. Numbers may be positive or negative. Optionally you may also rotate the solid multiple times.

MIRROR_SOLID <AXIS>, <AXIS VALUE>, [<# OF COPIES>: This command will flip the reflect the currently selected solid across the specified point along a specific axis. The "axis" parameter for the mirror command is either `X` or `Y`. Optionally you may also make more than one copy of the mirrored solid.

SCALE_SOLID <SCALE FACTOR>: This command will scale the selected solid.

```
SCALE_SOLID 0.1

SCALE_SOLID 25.4
```

The first example command will scale the selected solid to 10% of its current size. The second example will scale the selected solid to 25.4 times its current size. Typically this command will be used with variables whose values are set at runtime.

SOLID_UNION <SOLID REF>, <SOLID REF>: This command will perform the Union boolean function on the two specified solids.

SOLID_SUBTRACT <SOLID REF>, <SOLID REF>: This command will perform the Subtract boolean function on the two specified solids.

SOLID_INTERSECT <SOLID REF>, <SOLID REF>: This command will perform the Intersection boolean function on the two specified solids.

SELECT_SOLID <SOLID REF>: This command will select the specified solid body.

SELECT_ALL_SOLIDS: This command will select all solids, both in the workspace and the Body Bag if it is open. This command has no arguments.

DELETE_SOLID <SOLID REF>: This command will deselect the specified solid body.

DESELECT_ALL_SOLIDS: This command will deselect all solids, both in the workspace and the Body Bag if it is open. This command has no arguments.

GET_SOLID_BAGGED <SOLID REF>, <VARIABLE NAME>, <0 | 1>: This command will check if the given solid is in our out of the body bag. `1` means that it is in the bag, `0` means that it is not.

SET_SOLID_BAGGED <SOLID REF>, <0 | 1 >: This command will put a specified solid in the Body Bag or take it out. "`0`" takes the body out of the Body Bag while "`1`" puts it in the Body Bag.

Tool Commands

GET_TOOL_LIST [0 | 1]: This command creates a list of tools in the Tool list. This command does not require any arguments, by default the it looks at the entire list (which is "`0`"). Alternatively you can use "`1`" in the argument and the command will create a list only of selected tools. Once the list is generated the macro sets the variables `number_of_tools`, `first_tool_number`, `last_tool_number` and `first_free_tool_number`. This allows the macro to determine how many tools there are, what the number of the first tool is and the number of the first empty tile. Then you can use the variable `next_tool_number`. The first time you use this it is set to the number of the first tool, the next time it is set to the second tool and each time you use it the number is incremented to the next tool. So you can use a FOR/NEXT loop to look at each tool.

```
GET_TOOL_LIST 1
```

This command creates a list in memory of the selected tools in the tool list. It also sets four variables that let you look through the list of selected tools.

CREATE_MILL_TOOL <NUMBER>: This command creates a new mill-type tool. The tool is created in the Tool list at the tile number specified

CREATE_LATHE_TOOL <NUMBER>: This command creates a new lathe-type tool. The tool is created in the Tool list at the tile number specified

DELETE_TOOL <NUMBER>: This command deletes the tool that occupies the specified position in the Tool list.

NUMBER_OF_TOOLS: This command has no arguments.

GET_TOOL_STATUS <TOOL NUMBER>, <VARIABLE NAME>: This command determines the state of a tool tile. A returned value of "`0`" means the tile position is empty, "`1`" means that the tool is defined and valid while a value of "`-1`" means that not all of the data required to define the tool has been defined.

GET_TOOL_DATA <TOOL NUMBER>, <PARAMETER>, <VARIABLE NAME>: This command is used to get tool specific data. A list of the available parameters is provided in the section "Tool Data" on page 40.

SET_TOOL_DATA <TOOL NUMBER>, <PARAMETER>, <VALUE>: This command is used to set tool specific data. A list of the available parameters is provided in the section "Tool Data" on page 40.

GET_TOOL_SELECTED <TOOL NUMBER>, <VARIABLE NAME>: This command is used to check if a given tool number is currently defined. If it is the variable is set to `1`, otherwise it will be set to `0`.

SELECT_TOOL <NUMBER>: This command selects the tool that occupies the specified position in the Tool list.

SELECT_ALL_TOOLS: This command selects all tools in the tool list. This command has no arguments.

DESELECT_TOOL <TOOL NUMBER>: This command deselects the tool that occupies the specified position in the Tool list.

DESELECT_ALL_TOOLS: This command deselects all tools in the tool list. This command has no arguments.

ToolGroup Commands

GET_TG_DATA <SPINDLE #>, <TOOLGROUP #>, <PARAMETER>, <VARIABLE NAME>: This command is used to get the tool change position for a given toolgroup. The command requires input for the spindle currently being referenced, the tool group number, a parameter and a variable name to store the data. The valid parameters for this command are `tool_change_x`, `tool_change_y` and `tool_change_z`.

SET_TG_DATA <SPINDLE #>, <TOOLGROUP #>, <PARAMETER>, <VALUE>: This command is used to set the tool change position for a given toolgroup. The command requires input for

the spindle currently being referenced, the tool group number, a parameter and a value to assign to the parameter. The valid parameters for this command are `tool_change_x`, `tool_change_y` and `tool_change_z`.

Machining Process Commands

Processes are used to make operations. This set of commands lets you define and manipulate process data and create one or more operations.

GET_PROC_LIST [0 | 1]: This command creates a list of processes in the Process list. This command does not require any arguments, by default the it looks at the entire list (which is "`0`"). Alternatively you can use "`1`" in the argument and the command will create a list only of selected processes. Once the list is generated the macro sets the variables `number_of_procs`, `first_proc_number`, `last_proc_number` and `first_free_proc_number`. This allows the macro to determine how many processes there are, what the number of the first process is and the number of the first empty tile. Then you can use the variable `next_proc_number`. The first time you use this it is set to the number of the first process, the next time it is set to the second process and each time you use it it is incremented to the next process. So you can use a FOR/NEXT loop to look at each process.

```
GET_PROC_LIST 1
```

This command creates a list in memory of the selected processes in the Process list. It also sets four variables that let you look through the list of selected processes.

CREATE_PROC <NUMBER>: This command creates a process. You must follow this with commands to set the process type and process values using SET_PROC_DATA commands

DELETE_PROC <PROCESS NUMBER>: This command deletes the process number specified from the Process list.

GET_PROC_STATUS <PROCESS NUMBER>, <VARIABLE NAME>: This command checks the status of a given process number. The variable will be set to 0 if the process tile is empty, 1 if it contains a valid process definition and –1 if contains a partially complete process.

GET_PROC_DATA <PROCESS NUMBER>, <PARAMETER>, <VARIABLE NAME>: This command is used to get process data. A list of the available parameters is provided in the section "Process Data" on page 42.

SET_PROC_DATA <PROCESS NUMBER>, <PARAMETER>, <VALUE>: This command is used to set process data. A list of the available parameters is provided in the section "Process Data" on page 42.

GET_UTIL_PROC_DATA <PROCESS NUMBER>, <PARAMETER>, <VARIABLE NAME>: This command is used to get data from a utility process. A list of the available parameters is provided in the section "Utility Process Data" on page 46.

SET_UTIL_PROC_DATA <PROCESS NUMBER>, <PARAMETER>, <VALUE>: This command is used to set data for a utility process. A list of the available parameters is provided in the section "Utility Process Data" on page 46.

CLEAR_PROC_LIST: This command will delete all of the processes in the Process list. This command has no arguments.

LOAD_PROC <"FILENAME">: This command enables a macro to load a stored/saved process. The name may be simply the name of the file or it may be the entire path to the file. This command deprecates "load_process".

```
LOAD_PROC "c:\Documents and Settings\wilg\Desktop\SavedProcess3.prc"
```

CALC_PROC: This command enables a macro to calculate toolpath and create one or more operations. A process must have been set or loaded for this command to work. This is the equivalent to clicking the "Do It" button. This command deprecates "calc_process".

GET_PROC_SELECTED <PROCESS NUMBER>, <VARIABLE NAME>: This command is used to check if a given process is currently selected. The variable will be set to `1` if it is, `0` if not.

SELECT_PROC <PROCESS NUMBER>: This command will select the Process tile specified.

SELECT_ALL_PROCS: This command will select all of the processes in the Process list. This command has no arguments.

DESELECT_PROC <PROCESS NUMBER>: This command will deselect the Process tile specified.

DESELECT_ALL_PROCS: This command will deselect all of the processes in the Process list. This command has no arguments.

SET_MARKERS <0 | 1 | 2 (TOOL SIDE)>, <START FEAT>, <START DIST>, <END FEAT>, <END DIST>, [<DIRECTION>, <SINGLE FEATURE>]: This command lets you specify how the machining markers are set prior to creating toolpath.

<Tool side> values may be "`0`" (no offset), "`1`" (tool left) or "`2`" (tool right).

<Start feat> specifies which geometry feature to start on. Valid values are integers where "`1`" is the first feature in the contour, "`2`" is the second feature, etc.

<Start dist> specifies the distance along the start feature at which to start machining. Valid values are a percentage along the feature expressed as a decimal, where "`-0.1`" is 10% of the feature length before the start of the feature, "`0.5`" is halfway along the feature and "`0.9`" is 90% of the way along the feature.

<End feat> specifies which geometry feature to end on. Valid values are integers where "`1`" is the first feature in the contour, "`2`" is the second feature, etc.

<End dist> specifies the distance along the end feature at which to stop machining. Valid values are a percentage along the feature expressed as a decimal, where "`-0.1`" will start before the feature by 10% of the feature length, "`0.5`" is halfway along the feature and "`1.1`" is 10% of the feature length past the end of the feature.

<Direction> is set to 1 to machine in the direction that the shape was created, 0 to machine in the opposite direction.

<Single Feature> is set to 1 if the shape is to be cut as a single feature shape.

```
SET_MARKERS 0, 1, -0.05, 4, 1.05 1 0
```

This example sets the machining markers to cut on center, starting off of the first feature by 5% of the feature length. The 4th feature is the last to be cut and the toolpath will finish off the end of the feature by 5% of its length. The markers are set to go all the way around and cut in the direction the shape was made.

Machining Operation Commands

Operations contain toolpath data and are what gets rendered by and postprocessed GibbsCAM. This set of commands lets you manipulate operation data.

GET_OP_LIST [0 | 1]: This command creates a list of operations in the Operations list. This command does not require any arguments, by default the it looks at the entire list (which is "`0`"). Alternatively you can use "`1`" in the argument and the command will create a list only of selected processes. Once the list is generated the macro sets the variables `number_of_ops`, `first_op_number`, `last_op_number` and `first_free_op_number`. This allows the macro to determine how many operations there are, what the number of the first operation is and the number of the first empty tile. Then you can use the variable `next_op_number`. The first time you use this it is set to the number of the first operation, the next time it is set to the second operation and each time you use it it is incremented to the next operation. So you can use a FOR/NEXT loop to look at each operation.

```
GET_OP_LIST 1
```

This command creates a list in memory of the selected operations in the Operations list. It also sets four variables that let you look through the list of selected operations.

GET_OP_STATUS <NUMBER>, <VARIABLE NAME>: This command gets the status of the given operation tile number. `0` means that it is empty, `1` means that it contains a valid operation and `-1` means it is a partially completed operation tile.

DELETE_OP <NUMBER>: This command deletes the operation number specified from the Operation list.

GET_OP_DATA <NUMBER>, <PARAMETER>, <VARIABLE NAME>: This command is used to get data from an operation process. A list of the available parameters is provided in the section "Operation Data" on page 47.

SET_OP_DATA <NUMBER>, <PARAMETER>, <VALUE>: This command is used to set data for an operation process. A list of the available parameters is provided in the section "Operation Data" on page 47.

GET_OP_SELECTED <NUMBER>, <VARIABLE NAME>: This command is used to check if a given operation tile is currently selected. The variable will be set to 1 if it is, 0 if not.

SELECT_OP <NUMBER>: This command will select the Operation tile that is specified.

SELECT_ALL_OPS: This command will select all of the operations in the Operation List. This command has no arguments.

DESELECT_OP <NUMBER>: This command will deselect the Operation tile that is specified.

DESELECT_ALL_OPS: This command will deselect all of the operations in the Operation List. This command has no arguments.

DESELECT_OPS: This command has been deprecated by "deselect_all_ops". Use that command instead.

View Commands

SET_VIEW <TOP | FRONT | LEFT | RIGHT | BACK | ISO | HOME>: This command will set the view of the part to one of the seven standard views, either top, front, left, right, back, isometric or the home view for the current CS.

ZOOM_VIEW <(ZOOM FACTOR)>: This command will zoom in or out on the parts based on the zoom factor entered. A value of "`0`" is unzoom, meaning the stock boundary will fit the screen. The zoom factor must be a positive floating point value. A factor of "`0.5`" will set

the view to half the current size, "1" is the current view size and "2" is double the current size.

SHRINK_WRAP: This command will enact the Shrink Wrap function, causing the stock to shrink or expand to the bounds of all geometry and solids. This command has no arguments.

REDRAW: This command will force the screen to redraw. This is typically used at the end of macro to ensure the user sees what has been done if the system does not force a redraw.

COMMANDS TO WORK WITH EXTERNAL DATA

The GibbsCAM Macros function can read and write to text files and can also extract data from Microsoft Excel® spreadsheets.

