OpenCNC_Macro Development Manual.

1 **Preface**

In order to increase the flexibility of the controller application, Syntec controller provides MACRO program editing functions. When the processing program is declared in the MACRO format, specific mathematical functions are applicable like other programming languages. In this way, in addition to the original movement and compensation command functions, logical judgment and mathematical calculation functions are also included.

2 **File Format**

The first line of the program should be declared as the title line with "**%**" and add the keyword "**@MACRO**". Otherwise, the file will be regarded as a normal ISO format file, and user is not able to to use the full functionality of the MACRO. In addition, each line of the program content must be followed by a **semicolon** "**;**", but there are exceptions to some of the syntax, refer to the syntax description.

ISO format	MACRO format
Do not support MACRO syntax. (While there are syntax can not be complied correctly, there may be alarm "COM-003."	Support full functionality of MACRO syntax.
With or without semicolon at the end of line is acceptable.	1. Except the special syntax, every line should end with semicolon. 2. If the line does not end with semicolon, CNC combine it together with the next line when checking the syntax. If there is no alarm, the NC program run normally; otherwise, there comes the alarm "COM-008," which is "line does not end with semicolon." Example: _(see table below)_
While using "(...)", the ... is regarded as comment.	While using "(*...*)", the ... is regarded as comment.
Programming according to Pr3201.	Programming according to Lathe C-Type.

Example table:

	Original	what CNC read	Description
Alarm	%@MACRO #1:=SIN(100) G01 Y100.; M99;	%@MACRO #1:=SIN(100) G01 Y100.; M99;	Due to wrong syntax after combined, there is alarm, "COM-008."
No alarm	%@MACRO G01 X100. G01 Y100.; M99;	%@MACRO G01 X100. G01 Y100.; M99;	Due to correct syntax even after combined, there is no alarm. The spindle moves to X100. Y100.

Note:

1. It is recommended that multi-path style(including $1 and $2) might not be used in NC Program which is called as sub-program or a macro.
    Should it be inevitable, the size of the program **must be** less than **60KB(60000bytes)**. Otherwise, there comes the alarm COR-203 Illegal NC file format

2. Not support the sub-program, which is in MACRO format, using the multi-path style (including $1, $2).

3. If the size of file is larger than 60KB(60000bytes), it would not support the syntax like IF, CASE, REPEAT, FOR, WHILE, which are longer than one line. If these syntax are used, there will be alarms.(COR-204 File size too large)

4. Only ASCII characters are acceptable in NC files. Using non-ASCII characters is thus unacceptable and will trigger COM-027 Invalid character.
    Note: Listed below are the special cases in which non-ASCII characters are considered acceptable:
    a. Comment.
    b. Arguments of MACRO functions that are of string type.

5. Support CR ("\r")、 LF ("\n")、 CR LF ("\r\n") line feed.

3 **Block Format**

The writing format of the block (one line) is instructed as follows:

/	N	G	X	Y	Z	A	B	C	I	J	K	F	S	T	D	M

/	Block selective jump function. C41 in PLC must coordinate as well; does not support "function syntax" and "variable calculation."
N	The block sequence No., which must be written at the head of a block, and the MACRO command or variable designation cannot be written in the same line.
G	Function specification code, need to be written after N code
X	The X axis movement command, or the expansion of the G code, must be written after the G code.
Y	The Y axis movement command, or the expansion of the G code, must be written after the G code.
Z	The Z axis movement command, or the expansion of the G code, must be written after the G code.
A	The A axis movement command, or the expansion of the G code, must be written after the G code.
B	The B axis movement command, or the expansion of the G code, must be written after the G code.
C	The C axis movement command, or the expansion of the G code, must be written after the G code.
I	The radius command in the X direction or the argument of the expansion G code, must be written after the G code.
J	The radius command in the Y direction or the argument of the expansion G code, must be written after the G code.
K	The radius command in the Z direction or the argument of the expansion G code, must be written after the G code.
F	Block feed rate, or the argument of expansion G code.

S	Spindle rotation speed, or the argument of expansion G code.
T	Tool selection function, or the argument of expansion G code.
D	Tool compensation function, or the argument of expansion G code.
M	Auxiliary, or the argument of expansion G code.

