Other features of parametric programming

As you have seen, this chapter has been working from common CNC features toward the less popular. In this discussion, we simply mention CNC features that are equipped with some versions of parametric programming that have nothing to do with system variables. It will be up to you to reference your control manufacturer’s programming manual to determine whether or not you have one or more of these capabilities in your particular version of parametric programming.

Protection for important programs

If you have important parametric programs that you do not want accidentally deleted or modified, it would be nice to protect them. This ability is especially handy with parametric programs related to the family-of-parts, user-created canned cycle, and utility application categories. Fanuc, for example, allows you to protect the entire series of programs numbered in the 9000 series. Unfortunately, the parameters that are related to this function vary even among different Fanuc control models, meaning you will have to reference your programming manual to determine the parameters involved.

Labeling variables on the display screen page

Some versions allow you to label variables on the display screen page. This can be helpful to alert the operator to the fact that certain variables that should not be changed. When using global variables and system constants, for example, you wouldn’t want an operator accidentally changing an important variable. This is also quite helpful for variables used as calibration values needed by probing systems.

In custom macro B, for example, permanent common variables (the #500 series) can be labeled with a SETVN command (SETVN stands for SET Variable Name). The command

SETVN = 500 (COUNTER)

for example, places the message COUNTER next to permanent common variable #500. In custom macro B, this message is limited to being eight characters long.

User defined G and M codes

Some versions of parametric programming allow you to actually invoke your custom macros with G and M codes. This can make your parametric programs appear to be true G and M code functions. Since this can be a very helpful function (unique to custom macro B), we discuss it in detail.

As you know, you can easily command that a custom macro be executed by giving the G65 command. Included within the G65 command is a P word that specifies the program number to be executed. Though this is not at all a bad way to call a Custom Macro, there is an easier way.

When you have an often used custom macro, for example, it is sometimes helpful to assign it its own G code. You can actually have a G code of your choosing call the Custom Macro. To do this, you will first need to make sure that you do not overwrite a currently used G code. Look in the list of G codes included in the programming manual to find an unused G code number. Also,
keep in mind that you can use G codes for calling custom macros that range from 100 to 255 (not just up to 99). This should help you avoid conflicts.

There are certain parameters related to how G codes are created. The actual parameter numbers will vary even from one control model to another, so you will have to look in the custom macro section of your programming manual to find the exact parameter numbers. Here are the user defined G code parameters and related program numbers for 10, 11, and 15 series Fanuc controls.

In order to execute a custom macro by a user defined G code, the custom macro program number must be a number between O9010 and O9019. For 10, 11, and 15, series controls, parameters 7050-7059 allow the user to specify which G code numbers are related to G code calling program numbers. Say for example, parameter number 7050 is set to 12. Whenever a G12 is executed, the control will execute program O9010. The control will execute program O9011 whenever the G code specified by the value of parameter 7051 is read. The list of letter address arguments must still be specified.

When you think about it, creating G codes does not really help you all that much. It simply allows you to specify a G code, say G12, instead of G65 P1004. However, this can sometimes be helpful to solve certain G code numbering compatibility problems. For example, say you have a Yasnac control that has two nice circle milling G codes (G12 for clockwise and G13 for counterclockwise). With custom macro B, you intend to write a user-created canned cycle for circle milling on your Fanuc control that duplicates the Yasnac style. If you use the same letter address arguments as are used by Yasnac, if you name your programs as O9010 and O9011, and if you set parameters 7050 and 7051 (10, 11, or 15 series controls only) to 12 and 13, programming for circle milling operations will be consistent between Fanuc and Yasnac.

Defining your own M codes. In similar fashion, you have the ability to create new and modify the function of current M codes with custom macro B. This is done in much the same way you create user defined G codes. For most controls, programs O9001 through O9009 are used as user defined M codes. Parameters 7071 through 7079 are used to specify the M code numbers involved (on 10, 11, and 15 series controls). If for example, parameter 7071 is set to 14, program number O9001 is called whenever M14 is executed. By comparison to user defined G codes, most control models do not allow a list of arguments to be passed with user defined M codes. This means, of course, that you will only be using user defined M codes to activate custom macros that do not require arguments to be passed.