RUN_EXE "APPLICATION FILENAME", ["TEXT" | <VAR NAME>]: This command will launch the specified application. The path to the application is required. The optional parameter is simply passed to the EXE that is being run as a command line argument. The optional parameter may be a string of text in quotes or a text variable.

So in the example you found, the result is the same as running notepad with a command line argument that is the contents of the variable $output_file. Instead of a variable, you could just as easily have the second argument as a string of text in double quotes, (for example run_exe "\Windows\System32\Notepad.exe", "c:\myfile.txt")

```
run_exe "c:\Program Files\Microsoft Office\Office10\EXCEL.EXE"

run_exe "\Windows\System32\Notepad.exe", "c:\myfile.txt"

run_exe "\Windows\System32\Notepad.exe", output_file$
```

The first item above simply launches Excel. The second example launches Notepad and opens a specific file. The third example launches Notepad and opens the file whose name is stored in a variable. Please refer to the macro "Run a Post Processor" for a working example of this.

Text Files

The macro language provides the ability to create, open, read from and write to text files. Text files should be space delimited.

FILE_OPEN <FILE NUMBER>, "FILENAME": This command will open a file for reading and writing. A maximum of 10 files may be opened and the file number must be between 1 and 10.

FILE_CLOSE <FILE NUMBER>: This command will close a file previously opened using the FILE_OPEN command.

FILE_DELETE "FILENAME": This command will delete the given filename.

FILE_READ_VARS <FILE NUMBER>, <VAR NAME 1>, ... , <VAR NAME 10>: This command will read one line from the given file number and separate out the data from that line into a set of values. Each value will be returned in the variable names used in this command.

FILE_WRITE_VARS <FILE NUMBER>, <VAR NAME 1>, ... , <VAR NAME 10>: This command will write a line of text to a file, that line containing the values of the variables in this command.

FILE_READ_TEXT <FILE NUMBER>, <VARIABLE>: This command will read a line of text from a file and save the contents in the string variable given in this command.

FILE_WRITE_TEXT <FILE NUMBER>, <TEXT>: This command will write a line of text to a file.

Excel Files

The macro language provides the ability to open an Excel spreadsheet, scan cells on any given sheet in the XLS file and extract data from specific cells.

EXCEL_OPEN "FILENAME": This command will open the specified Excel spreadsheet. The path to the file is required.

```
excel_open "c:\My Archive\Macros\point matrix.xls"
```

EXCEL_CLOSE: This command will close Excel. This command has no arguments.

EXCEL_SELECT_SHEET <SHEET NAME>: This command will select all of the data in the specified sheet of an open Excel file. As a note, Excel files have three sheets by default, labelled "Sheet1", "Sheet2" and "Sheet3", these may be renamed and the macro requires the correct sheet name.

EXCEL_FIND_CELL <ROW1>, <COL1>, <ROW2>, <COL2>, <CELL TEXT>, <ROW VARIABLE>, <COL VARIABLE>: This command finds a cell containing a particular value within a specified range. This command matches the exact value only.

row1, col1 - upper-left coordinates of the search range

row2, col2 - lower-right coordinates of the search range

cell text - the text to search for

row variable - output; row of found cell

col variable - output; column of found cell

EXCEL_GET_CELL <ROW>, <COL>, <VARIABLE NAME>: This command will extract data from a particular cell. You must specify the row and column to identify the cell. The data in the cell will be assigned to a given variable.

```
EXCEL_GET_CELL 3, e, data3
```

EXCEL_GET_RANGE <RANGE NUMBER>, <ROW 1>, <COL 1>, <ROW 2>, <COL 2>: This command will extract data from a range of cells. You must specify the start cell (row and column) followed by the cell at the end of the range.

```
EXCEL_GET_RANGE 1, 1, a, 1, j
EXCEL_GET_RANGE 2, 2, a, 11, j
```

The first example above gets the text from the first 10 cells (a-j) in the first column. The second range example gets the text from a 10x10 cell range, rows 2-11 and columns a-j.

MISCELLANEOUS COMMANDS

SLEEP <TIME IN SECONDS>: This command will cause the macro to wait for a given amount of time. This puts the macro to "sleep" for a few seconds, which gives the operating system time to close files and clear buffers when you are creating and/or reading data from external applications.

DEBUGGING MACROS

These commands are provided to assist with debugging.

MESSAGE "TEXT", ["CAPTION"]: This command will display a message box with given text. Optionally you may also include a caption on the message by including text in a second set of quotation marks. This command is often used with a logic statement.

```
MESSAGE "Create contour", "Let's Make Some Toolpath"
```

DEBUG <VARIABLE>, <VARIABLE> ... This command will display a message box with the value of the given variable(s).

STOP "TEXT", [<CAPTION>]: This command will stop the macro and display a dialog with the text of your choosing. This command is often used with a logic statement for error checking. The text must be contained within quotation marks. Optionally you may give the message box a custom title.

```
STOP "This is not used properly. RTFM."
```

CHECK <PARAMETER>, <ERROR MSG TEXT>: This command can check to see if there is a part currently open in GibbsCAM and what type of part it is. The command has three parameters, "part_open", "part_mill", "part_turn"and "part_mtm". In each case the variable gets set to "1" if the case is true, so if a part is open the value is "1". If the part type and parameter do not match then the macro will stop and the error message will be displayed.

PART_OPEN	Stop if there is no part currently open
PART_MILL	Stop if the current part is not MILL
PART_LATHE	Stop if the current part is not LATHE
PART_MTM	Stop if the current part is not MTM

TRACE <ON | OFF>: This command will display each macro in a message box. You will need to press OK to continue processing the macro.

PARAMETERS FROM GIBBSCAM

PART DATA

The following parameters are accessible by the get_part_data and set_part_data commands.

All Part Types		
type	units	radius
auto_clear	num_flows	num_tool_groups
num_spindles	cp1	clear_rad
auto_clear	graphic_part_dist_on	graphic_part_dist_val
part_name (string variable)	part_file (string variable)	mdd_name (string variable)
mdd_file (string variable)	comment (string variable)	alloy (string variable)
family (string variable)	hardness (string variable)	
Mill Parts		
tool_change_x	tool_change_y	stock_x1
stock_y1	stock_z1	stock_x2
stock_y2	stock_z2	
Turn Parts		
tool_change_x	tool_change_z	stock_rad
stock_z1	stock_z2	

MTM SETUP DATA

The following parameters are accessible by the get_mtm_data and set_mtm_data commands.

General MTM Data		
spindle_used	stock_z1	stock_z2
stock_rad	chuck_width	chuck_part_len

TOOL DATA

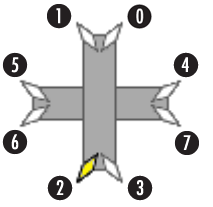
The following parameters are accessible by the get_tool_data and set_tool_data commands.

All Tools		
type	material	offset
spindle_dir	id	use_id
tool_group_pos	tool_group	prime_spindle
comment (this is a string variable)		

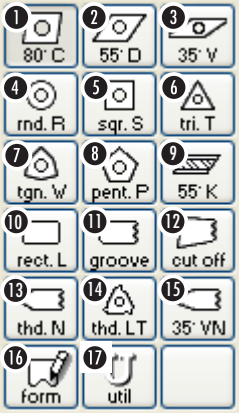
Mill Tools		
mtool_type	mtool_num_flutes	mtool_len_offset
mtool_options	mtool_length	mtool_flute_length
mtool_radius	mtool_shank_rad	mtool_top_corner_rad
mtool_corner_rad	mtool_draft_angle	mtool_non_cut_dia
mtool_orientation	mtool_tip_angle	mtool_pitch_tpi
mtool_preset	mtool_tc_shift	mtool_lead_tip
mtool_tip_rad	mtool_inner_top_corner_rad	mtool_bottom_rad
mtool_top_rad	holder_type	holder_solid_ref

Lathe Tools		
ltool_type	ltool_isize	ltool_orientation
ltool_holder	ltool_thread_style	ltool_thread_type
ltool_thread_id_od	ltool_thread_dir	ltool_offset
ltool_cut_side	ltool_face_up	ltool_neg_side
ltool_other	ltool_holder_option	ltool_size
ltool_thick	ltool_length	ltool_tip_rad
ltool_insert_angle	ltool_tip_offset	ltool_preset_pt_x
ltool_preset_pt_z	ltool_turret_shift_x	ltool_turret_shift_z
ltool_face_angle	ltool_side_angle	ltool_tip_width
ltool_tip_length	ltool_thread_pitch	ltool_thread_flat_len
ltool_thread_insert_width	ltool_thread_edge_pos	ltool_mid_angle
ltool_tip_centre_to_preset	ltool_face_relief	ltool_dia_relief
ltool_holder_thickness	ltool_b_axis	ltool_util_len
ltool_util_dia	ltool_util_angle	

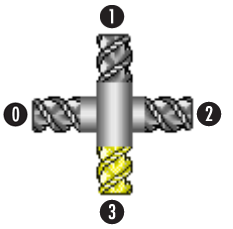
LTOOL_ORIENTATION: This item has eight values, 0 through 7. The values correspond to a position in the tool setup dialog as shown here.



LTOOL_TYPE: This item has 17 values, numbered 1 through 17. The values correspond to a position in the tool setup dialog as shown here.



MTOOL_ORIENTATION: This item has four values, 0, 1, 2 and 3. The values correspond to a position in the tool setup dialog as shown here.



MTOOL_TYPE: This item has 20 values, numbered 1 through 20. The values correspond to a position in the tool setup dialog as shown here.

PROCESS DATA

All Process Types

The following parameters are accessible by the get_proc_data and set_proc_data commands when referencing any type of part.

Standard Process Parameters		
type	mill_type	lathe_type
tool_num	prog_stop	rapid_in
material_only	spring_passes	cut_type
repeats	tool_change	cs_num
path_cs	crc	coolant

Mill Processes

The following parameters are accessible by the get_proc_data and set_proc_data commands when referencing a Mill-type part.

Milling Process Parameters		
entry_clear	exit_clear	entry_line
exit_line	entry_rad	exit_rad
entry_feed	contour_feed	surf_z
tip_z	z_step_wanted	cut_width
stock	boss_stock	overhang
overlap	open_clearance	rpm
drill_feed	prefer_subs	cp1
curve_tol	auto_entry_cp	auto_entry_clear_rad

Milling Process Parameters		
auto_exit_cp	auto_exit_clear_rad	axis_info
geo_rad	min_rad	z_step
fm_clear	fm_cut_width	cut_angle
stepover_feed	clear_feed	scallop_feed
min_cut	msurf_clear	mstock_tol
mfix_tol	mzshift	

Mill Drilling Process Parameters		
drill_type	drill_pos_approach	drill_clear_plane
drill_dwell	drill_peck_amount	drill_peck_clearance
drill_tap_percent	drill_bore_pulloff	drill_peck_retract

Pocket & Contour Process Parameters		
feed_connect	crc_offset	len_offset
custom_cbset	pattern_on	pattern_index
round_corners	entry_perp	exit_perp
ignore_tools	depth_first	fm_use_shape
fm_hplus	fm_vplus	fm_moveh
fm_before_zigzag	fm_back_and_forth	fm_type
entry_type	start_on_right	retracts
hit_parallel	stay_clear	cut_back
same_stroke	full_dia	stepover_feed_lock
clear_feed_lock	scallop_feed_lock	clear_periph
past_stock	pocket_type	wall_mode
hit_flats		

Pocket Entry Parameters	
pocket_entry_ramp_type	pocket_entry_helix_type
pocket_entry_plunge_pt	pocket_entry_ramp_slope
pocket_entry_ramp_start_z	pocket_entry_ramp_max_cut
pocket_entry_ramp_wall_clear	pocket_entry_ramp_angle
pocket_entry_helix_slope	pocket_entry_helix_angle
pocket_entry_helix_start_z	pocket_entry_helix_max_cut
pocket_entry_helix_wall_clear	pocket_entry_helix_dia
pocket_entry_helix_cp	pocket_entry_periph_start_z
pocket_entry_periph_slope	wall_top
wall_angle	wall_bottom
walli_top	walli_angle
walli_bottom	wall_user_step
wall_shape_step	wall_ridge_height

Surface Process Parameters		
surface_type	surface_cut_type	surface_back_and_forth
surface_lace_cut_dir	surface_output_pref	surface_efeed
surface_cfeed	surface_stock	surface_fix_clear
surface_stepover	surface_ridge_height	surface_cut_angle
surface_desired_z	surface_const_faces_clear	surface_cut_tol
surface_stock_tol	surface_fix_tol	surface_constraint_tol
surface_smooth_tol	surface_arc_fit_tol	surface_spline_approx_tol
Lace Cut Parameters		
surface_cut_opt	surface_offsets_surfs	surface_one_pass
surface_constrain	surface_retract_option	surface_stay_in_stock
surface_clear_stock	surface_clear_stock_type	surface_cut_over_edges
surface_skip_flats	surface_normal_angle	surface_step_cut_ratio
2 Curve Flow Parameters		
curve_cut_dir	surface flow	
surface_flow_mach_dir	surface_flow_mach_z_order	
surface_flow_travel_around_mode	surface_flow_cut_type	
Intersection Parameters		
surface_intersect_type	surface_cuts_type	surface_passes_type
surface_cleanup_type	surface_passes_per_side	surface_corner_rad
surface_max_cut_angle		
Surface Process Entry Parameters		
surface_entry_type	surface_entry_ramp_type	wall_step
wall_top_to_bottom	wall_same_dir	wall_same_dcep_side
wall_swept_pocket	wall_swept_island	s_use_adv_tol
m_use_adv_tol	cut_side	md_from_tool
md_vary_with_geo	md_vary_with_feat	md_retract_level
md_process_load_hız_feat	surface_entry_plunge_pt	rapid_clearance
surface_stock		
Contour Process Entry Parameters		
contour_entry_ramp_type	contour_entry_helix_type	
contour_entry_ramp_slope	contour_entry_ramp_start_z	
contour_entry_ramp_max_cut	contour_entry_ramp_angle	
contour_entry_helix_slope	contour_entry_helix_start_z	
contour_entry_helix_max_cut	contour_entry_helix_dia	
Thread Milling Process Parameters		
mthread_traverse_cpı	mthread_id_od	mthread_thread_dir
mthread_rapid_in	mthread_start_thread	mthread_end_thread
mthread_cut_dia	mthread_clear_dia	mthread_pitch

Thread Milling Process Parameters
mthread_feed

Lathe Processes

The following parameters are accessible by the get_proc_data and set_proc_data commands when referencing a Mill-type part.