Modifying current M codes. It is important that you know you have the ability to change the function of any M code your machine is currently using. You can even activate the original M code from within the custom macro program the M code is calling. One time this is especially helpful is with the machining center M06 command.

Say you have a vertical machining center that uses M06 to make tool changes. However, the normal function of M06 with most machines is only to make tool changes. If the machine is not currently resting at the tool change position (normally the Z axis zero return position for vertical machining centers) when the M06 is commanded, the machine will go into alarm state.

With custom macro B, you can change the function of M06 so that whenever it is activated, the machine will first go to the zero return position (even incorporating the spindle orient M19 on
the way) and then make the actual tool change. Here’s how (for a 10, 11, or 15 series Fanuc control).

First, set parameter number 7071 to 6. This will cause the control to execute program number O9001 whenever M06 is executed. Next, load this program into the control.

O9001 (Program number)
G91 G28 Z0 M19 (Send the machine to tool change position, orient spindle)
M06 (Change tools)
G90 (Reselect absolute mode)
M99 (End of Custom Macro)

With this program, first the machine will be sent to its tool change position. In the next command, the M06 will actually make the tool change. Note that an M code of the same number specified in a user defined M code program will cause the control to activate its original function. (A custom macro cannot call itself.) Finally, we save some work in the main program by reselecting the absolute mode.

Adding M codes. As with G codes, you can add new M codes. As stated during the discussion of the #1000 series system variables, you can even access outside devices with user defined M codes. To complete the example given during the discussions of the #1000 series system variables, here we show how you could make M13 activate the outside device.

First, set the first available M code parameter (say number 7071) to 13. From this point, any time the control reads an M13, it will activate Custom Macro O9001. Next, the outside device activating custom macro is loaded into the control as program number O9001.

O9001 (Program number)
#1100 = 1 (Activate indexer)
N1 IF [#1000 EQ 1] GOTO 99
GOTO 1
N99 M99

As stated earlier, the #1100 = 1 statement applies a twenty-four volt signal to terminal location UO000 (to which the indexer is connected). Then the custom macro program goes into a waiting loop, looking for the confirmation signal from the outside device. As soon as UI000 is closed (by the outside device), system variable #1000 will be equal to one, ending the loop and letting the control continue.

Making machines compatible with regard to M codes. One more time that user defined M codes can really help is when you have similar machines made by different machine tool builders. Say, for example, you have to similar turning centers, each having the same approximate horse power, chuck size and range of travels. It is quite likely in this case that you will want to run programs on either machine, depending on which is available first.

Unfortunately, machine tool builders have not standardize on (even some rather common) M codes. For example, one machine may require an M23 to select the spindle’s low range and M25 to select high range. Another may use M41 for low range and M42 for high range. Yet another may require no M code (only one spindle range) but the control for this machine may hang up if any unrecognizable M code is read.
Rather than forcing the operator or programmer to having to constantly deal with this incompatibility problem (constantly changing programs), you can create user defined M codes for one of the machines involved. In the spindle range example, say you want to use M41 and M42 for low and high spindle range on both machines. On the machine that currently uses M23 and M25, set parameter 7071 to 41 and parameter 7072 to 42 (for 10, 11, and 15 series Fanuc controls). Then load these two simple programs into the controls memory.

O9001 (Program number)
M23 (Select low spindle range)
M99 (End of Custom Macro)

O9002 (Program number)
M25 (Select high spindle range)
M99 (End of Custom Macro)

Once this is done, whenever this control reads an M41, it will execute program O9001, which of course selects the low spindle range. When it reads an M42, program O9002 is executed, selecting the high spindle range.

If you happen to have a machine that has only one spindle range, simply add two user defined M codes (for M41 and M42) that execute empty custom macro programs (programs with a program number followed immediately by M99). Though these Custom Macros will do absolutely nothing, at least the control will not hang up when an M41 or M42 is read.

Keep in mind that the spindle range selection for turning centers is but one example of incompatibility problems with M codes. Other examples include bar feed activating M codes, part catcher M codes, tailstock M codes, indexer M codes, pallet changer M codes, and tool changer activation M codes.

**Communicating data through the communications port**

Some versions of parametric programming allow external data output commands designed to send data through the communications (RS-232C) port. Some even allow data transfers in both directions (to and from the CNC control). These functions are especially helpful when you wish to transmit production control related data, such as workpiece count, time in cycle, and statistical process control data (when probing systems are used).

Fanuc’s custom macro B is limited to sending data out, from the CNC control to some receiving device. The receiving device usually takes the form of a (serial) printer or personal computer. Four commands are related.

- **OPEN** - Tells the control to open the serial port for outputting data
- **BPRNT** - Outputs data (without decimal points)
- **DPRNT** - Outputs data (with decimal points)
- **PCLOS** - Closes the serial port at the end of a transmission

Since most values being communicated require decimal point placement, the DPRNT command is the more popular of the printing commands.
Say you have a custom macro program that uses a probe to measure the width of a slot. After machining, the probe measures the slot and you wish the slot width to be automatically reported to your statistical process control (SPC) computer. You wish your parametric program to report the current number of the workpiece in the production run, as well as the slot width. We’ll say that permanent common variable #500 is used to track the current part count and common variable #100 is used in which to store the slot width. Here is a series of commands that will send this data through the communications port. Of course, the outside device must be connected and ready to receive this data at the time these commands are read.

```
POPEN (Open the serial port)
DPRNT [PART*NUMBER*] #500 40 [SLOT*WIDTH*] #100 24
PCLOS (Close the serial port)
```

The POPEN command opens the serial port. In the DPRNT command, the words PART NUMBER and SLOT WIDTH simply specify notes that will appear next to the values stored in #500 and #100. The numbers 40 and 24 specify the decimal point format for the numbers being printed. The format “40” is telling the control to print up to four digits to the left of the decimal point and no (0) digits to the right of the decimal point. Since #500 is an integer number (whole number), there will be no portion of a whole number involved. This format (40) allows the printing of four digit numbers (up to 9999).

In similar fashion, the 24 after #100 will have the control print two digits to the left of the decimal point and four places to the right. This will print in xx.xxxx format for the slot width.

The PCLOS command closes the serial port after the transmission.

If this is part number 45 and the slot is measured as 2.2018 wide, the actual message, if printed, will appear as follows:

```
PART NUMBER 0045 SLOT WIDTH 02.2018
```

**Conclusion to other CNC features of parametric programming**

Though we have exposed you to many of the CNC features of parametric programming, it is likely that you have more available with your particular version. You must reference your own control manufacturer’s programming manual to determine what else may be available. And remember, some CNC features are very obscure, and the discussions given may be somewhat difficult to follow. However, with study, you may find you have many more CNC features available.
Understanding argument assignment number two

As mentioned during our discussion of variables in chapter two, there are two methods of argument assignment in custom macro B. To this point, all of our examples have used argument assignment number one. In this section, we will present argument assignment number two.

As you know, arguments placed in the call statement (the G65 command) are passed to the custom macro program. Each argument contains a letter address and a numerical value and is used to represent something about the application. You know that certain letter addresses (G, N, L, O, and P) cannot be used as arguments. We also asked you to avoid I, J, and K.

Once in the actual custom macro program, you know you can no longer reference the arguments by their letter addresses. The letter addresses must be referenced in the custom macro program by local variables. You also know that the local variable numbers (#1 through #26) don’t make much sense. You must reference a special chart in order to determine the local variable representations for all arguments used in the call statement. This discussion will explain why the local variable assignments in argument assignment number one don’t make much sense. It will also explain why we asked you to avoid using I, J, and K for most all custom macro applications.

Local variable assignments for argument assignment number two

This chart shows how local variables are used to represent arguments in argument assignment number two. Notice that ten sets of I, J, and K letter address arguments are used in conjunction with one set of A, B, and C arguments.