Lathe Process Parameters		
lathe_prefer_canned	lathe_clear_id	
Lathe Drilling Process Parameters		
ldrill_entry	ldrill_dwell	ldrill_peck
ldrill_clearance	ldrill_retract	ldrill_tap_percent
ldrill_feed	ldrill_tip_z	ldrill_surface_z
Lathe Roughing Process Parameters		
lrough_round_corners	lrough_cut_off	lrough_cut_dir
lrough_entry_perp	lrough_exit_perp	lrough_rough_type
lrough_css	lrough_rapid_step	lrough_monotonic
lrough_avoid_air	lrough_xplus	lrough_xminus
lrough_zplus	lrough_zminus	lrough_autofin
lrough_spindle_dir	lrough_finish_last	lrough_center_out
lrough_constant_step	lrough_peck	lrough_plunge_type
lrough_constant_path	lrough_fixed_sp	lrough_num_passes
lrough_min_rad	lrough_stock	lrough_stock_x
lrough_stock_z	lrough_entry_line	lrough_entry_rad
lrough_exit_rad	lrough_exit_line	lrough_rough_start
lrough_max_rpm	lrough_sfpm	lrough_feed
lrough_clearance	lrough_turn_cut_width	lrough_ramp_angle
lrough_plunge_angle	lrough_cut_width	lrough_feed_percent
lrough_peck	lrough_retract	lrough_cut_x
lrough_cut_z	lrough_shift_cut_width	lrough_cycle_sp
Lathe Threading Process Parameters		
lthread_style	lthread_cut_dir	lthread_cut_type
lthread_css	lthread_balanced_in_feed	lthread_alternate
lthread_last_cut	lthread_nominal_pitch_index	lthread_taper_index
lthread_num_spring_passes	lthread_num_starts	lthread_max_rpm
lthread_sfpm	lthread_z_start	lthread_z_end
lthread_nominal	lthread_pitch	lthread_major_dia
lthread_minor_dia	lthread_thread_height	lthread_taper
lthread_thread_angle	lthread_cut_depth	lthread_last_cut

Lathe Threading Process Parameters		
lthread_z_start_ext	lthread_z_end_ext	lthread_x_start_ext
lthread_x_end_ext		

UTILITY PROCESS DATA

The following parameters are accessible by the get_util_proc_data and set_util_proc_data commands.

Utility Process Names		
load_spindle	unload_spindle	part_shift
sub_spindle_in	sub_spindle_return	parts_catcher_in
parts_catcher_out	all_stop	move_toolgroup
Load Spindle - Manual Chuck		
spindle_num	time	
Load Spindle - Auto Chuck		
spindle_num	time	
Load Spindle - Bar Feed		
spindle_on	forward	spindle_speed
z_clearance	x_position	feedrate
initial_face_z		
Load Spindle - Auto Bar Feed		
flow_num	spindle_on	forward
spindle_speed	feedrate	feed_distance
Load Spindle - Bar Pull		
z_clearance	x_position	feedrate
initial_face_z	grip_z	
Unload Spindle		
spindle_num	time	
Part Shift - Bar Feed		
spindle_on	forward	spindle_speed
z_clearance	x_position	feedrate
initial_face_z	shift_distance	
Part Shift - Bar Pull		
z_clearance	x_position	feedrate
initial_face_z	grip_z	shift_distance
Part Shift - Bar Pull		
spindle_on	forward	spindle_speed
feedrate	shift_distance	

SubSpindle In		
flow_1	flow_2	fromworkpiece
to_workpiece	c_synched	spindle_speed
forward	sub_in_unload	part_in_main
part_in_sub	z_clearance	z_grip
feedrate	orientation	

Please note that the options that are used for check boxes will only have a value of 0 or 1.

SubSpindle Return		
spindle_on	forward	with_part
open_main_collet	main_loaded	spindle_speed

Parts Catcher In		
spindle_num	z_position	time

Parts Catcher Out		
spindle_num	time	spindle_unloaded

Move ToolGroup		
spindle_num	location	x_home
x_value	z_home	z_value
cs	control_point	

All Stop
This Utility op has no variables

OPERATION DATA

All Operation Types

The following parameters are accessible by the get_op_data and set_op_data commands.

Type	
mill_type	lathe_type

All Operation Types		
tool_num	proc_group	proc_id
proc_op	wg_num	cs_num
path_cs	locks	num_repeats
crc_dir	crc_offset	coolant
cut_type	css	crc_side
mach_engine		

Comment String	
op_start (this is a string variable)	op_end (this is a string variable)

Mill-Type Operations

The following parameters are accessible by the get_op_data and set_op_data commands when referencing a mill-type operation.

All Milling Operations		
entry_clear	exit_clear	entry_feed
contour_feed	len_offset	wrap
wrap_dups	prog_stop	pocket_type
tool_group	workpiece	flow
spin_control	wrap_start_angle	wrap_dup_angle
cut_width	stock	boss_stock
rpms	time	length
css	surf_stock	

Mill Drilling Operations		
drill_type	drill_clear_plane	drill_pos_approach
material_only	rapid_in	drill_clear_plane
drill_dwell	drill_tap_percent	drill_peck_clearance
drill_peck_amount	drill_peck_retract	drill_bore_pulloff
drill_cb_start_riz	drill_cb_end_riz	drill_clear_plane_loc_cs

Mill Pocket & Contour Operations		
feed_connect	crc_offset	pattern_on
pattern_index	round_corners	entry_perp
exit_perp	ignore_tools	depth_first
fm_use_shape	fm_hplus	fm_vplus
fm_moveh	zig_zag	back_forth
fm_type	entry_type	start_on_right
retracts	hit_parallel	stay_clear
cut_back	same_stroke	full_dia
stepover_feed_lock	clear_feed_lock	scallop_feed_lock
clear_periph	hit_flats	cp1
curve_tolerance	auto_entry_cp	auto_entry_clear_rad
auto_exit_cp	auto_exit_clear_rad	geo_rad
min_rad	z_step	fm_clear
fm_cut_width	cut_angle	stepover_feed
clear_feed	scallop_feed	min_cut
past_stock	pocket_entry_plunge_point	pocket_entry_ramp_slope
pocket_entry_ramp_start_z	pocket_entry_ramp_max_cut	pocket_entry_ramp_wall_clear
pocket_entry_ramp_angle	pocket_entry_helix_slope	pocket_entry_helix_angle
pocket_entry_helix_start_z	pocket_entry_helix_max_cut	pocket_entry_helix_wall_clear
pocket_entry_helix_dia	pocket_entry_helix_cp	pocket_entry_periph_start_z

Mill Pocket & Contour Operations		
pocket_entry_periph_slope	entry_line_len	exit_line_len
entry_rad	exit_rad	z_step_wanted
overhang	overlap	open_clearance
msurf_clear	msurf_stock_tol	mfixture_tol
m_zshift	contour_entry_ramp_slop	contour_entry_ramp_start_z
contour_entry_ramp_max_cut	contour_entry_ramp_angle	contour_entry_helix_slope
contour_entry_helix_start_z	contour_entry_helix_max_cut	contour_entry_helix_dia

Mill Contour & Pocket Wall Operations		
wall_step	wall_top_to_bottom	wall_same_dir
wall_dcep_side	wall_swept_pocket	wall_swept_island
wall_top	wall_ang	wall_bot
wall_itop	wall_iang	wall_ibot
wall_user_step	wall_shape_step	wall_ridge_height
cut_side	s_use_adv_tol	m_use_adv_tol
wall_mode	inter_op_override	util_type
md_from_tool	md_vary_with_geo	md_vary_with_feat
md_retract_level		

Contour Entry Information		
contour_entry_ramp_type	contour_entry_helix_type	prefer_subs

Pocket Entry Information	
pocket_entry_ramp	pocket_entry_helix_type

Thread Milling		
mthread_traverse_cp1	mthread_id_od	mthread_thread_dir
mthread_start_thread	mthread_end_thread	mthread_cut_dia
mthread_clear_dia	mthread_pitch	mthread_feed

Surfacing Information - General		
surface_type	surface_cut_type	surface_back_forth
surface_lace_cut_dir	surface_output_pref	surrface_depth
surface_entry_plunge_pt	rapid_clearance	surface_step_over
surface_ridge_height	surface_cut_angle	surface_desired_z
surface_constraint_faces_clear	surface_stock_tol	surface_fix_tol
surface_constraint_tol	surface_smooth_tol	surface_arc_fit_tol
surface_spline_tol	surface_normal_angle	surface_step_cut_ratio
surface_corner_rad	surface_max_cut_angle	

Surfacing Information - Lace Cut		
surface_cut_opt	surface_offset_surfs	surface_one_pass
surface_constrain	surface_retract_opt	surface_stay_in_stock
surface_clear_stock	surface_clear_stock_type	surface_cut_over_edges

Surfacing Information - Lace Cut		
surface_skip_flats		
Surfacing Information - 2 Curve Flow		
surface_curve_cut_dir		
Surfacing Information - Surface Flow		
sflow_mach_dir	sflow_mach_zorder	
sflow_travel_around_mode	sflocut_type	
Surfacing Information - Intersection		
intersect_type	intersect_cuts_type	intersect_pass_type
intersect_cleanup_from	intersect_passes_per_side	
Surfacing Entry Information		
surface_entry_type	surface_entry_ramp_type	

Lathe-Type Operations

The following parameters are accessible by the get_op_data and set_op_data commands when referencing a lathe-type operation.

All Turning Operations		
material_only_clearance	lplunge_max_cut	lplunge_angle
lplunge_peck_amount	lplunge_peck_clearance	lplunge_peck_retract
lplunge_feed_percent	lplunge_entry_type	lathe_pattern_shift_cut
lathe_pattern_shift_point	lathe_max_rpm	lathe_xz_stock
lathe_cut_width	cut_tol	step
depth	od_id_face	lathe_depth
Lathe Threading		
lthread_balanced	lthread_angle_alt	lthread_starts
lthread_last_cut	lathe_prefer_canned	lathe_canned_autofin
lthread_type	lathe_cut_dir	num_spring_passes
Lathe Roughing		
lathe_rough_type	lathe_pull_off_wall	lathe_plunge_cut_type
lathe_plunge_center_out	lathe_plunge_type	lathe_plunge_entry_type
lathe_pat_shift_fixed_srt	lathe_pat_shift_passes	lathe_square_corners
lathe_no_drag	lathe_cut_off	lathe_od_id_face
lathe_cut_other_side	lrough_round_corners	lrough_exit_perp
lrough_xplus	lrough_xminus	lrough_zplus
lrough_zminus	lrough_autofin	lrough_spindle_dir
lrough_constant_path	lrough_min_rad	lrough_entry_line
lrough_entry_rad	lrough_exit_rad	lrough_exit_line
lrough_start	lrough_cut_width	lrough_ramp_angle

Lathe Threading		
lthread_stype	lthread_alternate	lthread_nom_pitch_index
lthread_angle	lthread_nominal_xd	lthread_tpi
lthread_pitch	lthread_slope	lthread_minor_xd
lthread_height_xr	lthread_run_in	lthread_run_out
lthread_first_cut	lthread_last_cut	lthread_z_start
lthread_z_end	lthread_major_dia	lathe_surface_z

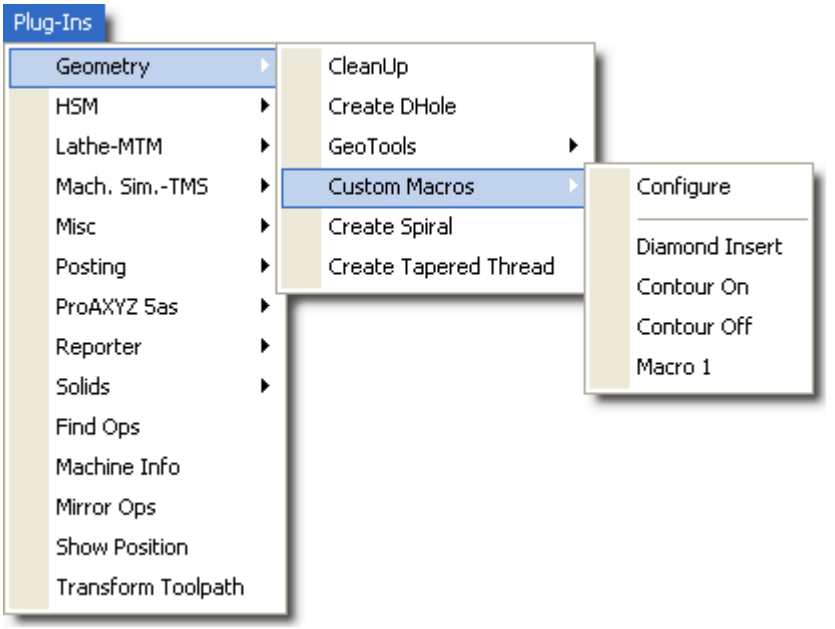
POST DATA

The following parameters are accessible by the get_post_data and set_post_data commands.