<table>
<thead>
<tr>
<th>A...#1</th>
<th>B...#2</th>
<th>C...#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>l(1)...#4</td>
<td>J(1)...#5</td>
<td>K(1)...#6</td>
</tr>
<tr>
<td>l(2)...#7</td>
<td>J(2)...#8</td>
<td>K(2)...#9</td>
</tr>
<tr>
<td>l(3)...#10</td>
<td>J(3)...#11</td>
<td>K(3)...#12</td>
</tr>
<tr>
<td>l(4)...#13</td>
<td>J(4)...#14</td>
<td>K(4)...#15</td>
</tr>
<tr>
<td>l(5)...#16</td>
<td>J(5)...#17</td>
<td>K(5)...#18</td>
</tr>
<tr>
<td>l(6)...#19</td>
<td>J(6)...#20</td>
<td>K(6)...#21</td>
</tr>
<tr>
<td>l(7)...#22</td>
<td>J(7)...#23</td>
<td>K(7)...#24</td>
</tr>
<tr>
<td>l(8)...#25</td>
<td>J(8)...#26</td>
<td>K(8)...#27</td>
</tr>
<tr>
<td>l(9)...#28</td>
<td>J(9)...#29</td>
<td>K(9)...#30</td>
</tr>
<tr>
<td>l(10)...#31</td>
<td>J(10)...#32</td>
<td>K(10)...#33</td>
</tr>
</tbody>
</table>

To actually make call statements with argument assignment number two, you specify a series of I, J, and K arguments in conjunction with A, B, and C (if required). Though it may not be entirely clear yet, here is an example call statement given in argument assignment number two.


As with argument assignment number one, all arguments must include a decimal point in the numerical value.

When the custom macro program is executed, local variables #1, #2, and #3 will be used to represent A, B, and C (just as in argument assignment number one). The first I the control
comes across in this call statement (I2.235) will be stored in local variable #4. The first J will be
stored in #5. The first K in #6. The second I (I1.2) will be stored in local variable #7. The
second J in #8. The second K in #9. This continues through #33 representing the tenth
occurrence of K.

How does the control know the difference? Actually, this is an automatic function, one that is
rather transparent to the unsuspecting programmer. As the control assigns the local variables, if
it happens to come across an I (the first one), it automatically places its value in local variable
#4. If it comes across a second I, it places its value into #7. If it comes across a K (the first one),
it places it into #5.

By the way, this is why, we warned you against using I, J, and K when you are using argument
assignment number one. If you avoid I, J, and K, you can rest assured that you will never have a
problem caused with the control misunderstanding which argument assignment type you are
using.

There may be times when you do wish to use I, J, and K in argument assignment number one.
Maybe you are running out of letter addresses to choose from. Maybe the letter address name
make so much sense, you feel you should use it (possibly using argument I as a flag to specify
incremental mode). In these cases, you must know how the control will behave with regard to
local variable assignments.

First of all, if you use only one of I, J, or K, you will never have a problem. This means if you
feel that it just makes so much sense to use letter address I, J, or K, by all means, do so. But if
you use I, J, and/or K together, you must be very careful with their placement order in the call
statement. If you do not specify them in their correct alphabetical order, the control will not
assign your local variables in the way you expect. Consider this example call statement.


In this example, everything will work just fine. The X will be assigned as #24, Y as #25, Z as
#26, D as #7, I as #4, and K as #5 (right from the table from argument assignment number one).

However, if by mistake, you reverse the order of I and J, you’re in for an unexpected problem.
Consider this new and incorrect call statement.


In this case, notice that J comes before I. J will be correctly assigned as #5. But when the
control comes across the I, it will be taken as the second I in argument assignment number two
(since it follows the first J). Instead of being assigned as #4, it will be assigned as #7, which by
the way, will overwrite the value currently stored in #7 by the D argument. This, of course, will
cause problems in the custom macro program.

When to use argument assignment number two

Do to the complexity of setting up argument assignments for custom macro programs with
argument assignment number two, you should only consider using it when the number of
arguments needed in the call statement is variable. Consider this circular screw machine cam
eexample in figure 6.9.
Say this is but one part in a family of cams. You need to develop a parametric program that will drive the tool around the cam, automatically generating the various rises and falls needed on the cam. This particular cam has five rises and falls. However, other workpieces in the family have a different number. Some have four, some have three, some have eight, and so on. With argument assignment number two, you can have up to ten sets of arguments (rises and falls). Each set can contain up to three arguments (I, J, and K).