All Posting Data		
start_prog_num	seq_from	seq_by
abs_moves	comments	op_stops
num_parts	work_fixtures	one_part_all_tools
inter_part_full_up	minimize	selected_ops
pref_header	pref_sub	pref_op
pref_footer	num_flows	num_tool_groups
num_spindles	part_spacing_x	part_spacing_y
part_spacing_z	post_file (string variable)	output_file (string variable)

USING GIBBSCAM MACROS

The macros function can be found in the Geometry section of the Plug-ins menu. The Configure option lets you tell the macro plug-in where your macros are stored. Any macro that the plug-in is aware of will appear in the menu. Simply select a macro from the menu to run the macro.

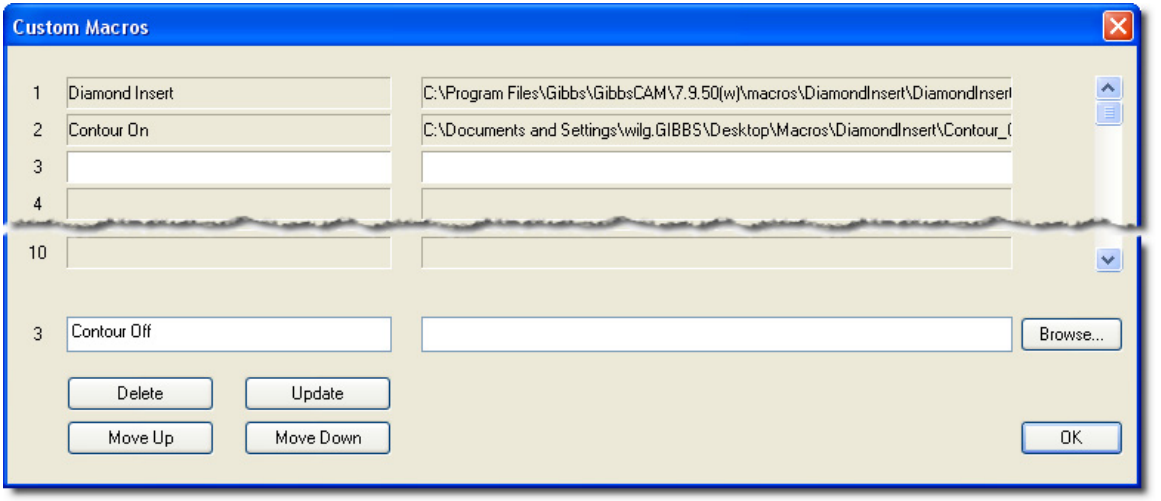


CONFIGURING THE CUSTOM MACROS MENU

The Custom Macros menu shows all of the macros that the system is aware of. You can add to and customize these entries by selecting the Configure menu item.

Once the Custom Macros dialog is open you can add and organize the macros. To add a macro simply select an empty field, enter a name you would like the macro to appear as and enter the path to the macro file. You can use the Browse function to point to the file you would like to add or you may type the path in manually. Once you have entered the path click the Update button to save this information. Finally, click the OK button. This

will close the Custom Macros dialog and the next time you launch GibbsCAM the macros you added will be available.

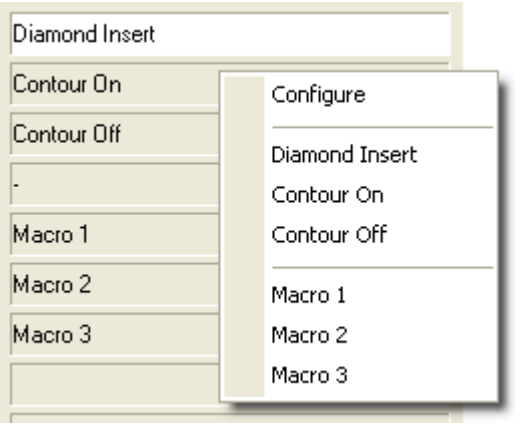


TIP

You can avoid having to go through the process of entering the macros each time you get a new version of GibbsCAM. The information is saved in the file "Macros.ini" which is located in the GibbsCAM application data folder. You can simply copy this file to the new version's folder.

From here: C:\Documents and Settings\All Users\Application Data\Gibbs\GibbsCAM\8.0\plugins\data
To here: C:\Documents and Settings\All Users\Application Data\Gibbs\GibbsCAM\8.1\plugins\data

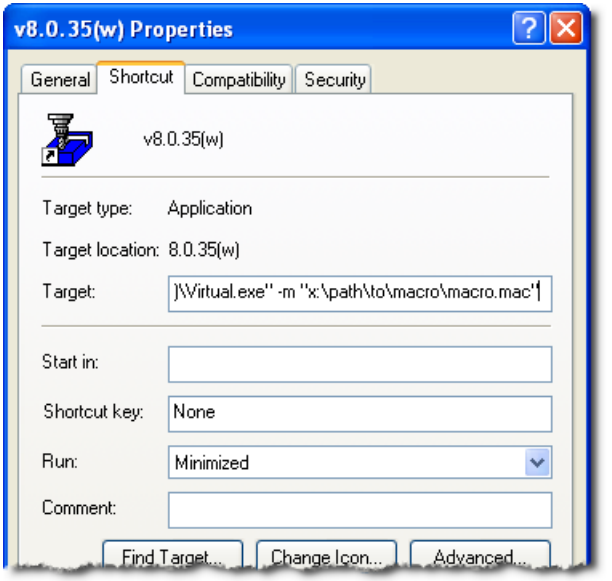
Items in the list can be reorganized. Simply select an entry and click the Move Up or Move Down button as needed. Additionally, you can create a divider by simply putting a dash (-) in an entry. This will help you organize groups of macros.

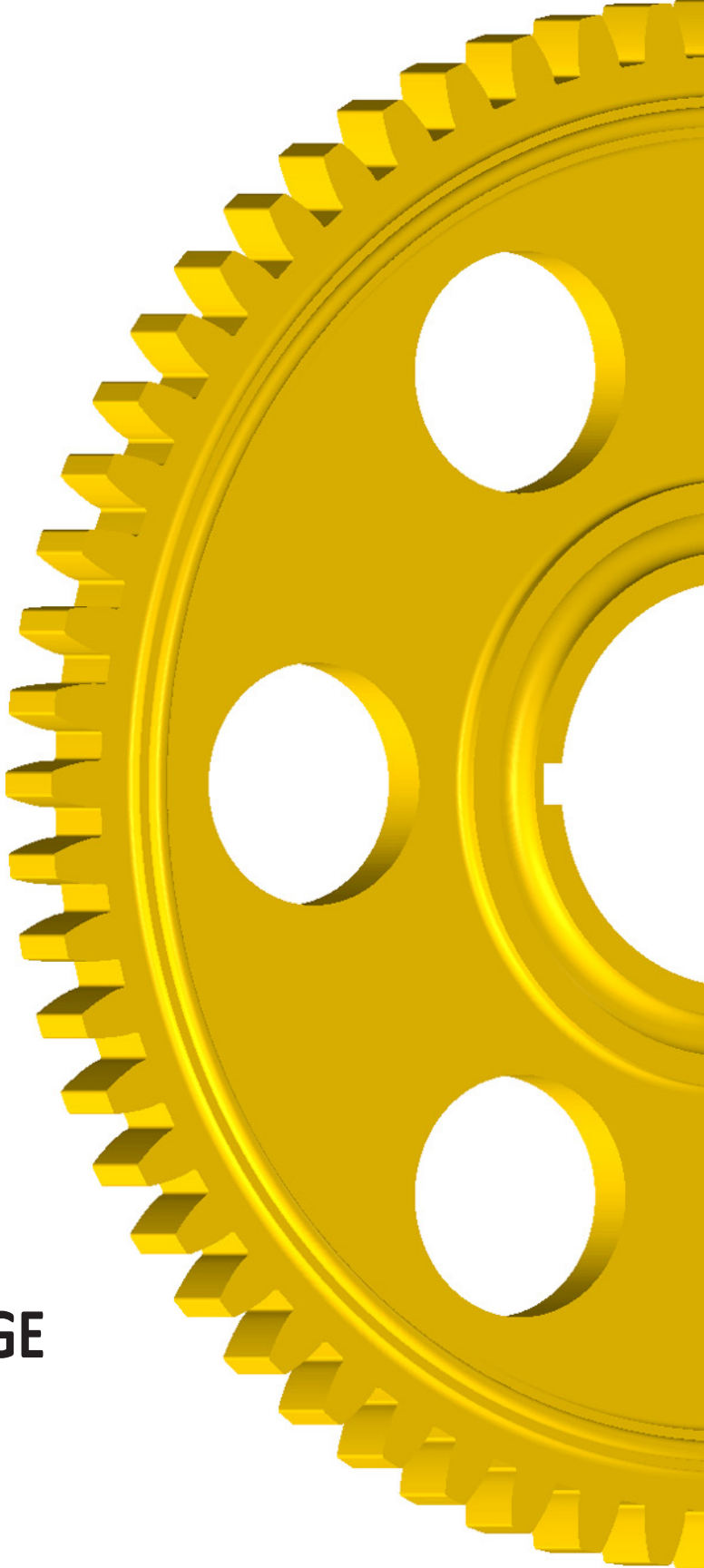


STARTING MACROS

Macros are accessed from the Plug-Ins menu of GibbsCAM. See the Plug-Ins Guide for information on adding macros to the menu. Additionally a macro may be automatically run when GibbsCAM is launched using a command line parameter. To use this the macro filename must be specified as an absolute path. The command may be run from the command line interface, added to the shortcut to GibbsCAM (see image to the right) or added to a batch file.

```
-m "x:\path\to\macro.mac"
```





MACRO LANGUAGE SAMPLES

CHAPTER 2: *Macro Language Samples*

GOOD PROGRAMMING PRACTICES

When you write a macro, remember that you may need to go back to it at a later date to fix bugs or to modify the code. You may also want to look at old macros to reuse parts of the code in other projects. It is therefore a good idea to take a little extra time to make your code easier to read and understand by using the following simple rules:

- Use meaningful variable names
- Break up code with blank lines between blocks of commands
- Add comments to describe what you are doing
- Indent code for loops to identify repeated code
- Add error checks to make sure the data is valid

GIBBSCAM MACRO SAMPLES

QUICK SAMPLES

User Input Example

An example of a dialog definition follows.

```
DIALOG 100,250,200,120
IMAGE 200, 30, 50, 50, "c:\macros\pic1.bmp"
LABEL 30, 30, 80, 20, "Width"
LABEL 30, 60, 80, 20, "Height"
INPUT 120, 30, 60, 20, w1, 10
INPUT 120, 60, 60, 20, h1, 6
CANCEL 30, 90, 60, 20
OK 120, 90,60, 20
```

Geometry Creation Example

An example of a contour definition follows.

```
CONTOUR [
   START 1, 2
   LINE 1, 4
   ARC 2, 4, 2, 5, CW
   LINE 4, 5
```

```
    ARC 4, 4, 5, 4, CW
    LINE 5, 2
    LINE 2, 2
]
```

Simple Geometry Selection and Rotation Example

An example of creating a shape and rotating it follows.

```
CONTOUR [
    START x, y
    LINE (x+10), y
    LINE (x+10), (y+6)
    LINE x, (y+6)
    LINE x, y
]
refnum = ContourRef
CLEAR_SELECT
SELECT_SHAPE refnum
ROTATE_GEO x, y, 30
```

Machining Example

As an example, we could follow the contour created in "Simple Geometry Selection and Rotation Example" with the following:

```
LOAD_PROCESS "c:\macros\process1.prc"
CALC_PROCESS
SET_MARKERS 1, 1, 0.5, 1, 0.5
```

This will start and end half way along the first feature and cut with tool to the left.

Advanced Geometry Create & Transform Example

This example will create a simple shape then apply successive transformations to the geometry.

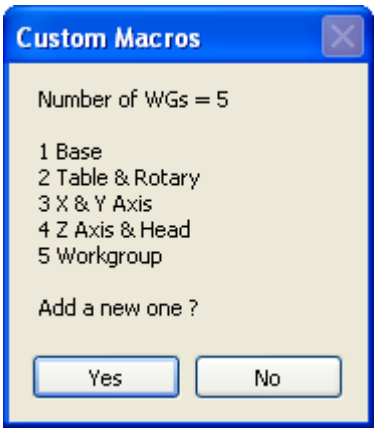
```
a1$ = "This macro will use the geometry transformations"
a2$ = "translate, rotate, mirror, scale and copy"
message "%a1$\n%a2$"

      ! ------------------------------------
      ! start a new part and set the stock size
new_part "Example.vnc"
set_part_data mdd_name, "VMill3a"
      ! 3 axis vertical mill
set_part_data units, 1
      ! inches
```

```
set_part_data stock_x1, -10
set_part_data stock_y1, -10
set_part_data stock_z1, -1
set_part_data stock_x2,  10
set_part_data stock_y2,  10
set_part_data stock_z2,  1

set_view top
zoom_view 0
      !  zoom full
message "Create a contour", "Geo Transform", 1
contour [
   start 0, 0
   line  5, 0
   line  4, 2
   line  4, 1
   line  0, 1
   line  0, 0
]
iref = ContourRef

      !  ------------------------------------
      !  Start the transformations
message "Select the contour", "Geo Transform", 1

clear_select
select_shape iref
redraw
get_selection_list 1
      !  save list of currently selected geometry

      !  ------------------------------------
      !  Translate
message "Translate by X 3, Y 4", "Geo Transform", 1
translate_geo 3, 4, 0
redraw

      !  ------------------------------------
      !  Rotate
message "Rotate cw by 30 degrees", "Geo Transform", 1
rotate_geo 0, 0, 30
redraw

      !  ------------------------------------
      !  Mirror
message "Mirror in X, about Y 0", "Geo Transform", 1
```

```
mirror_geo x, 0
redraw

      !  ------------------------------------
      !  Translate
message "Copy Translate by X 6, Y 0", "Geo Transform", 1
translate_geo 6, 0, 0, 1
redraw

get_selection_list 2
      !  save list of currently selected geometry
message "Select both shapes", "Geo Transform", 1

set_selection_list 1
      !  select the original shape
set_selection_list 2, 1
      !  add the copy
redraw

      !  ------------------------------------
      !  Mirror
message "Copy Mirror in Y about Y 1.5", "Geo Transform", 1
mirror_geo y, 1.5, 1
redraw

      !  ------------------------------------
      !  Scale
message "Scale all geometry to half size", "Geo Transform", 1
select_all_geo
scale_geo 0.5
redraw

message "Finished", "Geo Transform"
```

Get WG List Info

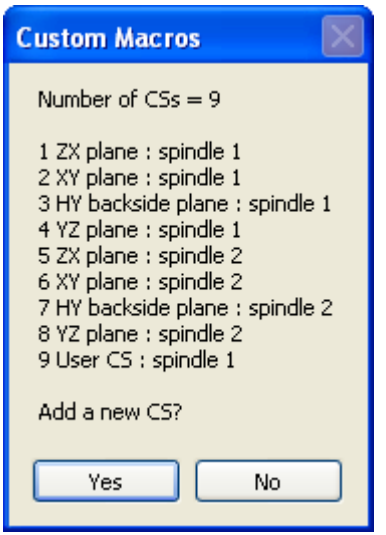
This example will create a list of all of the workgroups in the current part. The macro then prompts the user to optionally add a new WG.




```
a1$ = "This macro will get a list of all
   WGs,"
a2$ = "create a new WG and select it."
message "%a1$\n%a2$"

check part_open, "You must have a part open
   to run this macro"

get_wg_list

inum = number_of_wgs
if inum<1 then stop "No WG found"
if inum>20 then inum = 20
      !  just display data for the first 20 workgroups
a$ = "Number of WGs = %inum\n"

for i=1 to inum
   n = next_wg_number
   get_wg_name n, wg$
   a$ = a$ + "\n" + format$(n, "###0") + " " + wg$
next i

a$ = a$ + "\n\nAdd a new one ?"
yesno a$, iyesno
if iyesno=0 then stop "Finished"

new_wg "Macro WG"
iwg = WgNumber
set_wg iwg
update_wg_window

a$ = "Created new WG, number " + format$(iwg, "###0") + " = 'Macro
   WG'"

stop "Finished\n"+a$
```

Get CS List Info

This example will create a list of all of the coordinate systems in the current part including which spindle the CS is aligned to. The macro then prompts the user to optionally add a new CS (which is hard coded in this example), that will be the current CS at the end of the macro.




```
a1$ = "This macro will get a list the CSs and
   indicate"
a2$ = "which spindle number is associated
   with each CS."
a3$ = "Lastly you can create a new CS at the
   end of the macro."
message "%a1$\n%a2$\n%a3$"

check part_open, "You must have a part open
   to run this macro."

get_cs_list

inum = number_of_css
if inum<1 then stop "No CS found"
if inum>20 then inum = 20
      !  just display data for the first 20
a$ = "Number of CSs = %inum\n"

for i=1 to inum
   n = next_cs_number
   get_cs_name n, cs$
   get_cs_spindle n, ispin

   a$ = a$ + "\n" + format$(n, "###0") + " " + cs$
   a$ = a$ + " : spindle "+format$(ispin,"0")
next i

a$ = a$ + "\n\nAdd a new CS?"

      !  This section is for adding a new CS.
      !  Commenting out everything between here
      !  and the "stop" command will cause the
      !  macro to only generate a list of the CSs.
yesno a$, iyesno
if iyesno=0 then stop "Finished"

new_cs 3P, "Macro CS", 0, 0, 0, 1, 0, 0, 0, 1, 1
ics = CsNumber
```

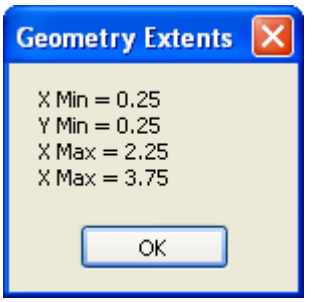
```
set_cs ics
update_cs_window

a$ = "Created new CS, number " + format$(ics, "###0") + " = 'Macro
   CS'"

stop "Finished\n"+a$
```

Calculate the Extents of Part Geometry

This macro will calculate the extents (the min/max values) in X and Y of selected geometry in a mill part and can optionally create geometry that bounds the extents.

```
a1$ = "This macro will calculate the extents"
a2$ = "(min/max X and Y) of the selected geometry"

message "%a1$\n%a2$"

check part_open, "You must have a part open to run this macro"
check part_mill, "This macro is not designed for turned parts"

global xmin, ymin, xmax,ymax ! these variables are used by all macros
global ics

xmin=99999
ymin=99999
xmax=-99999
ymax=-99999

      ! when we get the data for each feature, we can get the xyz values in either the
        world cs or the cs local to each feature.
ics = 1
      ! world cs (local cs would be zero)
      ! ----------------------------------
      ! get the number of selected features
get_num_feat_selected inum
if inum<1 then stop "No geometry selected"
      ! ----------------------------------
```

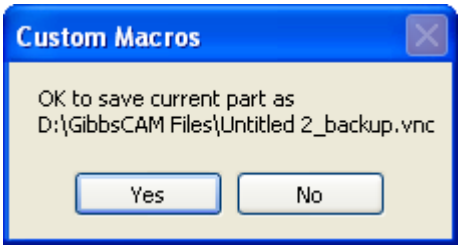
```
      !  loop through all selected features and update the min/max xy data
         according to each features xy data
for i=1 to inum
      !  ----------------------------------
      !  get the feature reference number for the 'i'th selected feature
   get_selected_geo_ref i, iref
      !  ----------------------------------
      !  get the feature type for this feature
get_feat_type iref, itype
   if itype=1 then call "Check_Point_Data.mac"
   if itype=2 then call "Check_Line_Data.mac"
   if itype=3 then call "Check_Circle_Data.mac"
   if itype=4 then call "Check_Arc_Data.mac"
next i

      !  ------------------
      !  display the results
xmin$=format$(xmin,"####0.0###")
ymin$=format$(ymin,"####0.0###")
xmax$=format$(xmax,"####0.0###")
ymax$=format$(ymax,"####0.0###")

msg$="X Min = %xmin$\n"
msg$=msg$+"Y Min = %ymin$\n"
msg$=msg$+"X Max = %xmax$\n"
msg$=msg$+"X Max = %ymax$"

message msg$, "Geometry Extents"

      !  ----------------------------------
      !  get the feature type for this feature
yesno "Draw a rectangle around the geometry ?", iyesno

if iyesno=0 then stop "Finished"

contour[
   start xmin, ymin
   line xmax, ymin
   line xmax, ymax
   line xmin, ymax
   line xmin, ymin
]

redraw
stop "Finished"
```



Backup the Current Part

This example macro will backup the current part by creating a new file name based on current file name.

```
a1$ = "This macro will make a backup copy of the current"
a2$ = "part by saving it under a filename which is the same"
a3$ = "as the current filename, but with '_backup' added to it."
      ! some variable strings are defined.
message "%a1$\n%a2$\n%a3$"
      ! This defines a message that will open when the macro is run.
check part_open, "You must have a part open to run this macro"
      ! Error checking in case a part is not open.
get_part_data part_name, pname$
get_part_data part_file, pfile$
      ! The name of the part and its path are assigned to strings.
ilen=len(pfile$)
      ! remove the /vnc from the end of the filename
f$=left$(pfile$,ilen-4)+"_backup.vnc"
      ! then add "_backup.vnc" to it
a$="OK to save current part as\n%f$"
yesno a$, iyesno
if iyesno=0 then stop "Backup not created"
      ! The user is prompted to save the file or not.
```




```
message "Backup was created"
      ! A dialog opens letting the user know that the backup is done.
save_part_as f$
      ! save the current part as the original filename
      ! if we don't do this now, using file > save will overwrite the backup file we
        just saved
save_part_as pfile$
stop "Finished"
      ! A dialog opens letting the user know that the macro is complete.
```

Create & Bag Solid Bodies

This sample will create 2 solids and put them into the body bag and then take one out. This macro does not require a part to be open.

```
      ! We will first create a message that describes what will happen.
```

```
a1$ = "This macro will create 2 solids and show"
a2$ = "them going in and out of the body bag."
message "%a1$\n%a2$"
      ! We will now make a new 3 axis vertical mill part that measures 4x3x1 inches.
new_part "Example.vnc"
set_part_data mdd_name, "VMill3a"
set_part_data units, 1
set_part_data stock_x1, 0
set_part_data stock_y1, 0
set_part_data stock_z1, 0
set_part_data stock_x2, 4
set_part_data stock_y2, 3
set_part_data stock_z2, 1
      ! The view is set to isometric and the part is zoomed to fit your screen.
set_view iso
zoom_view 0
      ! We make a triangular shape and extrude it along the depth axis by +1 inch.
        The geometry is deleted after the solid is made.
contour [
   start 0,  0
   line  2,  0
   line  1,  3
   line  0,  0
]
iref = ContourRef
deselect_all_geo
select_shape iref
redraw
extrude 0, 1
solref1 = SolidRef
deselect_all_geo
delete_shape iref
      ! A second triangular shape is made and extruded along the depth axis by +1
        inch. The geometry is deleted after the solid is made.
contour [
   start 2,  3
   line  3,  0
   line  4,  3
   line  2,  3
]
iref = ContourRef
deselect_all_geo
deselect_all_solids
select_shape iref
extrude 0, 1
solref2 = SolidRef
```

```
deselect_all_geo
delete_shape iref
redraw
      !   We will now start moving the bodies into and out of the Body Bag.
message "Put the first solid in the bag", "Bag Solids", 1
set_solid_bagged solref1, 1
redraw
      !   We generate a message to show if the bodies are bagged or not.
message "Check the bagged state of the 2 solids", "Bag Solids", 1
get_solid_bagged solref1, bag1
get_solid_bagged solref2, bag2
message "Solid 1 bagged = %bag1\nSolid 2 bagged = %bag2", "Bag
   Solids", 1
      !   We will now switch the status of each body.
message "Unbag the first solid and bag the second one", "Bag Solids",
   1
set_solid_bagged solref1, 0
set_solid_bagged solref2, 1
redraw
stop "Finished"
```

Convert A Part Between Inch & Metric

! change current part units from inch to metric

! or metric to inch, ie swap the current units

```
a1$ = "This macro will change the units of the current part"
a2$ = "from inch to metric, or metric to inch, scaling the"
a3$ = "geometry and solids accordingly"
message "%a1$\n%a2$\n%a3$"

check part_open, "You must have a part open to run this macro"


get_part_data units, iunits
if iunits=1 then goto inch

yesno "This part is Metric, Do you want to convert it to Inch ?",
   iyesno, "Inch Metric Macro"
if iyesno=0 then stop "Finished - no action taken"

      !   -------------------------
      !   convert from metric to inch
scale = 1/25.4
iunits= 1
```

```
goto scale

      !  -------------------------
      !  convert from metric to inch
:inch
scale = 25.4
iunits= 0

:scale

set_part_data units, iunits
      !  change the part units
      !  -------------------------
      !  any solids will have been scaled by changing the part units,
      !  now scale the geometry
select_all_geo
scale_geo scale

yesno "Scale the stock ?", iyesno, "Inch Metric Macro"
if iyesno=0 then redraw
if iyesno=0 then stop "Finished - stock size not changed"

      !  -------------------------
      !  get the current stock
get_part_data stock_x1, x1
get_part_data stock_y1, y1
get_part_data stock_z1, z1
get_part_data stock_x2, x2
get_part_data stock_y2, y2
get_part_data stock_z2, z2

      !  you would use these variables for a turned part
!get_part_data stock_z1, z1
!get_part_data stock_z2, z2
!get_part_data stock_rad, rr
      !  -------------------------
      !  change the current stock
set_part_data stock_x1, x1*scale
set_part_data stock_y1, y1*scale
set_part_data stock_z1, z1*scale
set_part_data stock_x2, x2*scale
set_part_data stock_y2, y2*scale
set_part_data stock_z2, z2*scale

      !  you would use these variables for a turned part
!get_part_data stock_z1, z1
```

```
!get_part_data stock_z2, z2
!get_part_data stock_rad, rr

zoom_view 0

message "Finished"
```