As you can see from figure 6.9, only I and J (K is not required in this example) are required to describe each rise or fall. I is the incremental angle of each rise or fall, and J is the ending radius of the rise or fall. Also notice that the starting radius is specified with letter address A. The
thickness of the workpiece (plus clearance) is specified with B. The incremental angle used for calculation is C.

Here is an example program that calls the cam custom macro program and machine an entire cam.

O2001 (Main program for cam)
N005 G54 (Set program zero)
N010 G65 P1002 A1.75 B-.03 C1.0 I12.0 J1.75 I35.0 J1.625 I175.0 J1.1 I25.0 J1.75 I113. J1.75 (Mill entire cam)
N015 M30 (End of program)

How will the Custom Macro know when to stop? Whenever argument assignment number two is used, you must have your custom macro program keep track of when the last set of arguments (I, J, & K) is passed. This can be done by monitoring for vacancy (#0). From the chart for argument assignment number two, notice that I(1) is place in local variable #4. I(2) is placed in #7. I(3) is placed in #10. And so on. Note that each local variable for I is precisely three local variable numbers from the previous one. Though not simple to do, we can monitor the current value of each I (#4, #7, #10, #13 etc.) to determine the first time an I is vacant. The custom macro program can assume that when an I is vacant, there are no more rises or falls on the cam.

To do this, you actually create a loop to determine the number of rises and falls. Here is a portion of the loop.

O2001 (Program number)
#102 = 0 (Step value for I test)
N1 IF [ #4 + #102] EQ #0] GOTO 99 (If current I is vacant, exit loop)
.
.
.
#102 = #102 + 3 (Step I tester by three)
GOTO 1 (Go back and test if finished)
N99 M99 (End of custom macro program)

With this loop, we can continue to test for the final occurrence of the set of arguments being passed. With this understood, look at this very powerful parametric program that contains a loop within a loop to machine the entire circular cam.

O1002 (Custom Macro for cam)
#101 = 0 (Initialize current angle to zero)
#102 = 0 (Initialize stepper for last I test)
#104 = #1 (Initialize starting radius)
G90 S600 M03 (Start spindle)
G00 X0 Y-[#1+.5] (Move to starting Y position at bottom)
G43 H01 Z.1 (Instate tool length comp, move into Z position)
G01 Z#2 F25. (Fast feed to bottom in Z)
Y-[#1] F5. (Come flush to surface in Y)
N10 IF [ #4 + #102] EQ #0] GOTO 99 (Start loop for ending I)
#103 = #101 + #4 + #102 (Calculate ending angle for this rise or fall)
#105 = [#104 - #5 + #102] / #4 + #102 * #3 (Calculate radius change per move)
IF [[#5 + #102] EQ #104] GOTO 15 (If start and end radius of rise or fall are equal, make simple circular move)
N1 #101 = #101 + #3  (Step current angle)
#104 = #104 - #105 (Step current radius)
IF [#101 GT #103] GOTO 50 (If finished with current rise or fall exit loop)
#110 = [SIN[#101] * #104] * [-1.] (Calculate current X position)
#111 = [COS[#101] * #104] * [-1.] (Calculate current Y position)
G01 X#110 Y#111 (Make current move)
GOTO 1 (Go back to beginning of loop)
N50 #104 = #5 + #102 (Calculate next beginning radius)
#102 = #102 + 3 (Step the current I and J)
#101 = #103 (Reset current angle)
GOTO 10 (Go to next rise or fall)
N15 #101 = #103 (Begin to make simple radius)
#110 = [SIN[#101] * #104] * [-1.] (Calculate ending X)
#111 = [COS[#101] * #104] * [-1.] (Calculate ending Y)
G02 X#110 Y#111 R#104 (Make circular move to form dwell)
GOTO 50 (Go to next rise or fall)
N99 G00 Z.5 (Finished, get out in Z)
G91 G28 X0 Y0 Z0 (Return to zero return position)
M99 (End of Custom Macro)

While this Custom Macro will require study, it truly stresses the use of argument assignment number two. It is also one of the more powerful parametric programs you will. With one loop inside another, notice how few commands are required. Depending on the value of C (incremental angle between each movement), this custom macro program could generate many, many commands. If, for example, C is set to 0.001, this cam macro could generate as many 360,000 motions (assuming there are no dwells on the cam)!