Save Geometry Data to a Text File

This macro looks at selected geometry and exports the geometric data to a text file. This macro is a good example of manpulatnig files outside of GibbsCAM.

```
a1$ = "This macro look at the selected geometry and"
a2$ = "print the data for that geometry to a text file,"
a3$ = "with the filename selected by the user."
message "%a1$\n%a2$\n%a3$"

check part_open, "You must have a part open to run this macro"
check part_mill, "This macro is not designed for turned parts"
get_num_feat_selected inum
if inum<1 then stop "No geometry selected"

      !  ------------------------
      !  create a file open dialog
file_dialog_new "Select output filename"
file_dialog_extension "Text files (*.txt)", "txt"
file_dialog_extension "All files (*.*)", "*"
file_dialog_show save, f$

      !  ------------------------
      !  open the file on unit 1 for writing
file1_open=0
file_open 1, f$, write
if FileError<>0 then goto file_error
file1_open=1

      !  ------------------------
      !  loop through all selected features and update the min/max
      !  xy data according to each features xy data
      !  try setting fmt$ to any of the next 3
      !  to get different formatted numbers
fmt$ = "+~~~0.0~~"
      !  ! output spaces in place of leading/trailing zeros
!fmt$ = "+###0.0##"
      !  do not output leading/trailing zeros
!fmt$ = "+0000.000"
```

```
      !   output leading/trailing zeros
for i=1 to inum

      !   ------------------------
      !   get the feature reference number for the 'i'th selected feature
   get_selected_geo_ref i, iref
      !   ------------------------
      !   get the feature type for this feature
get_feat_type iref, itype
   if itype=1 then goto label_point
   if itype=2 then goto label_line
   if itype=3 then goto label_circle
   if itype=4 then goto label_arc

   continue

:label_point
   get_feat_start iref, 1, xs, ys, zs

   xs$ = format$(xs, fmt$)
   ys$ = format$(ys, fmt$)
   zs$ = format$(zs, fmt$)

   a$ = "Point   X =%xs$  Y =%ys$  Z =%zs$"
   goto label_print

:label_line
   get_feat_start iref, 1, xs, ys, zs
   get_feat_end   iref, 1, xe, ye, ze

   xs$ = format$(xs, fmt$)
   ys$ = format$(ys, fmt$)
   zs$ = format$(zs, fmt$)
   xe$ = format$(xe, fmt$)
   ye$ = format$(ye, fmt$)
   ze$ = format$(ze, fmt$)

   a$ = "Line    Xs=%xs$  Ys=%ys$  Zs=%zs$"
   a$ = a$ + "  Xe=%xe$  Ye=%ye$  Ze=%ze$"
   goto label_print

:label_circle
   get_circle_data iref, 1, rad, xc, yc, zc

   xc$ = format$(xs, fmt$)
   yc$ = format$(ys, fmt$)
   zc$ = format$(zs, fmt$)
```

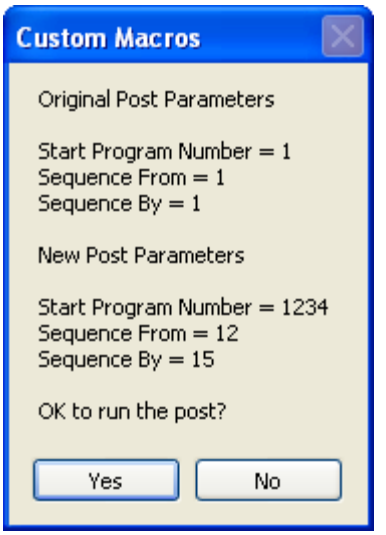
```
   rr$ = format$(rad, fmt$)

   a$ = "Circle  Xc=%xs$  Yc=%ys$  Zc=%zs$  R =%rr$"
   goto label_print

:label_arc
   get_feat_start iref, 1, xs, ys, zs
   get_feat_end   iref, 1, xe, ye, ze
   get_arc_data iref, 1, rad, dir, xc, yc, zc

   xc$ = format$(xc, fmt$)
   yc$ = format$(yc, fmt$)
   zc$ = format$(zc, fmt$)
   xs$ = format$(xs, fmt$)
   ys$ = format$(ys, fmt$)
   zs$ = format$(zs, fmt$)
   xe$ = format$(xe, fmt$)
   ye$ = format$(ye, fmt$)
   ze$ = format$(ze, fmt$)
   rr$ = format$(rad, fmt$)

   a$="Arc CCW"
   if dir=1  then a$ = "Arc CW "
   a$ = a$ + " Xc=%xc$  Yc=%yc$  Zc=%zc$  R =%rr$"
   a$ = a$ + "\n        Xs=%xs$  Ys=%ys$  Zs=%zs$"
   a$ = a$ + "  Xe=%xe$  Ye=%ye$  Ze=%ze$"
   goto label_print

:label_print
   file_write_text 1, a$
   if FileError<>0 then goto file_error

next i

file_close 1
if FileError<>0 then goto file_error

stop "Finished"

:file_error
   ierr=FileError
   if file_open=1 then file_close 1
   stop "File error, code = %FileError"
```

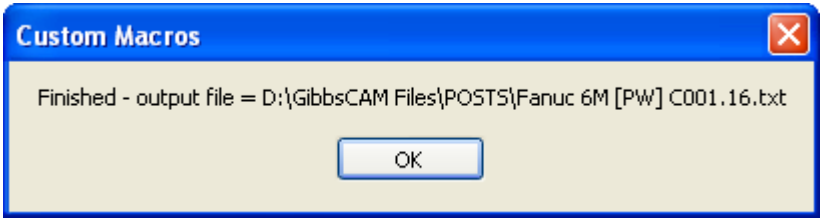
Run a Post Processor

This macro sets posting parameters, prompts the user for the post processor to be used and runs the post. The user can choose to view the posted code at the end. This sample is a good example of user and file interaction.

```
a1$ = "This macro will prompt for a post, set some"
a2$ = "post parameters and run the post to create code"
message "%a1$\n%a2$"
check part_open, "You must have a part open to run this macro"

      ! ---------------------------------------------
      ! create a file open dialog and get post filename
file_dialog_new "Select Post"
file_dialog_extension "Post files (*.pst)", "pst"
file_dialog_show open, post_file$
      ! ---------------------------------------------
      ! create a file save dialog and get code filename
file_dialog_new "Post Output File"
file_dialog_extension "All files (*.*)", "*"
file_dialog_show open, output_file$
      ! ---------------------------------------------
      ! get some post parameters
get_post_data, start_prog_num, post_start_prog_num
get_post_data, seq_from, post_seq_from
get_post_data, seq_by, post_seq_by

a$ = "Original Post Parameters\n"
b$ = "Start Program Number = " + format$(post_start_prog_num,
   "#####0")
a$ = a$ + "\n" + b$
b$ = "Sequence From = " + format$(post_seq_from, "#####0")
a$ = a$ + "\n" + b$
b$ = "Sequence By = " + format$(post_seq_by, "#####0")
a$ = a$ + "\n" + b$
      ! ---------------------------------------------
      ! change these post parameters
set_post_data, start_prog_num, 1234
set_post_data, seq_from, 12
set_post_data, seq_by, 15
      ! ---------------------------------------------
      ! get them again, to show they have been changed
```

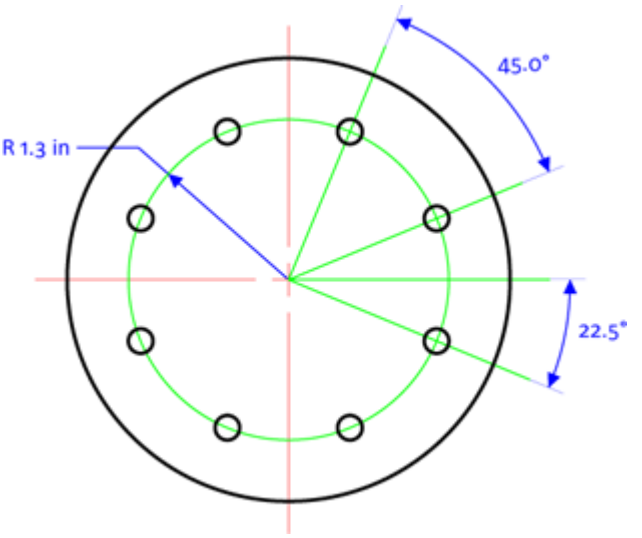
```
get_post_data, start_prog_num,
   post_start_prog_num
get_post_data, seq_from, post_seq_from
get_post_data, seq_by, post_seq_by

b$ = "New Post Parameters\n"
a$ = a$ + "\n\n" + b$
b$ = "Start Program Number = " +
   format$(post_start_prog_num, "#####0")
a$ = a$ + "\n" + b$
b$ = "Sequence From = " +
   format$(post_seq_from, "#####0")
a$ = a$ + "\n" + b$
b$ = "Sequence By = " + format$(post_seq_by,
   "#####0")
a$ = a$ + "\n" + b$
     ! ------------------------------------------
     ! run the post
a$ = a$ + "\n\nOK to run the post?"
yesno a$, iyesno
if iyesno = 0 then stop "Stopped without running post"

set_post_data, post_file, post_file$
set_post_data, output_file, output_file$

run_post
message "Finished - output file = %output_file$"
yesno "View the output ?", iyesno
if iyesno = 0 then stop "Finished"
run_exe "\Windows\System32\Notepad.exe", output_file$
```





Loop Example

Following is an example of how a "For/ Next Loop" (for i=1 to 3) variable works. In the example, we will make a variable loop to create a bolt-circle as shown to the right. The bolt-circle will have a radius of 1.3 with 8 holes equally spaced. The first hole will be located at 22.5°. The positive direction of rotation is c/ cw.



Known Values

- 1.3 = r1 (radius).
- 8 = n1 (# of holes).
- 22.5° = a1 (angle of hole 1)
- 360/8 = 45°

Executing the loop

The loop is executed with the following 10 lines of code: Note, the line numbers would not be used in the macro.

```
1. aa = a1
      !  22.5°
2. da = (360/n1)
      !  45°
3. points[
      !  start creating a set of points
   4. for i=1 to n1
      !  loop 8 times
   5. xx = r1*cos(aa)
      !  x position of hole
   6. yy = r1*sin(aa)
      !  y position of hole
   7. point xx,yy
      !  points 1 to 8
   8. aa = aa + da
      !  angle of next hole going ccw
   9. next i
      !  increment loop count i
```

```
10. ]
        !    end creating set of points
```

Results of 1st loop

1. Variable aa starts as 0°. Variable a1 starts as 22.5°.

2. Variable da starts as 45° (360°/8).

3. Macro command to start creating a set of points.

4. Begin the loop, set the variable i to 1.

5. Set variable xx to equal 1.3*cos(22.5°) or xx1.2010.

6. Set variable yy to equal 1.3*sin(22.5°) or yy0.4975.

7. Create one point at X,Y location using values xx,yy.

8. Reset variable aa to equal previous aa + da or 22.5°+45°.

9. Add 1 to variable I, making it 2. Since 2 is less than or equal to 8 (n1), jump back to line 5.

Results of 2nd loop

Please note that loops 2-8 begin at line 5.

5. Set variable xx to equal 1.3*cos(67.5°) or xx0.4975.

6. Set variable yy to equal 1.3*sin(67.5°) or yy1.2010.

7. Create one point at X,Y location using values xx,yy.

8. Reset variable aa to equal previous aa + da or 67.5°+45°.

9. Add 1 to I, making it 3. 3 is still <= 8, so jump back to line 5.

Results of the final loop

5. Set variable xx to equal 1.3*cos(337.5°) or xx1.2010.

6. Set variable yy to equal 1.3*sin(337.5°) or yy-0.4975.

7. Create one point at X,Y location using values xx,yy.

8. Reset variable aa to equal previous aa + da or 337.5°+45°.

9. Add 1 to I, making it 9. Since 9 is greater than 8 (n1), finish the loop and continue to line 10.

10. End of set of points.

The Math

i1 = 000.0° + 022.5° = 022.5° = x1.2010 y0.4975

i2 = 022.5° + 045.0° = 067.5° = x0.4975 y1.2010

i3 = 067.5° + 045.0° = 112.5° = x-0.4975 y1.2010

i4 = 112.5° + 045.0° = 157.5° = x-1.2010 y0.4975

i5 = 157.5° + 045.0° = 202.5° = x-1.2010 y-0.4975

i6 = 202.5° + 045.0° = 247.5° = x-0.4975 y-1.2010

i7 = 247.5° + 045.0° = 292.5° = x0.4975 y-1.2010

i8 = 292.5° + 045.0° = 337.5° = x1.2010 y-0.4975

FULL MACRO EXAMPLES

This example is a fully working macro program. This program creates a dialog that accepts user input to create a trapezoid, load a saved process file and create toolpath to machine the shape. For this to work a macro file is created (“Macro3.mac) which references an accompanying dialog file (“Macro3.dlg). The dialog references an image file that helps the user visualize the shape. A saved process file is referenced by the macro file so that tool and process data can be loaded.

First Example - “Macro3”

This is the actual code for the macro. If you do not have the sample you can copy the text and save it. The green text denotes our comments to help you understand the code.

Macro3.mac Code

```
dialog "Macro3.dlg"
       !   This calls the dialog file. The macro file is the basis of the program but the
           user interacts with the dialog. Since this macro can't do anything until it has
           data fro the user the dialog must be called first. Looking at the dialog file at
           the is point is recommended so that you become familiar with its
           components. The code for “Macro3.dlg” can be found on page 78.
! a1 = incline angle
! a2 = included angle
```

```
! ww = width
! ll = length
! xs = start point
! ys = start point
```

! These are comments to help you keep track of what the variables represent. A liberal use of comments is strongly recommended.

```
degrees  ! switch all trig functions to degrees
```

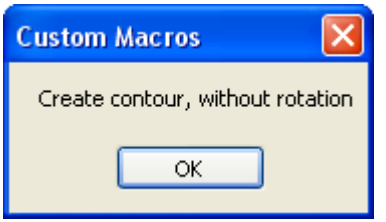
! Here we see an inline comment letting us know we're switching value types.

```
d2 = (ll * tan(90-a2))
w2= (ww-d2-d2)      ! length of short end
```

! As the comment says, this calculates the length of the short end of the trapezoid. The user inputs the length of the long end and the included angle of the sides, the system calculates the short end.

```
if messages=1 then message "Create contour, without rotation"
```

! If the user selected the "Show Messages" option a message will open that keeps the user informed. Please note that the user must click the "OK" button before the macro continues.




```
contour [
   start xs, ys
   line xs+ll, ys+d2
   line xs+ll, ys+d2+w2
   line xs, ys+ww
   line xs, ys
]
```

! The contour command creates the trapezoid from the user-supplied data.

```
iref = ContourRef
```

! The contour is assigned the variable name "iref".

```
if messages=1 then message "select contour"
```

! If messages are on the user is told the system is selecting the shape it just made.

```
clear_select
select_shape iref
redraw
```

! The macro first makes sure nothing is selected, then it selects "iref" and finally redraws the screen.

```
if messages=1 then message "Rotate to inclination angle"
```

! If messages are on then the user is told the shape is going to be rotated.

```
rotate_geo xs,ys,a1
redraw
```

! The shape is rotated about the start point by the user-supplied angle.

```
if messages=1 then message "Set markers, load process and create ops"
```
! If messages are on then the user is told that toolpath is going to be created.

```
set_markers 1, 1, 0.5, 1, 0.5
```
! The machining markers are set to cut on the left ("1") and to start and end halfway along the first feature ("1. 0.5").

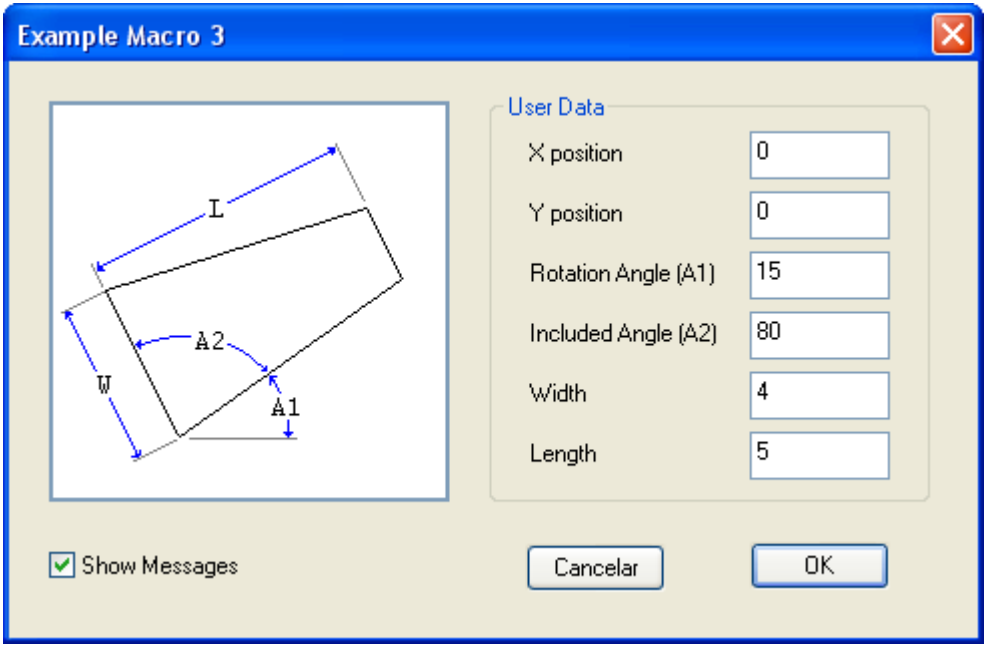
```
load_process "macro3.prc"
```
! The saved process file is loaded. Since only the file name is referenced the process file must be in the same directory as the macro.

```
calc_process
clear_select
redraw
```
! The process is applied to the shape. The selection is cleared and the screen is redrawn so the user can see the results.

```
if messages=1 then message "Finished"
```
! If messages are on then the user is told that the macros has done its job. The macro is not set up to save the file. A good way to learn more about the macros is to add a message warning the user about saving the file or even adding code to this file that will save the part.

Macro3.dlg Code

This is the actual code for the macro dialog. If you do not have the sample you can copy the text and save it. The green text denotes our comments to help you understand the code.

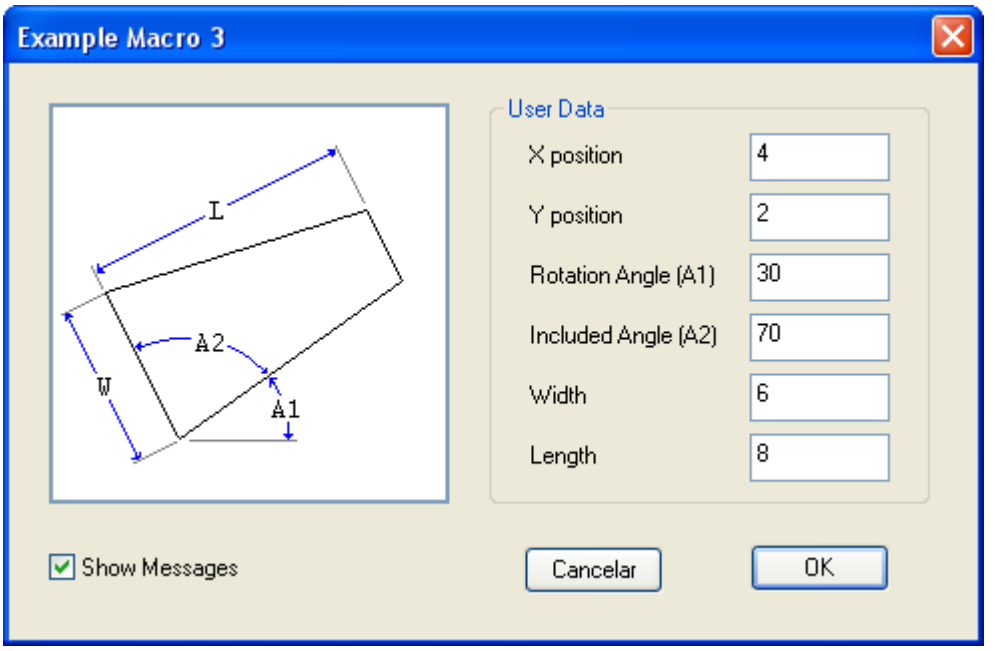


```
dialog "Example Macro 3",250,250,490,320
      !   Here we set the name of the dialog and its size.
! a1 = incline angle
! a2 = included angle
! ww = width
! ll = length
! xs = position
! ys = position
      !   These are comments to help you keep track of what the variables represent.
          A liberal use of comments is strongly recommended.
image 20, 20, 200,200,"Macro3.bmp"
      !   The dialog's image is placed.
frame 240, 16, 220,204, "User Data"
      !   The frame that contains the user input fields is placed. Frames are optional
          but are very useful for organizing data.
label 260, 39, 100, 24, "X position"
label 260, 69, 100, 24, "Y position"
label 260, 99, 100, 24, "Rotation Angle (A1)"
label 260, 129, 100, 24, "Included Angle (A2)"
label 260, 159, 100, 24, "Width"
label 260, 189, 100, 24, "Length"
      !   The text labels for user input are placed.
input 370, 35, 70, 24, xs, 0
input 370, 65, 70, 24, ys, 0
input 370, 95, 70, 24, a1, 15
input 370, 125, 70, 24, a2, 80
input 370, 155, 70, 24, ww, 4
input 370, 185, 70, 24, ll, 5
      !   The text input boxes are placed. Note the variable names and default values.
check 20, 240, 200, 24, "Show Messages", messages, 1
      !   An option is placed for the user. If this option is checked then the macro
          will show a text message at certain points during the macro's execution. The
          messages let the user know what is going on. This sets the variable
          "messages" to true ("1").
cancel 260, 240, 70, 24
      !   This creates a cancel button which is mandatory.
ok 370, 240, 70, 24
      !   This creates an okay button which is mandatory. That is all there is to the
          dialog file. All the data collected here gets passed to the macro file.
```

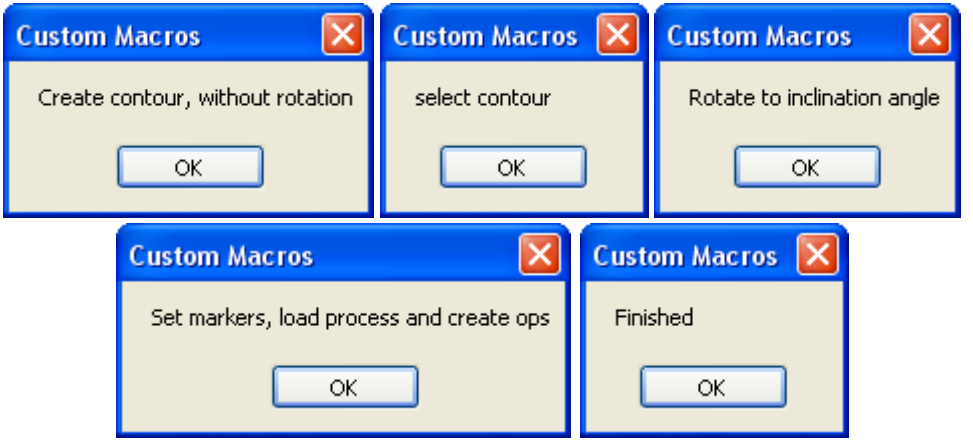
The Results of the Macro

This macro requires an open file. We start with an empty 3-axis mill part.

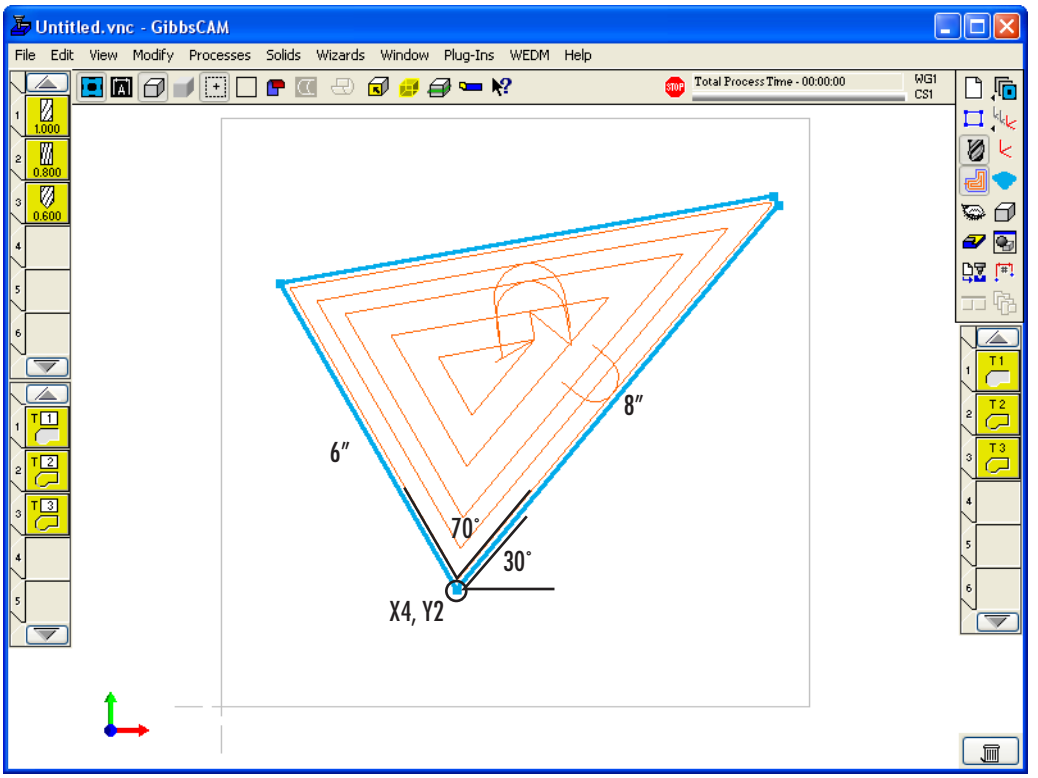
The user inputs data.



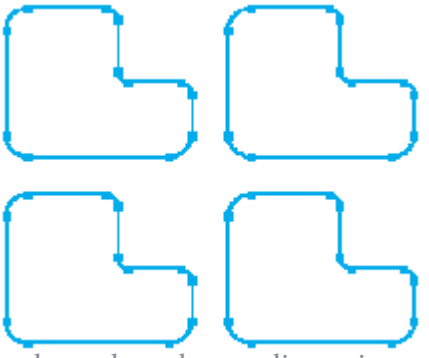
The user gets a series of messages.



The final results. The shape, tools. processes and operations are all created by the macro.



Second Example - “Macro2”



Macro2.mac

As with the first example, this is the complete code for a macro. This code is not as heavily commented, instead the comments are more focused on items the first macro does not have. This macro collects input from a user to create a six-sided shape and will then repeat that shape a number of times.

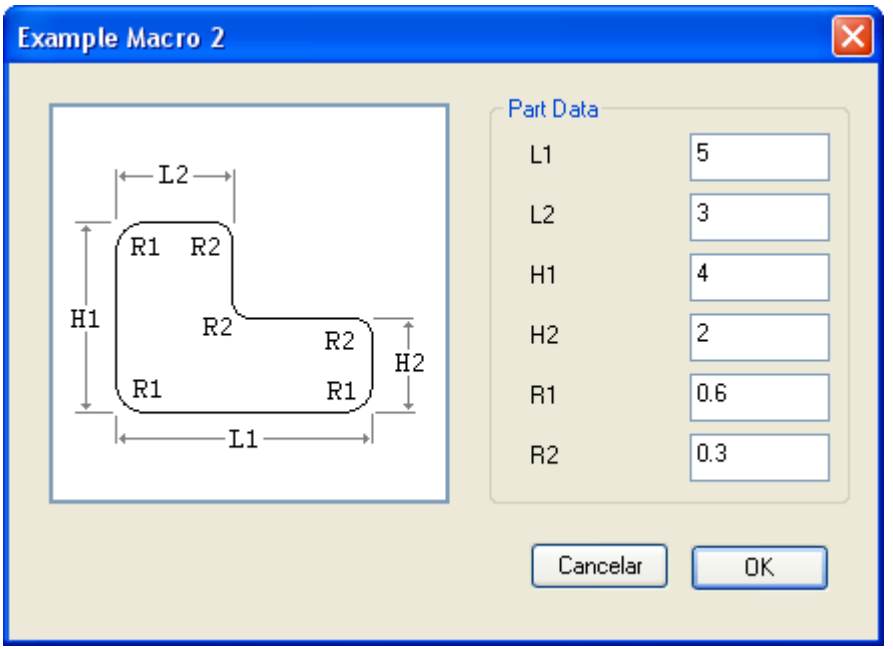
```
dialog "Macro2.dlg"
      ! The file “Macro2.dlg” is opened to gather the shape dimensions. “Macro2.dlg” can be found on page 83.
dialog "Macro2repeats.dlg"
      ! After data has been collected from “macro2.dlg” this dialog, “Macro2Repeats.dlg” opens to determine the number of repeats.“Macro2Repeats.dlg” can be found on page 85.
! l1 = long length
! l2 = short length
! h1 = long height
```

```
! h2 = short height
! r1 = large fillet
! r2 = small fillet
      !  These are the variables collected by "macro2.dlg".
! xs = x start
! ys = y start
! dx = x spacing
! dy = y spacing
! nx = number in x
! ny = number in y
      !  These are the variables collected by "macro2repeats.dlg".
if nx<2 then stop "Invalid number of parts in X (2 to 5)"
if nx>5 then stop "Invalid number of parts in X (2 to 5)"
      !  the macro can only create up to 5 shapes in an X direction. If less than 2 or
         more than 5 parts are requested then the macro will stop.
if ny<2 then stop "Invalid number of parts in Y (2 to 4)"
if ny>4 then stop "Invalid number of parts in Y (2 to 4)"
      !  the macro can only create up to 4 shapes in a Y direction. If less than 2 or
         more than 4 parts are requested then the macro will stop.
x1=xs
      !  The variable "x1" is set equal to the X start position the user specified.
for i=1 to nx
      !  a count is started
   y1=ys
      !  The variable "y1" is set equal to the Y start position the user specified.
   for j=1 to ny
      !  a count is started
      contour [
         start x1+r1, y1
         line x1+l1-r1, y1
         arc x1+l1-r1, y1+r1, x1+l1, y1+r1, ccw
         line x1+l1, y1+h2-r2
         arc x1+l1-r2, y1+h2-r2, x1+l1-r2, y1+h2, ccw
         line x1+l2+r2, y1+h2
         arc x1+l2+r2, y1+h2+r2, x1+l2, y1+h2+r2, cw
         line x1+l2, y1+h1-r2
         arc x1+l2-r2, y1+h1-r2, x1+l2-r2, y1+h1, ccw
         line x1+r1, y1+h1
         arc x1+r1, y1+h1-r1, x1, y1+h1-r1, ccw
         line x1, y1+r1
         arc x1+r1, y1+r1, x1+r1, y1, ccw
      ]
      !  This code creates the shape from the user-input using simple math.
   y1 = y1 + dy
      !  The variable "y1" is incremented by the Y offset
```

```
    next j
        !   The next contour is created in Y. This will repeat a number of times equal to
            "ny".
    x1 = x1 + dx
        !   The variable "x1" is incremented by the x offset
next i
        !   The next contour is created in X. This will repeat a number of times equal to
            "nx". Basically, this code creates a number of shapes in Y, then steps over in
            X and creates the same number of shapes. This will repeat until "nx" is
            reached.
clear_select
        !   This deselects any item to ensure it is not accidentally deleted.
redraw
        !   This forces a redraw so the user can see the results of the macro.
```

Macro2.dlg

This code sets the dialog "Example Macro 2" which collects user input for defining the shape that will be repeated.




```
dialog "Example Macro 2",250,250,440,320
        !   The size and position of the dialog are set.
! l1 = long length
! l2 = short length
! h1 = long height
! h2 = short height
! r1 = large fillet
```

```
! r2 = small fillet
```
! These are simply comments about the variables and what they represent.

```
! xs = x start
! ys = y start
! dx = x spacing
! dy = y spacing
! nx = number in x
! ny = number in y
```
! These are simply comments about the variables and what they represent. These variables are collected in "Macro2Reapeats.dlg".

```
image   20,  20, 200,200, "Macro2.bmp"
```
! The example graphic is positioned.

```
frame  240,  16, 180,204, "Part Data"
```
! A frame that contains the user input is created.

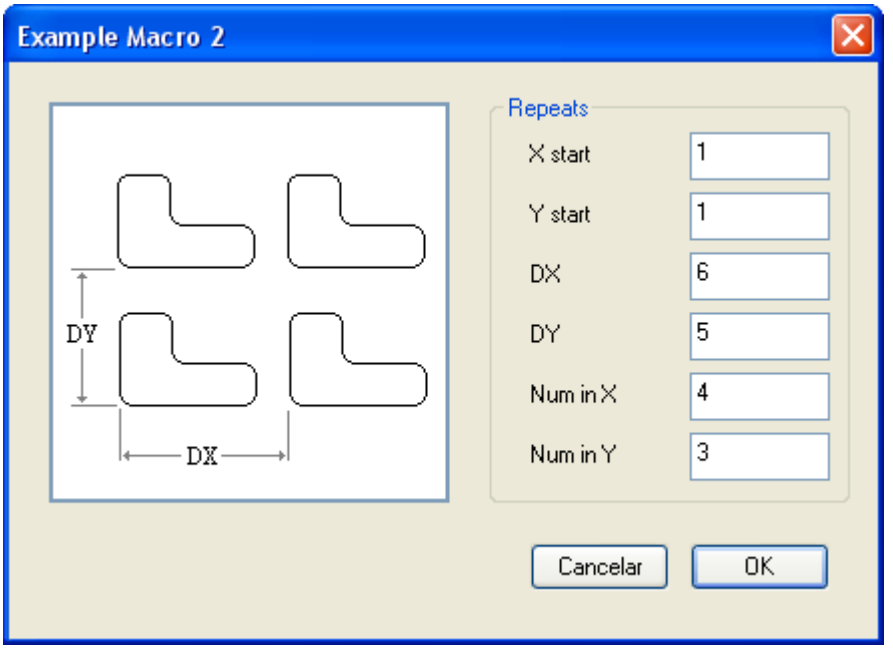
```
label  260,  39,  70, 24, "L1"
label  260,  69,  70, 24, "L2"
label  260,  99,  70, 24, "H1"
label  260, 129,  70, 24, "H2"
label  260, 159,  70, 24, "R1"
label  260, 189,  70, 24, "R2"
```
! Text labels for the input fields are set.

```
input  340,  35,  70, 24, l1, 5
input  340,  65,  70, 24, l2, 3
input  340,  95,  70, 24, h1, 4
input  340, 125,  70, 24, h2, 2
input  340, 155,  70, 24, r1, 0.6
input  340, 185,  70, 24, r2, 0.3
```
! Text entry boxes are created. A variable and default is set for each field.

```
cancel 260, 240,  70, 24
ok     340, 240,  70, 24
```
! The required "Cancel" and "OK" buttons are created. When this dialog is closed the macro will continue to run which means Macro2Repeats.dlg will open.

Macro2Repeats.dlg

This code sets the second dialog “Example Macro 2” which collects user input for how the shapes are to be offset.



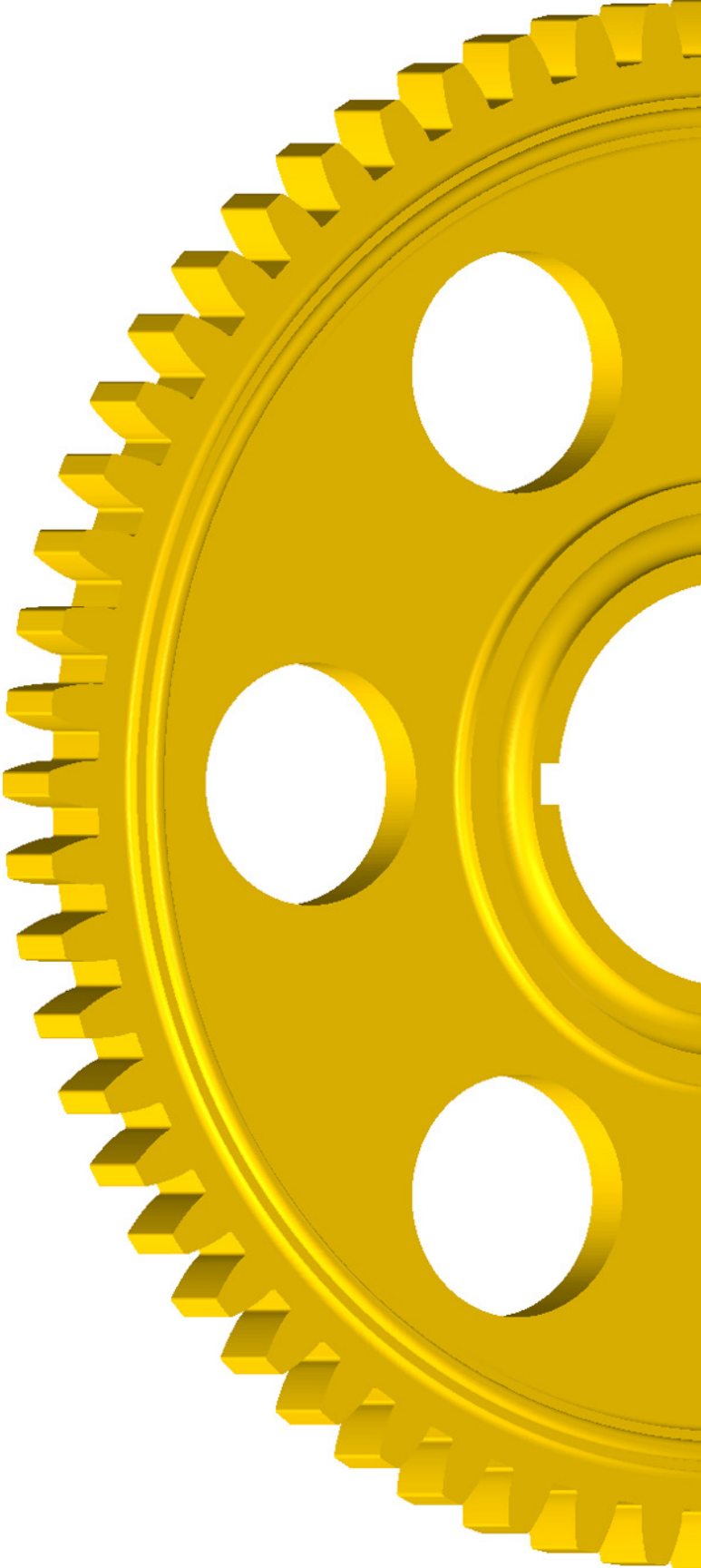

```
dialog "Example Macro 2",250,250,440,320
      !   The size and position of the dialog are set.
! xs = x start
! ys = y start
! dx = x spacing
! dy = y spacing
! nx = number in x
! ny = number in y
      !   These are simply comments about the variables and what they represent.
image   20,  20, 200,200, "Macro2repeats.bmp"
frame  240,  16, 180,204, "Repeats"
      !   The example graphic and frame are positioned.
label  260,  39,  70, 24, "X start"
label  260,  69,  70, 24, "Y start"
label  260,  99,  70, 24, "DX"
label  260, 129,  70, 24, "DY"
label  260, 159,  70, 24, "Num in X"
label  260, 189,  70, 24, "Num in Y"
input  340,  35,  70, 24, xs, 1
input  340,  65,  70, 24, ys, 1
input  340,  95,  70, 24, dx, 6
input  340, 125,  70, 24, dy, 5
```

```
input  340, 155,  70, 24, nx, 4
input  340, 185,  70, 24, ny, 3
```

! Text labels for the input fields and the accompanying text entry boxes are created. A variable and default is set for each field.

```
cancel 260, 240,  70, 24
ok     340, 240,  70, 24
```

! The required “Cancel” and “OK” buttons are created. When this dialog is closed with “OK” the macro will continue to run with values for all of the variables.



INDEX

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z