Core interpretation processing order (1. first ~10. last):

1. Modal G code(G15、G17、G70…), expansion G code macro(G73、G84…)
2. M code macro, T code macro
3. S code
4. F code
5. H code
6. D code
7. T code
8. M code
9. B code
10. Interpolated G code（G0、G1…）, functional G code（G4、G51、G68…）

Note:

1. The format not mentioned above is introduced by the relevant G code in the form of an argument.
2. Generally, the "GETARG" function is used in the sub-program to read the argument. The rules, instructing the form of argument in the main program (parent program), are as follows:
    a. For parameter D, E, H, I, J, K, L, M, P, Q, R, T, argument should be attached directly, such as "G101 X30.Y40. D50. ;". If parameter is followed by a symbol first then argument, such as "G101 X30. Y40. D1=50. ;", there is an alarm.
    b. For parameter A, B, C, F, S, U, V, W, X, Y, Z, in addition to attaching argument directly, user can also attach a number before the argument, for example "G101 X30. Y40 . Z1=50. ;"
    c. Right after the above action instructions, only the value or the variable stored as numeric can be used. Otherwise, the system error may be caused by the coding limitation from program interpretation. This misuse is not under the range of protection.

4 **Operator**

Operator	Sign	Operating order
Brackets	( ) [ ]	1
Function assignment	Identifier	2
Negative	-	3
Complement	NOT	3
Multiplication sign	*	4
Divisor	/	4
Modulus (remainder)	MOD	4
Plus	+	5
Minus	-	5
Comparison	<,>,<=,>=	6
Equal	=	7
Not equal	<>	8
Boolean "and"	&,AND	9
Boolean "Mutually exclusive"	XOR	10
Boolean "or"	OR	11

Note 1:

When using the "/" component (division), be aware that if the numerator and denominator are integers, the result is still an integer. The difference between an integer and a non-integer result is whether user adds the decimal point or not.

example:

- The numerator is a non-integer: 1. / 2 = 0.5
- The denominator is a non-integer: 1 / 2.0 = 0.5
- The numerator and denominator are integers: 1/2 = 0

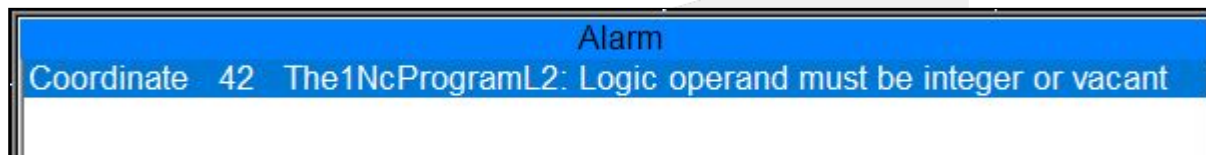- The number in the bracket is an integer: (1/2)*1.0 = 0

Note 2:

The MOD operator (modulo) is only applicable to the numeric type "Long". If the numeric type is "Double", the following alarm message shows up.

example:

%@MACRO

 @1:= 4. MOD 3;

 M99;

5 **Language Instructions**

5.1 Variable Designation

Variable Designation	
Syntax	<**variable**> := <**description**>;
Explanation	Designate variable content

Example 1: **Direct setting**	@1 := 123;
	#1 := 456;
	#10 := "12"; // The **local variable** #10 content is 12
	@10 := "12"; // **public variable** @10 content is 12849

Example 2: **Indirect setting**	#1:= 123;
	@[#1] := 567; // @123=567
	@[#1+7]:=890; // @130=890

Remarks	1. The "12" in the first example is a string, indicating that the string is stored in the variable. When the public variable is stored, the controller will translate the string into ASCII. For the local variable, the translation will not be executed. 2. To correctly read the contents of the string stored in the public variable, use the **SCANTEXT function**. 3. In the example 2, please notice that it's the "square bracket"

5.2 GOTO

GOTO	
Syntax	**GOTO n**;
Explanation	The use of GOTO should be paired up with block sequence code(n). CNC would jump to the specified N-number and execute from that line. If there are two same N-numbers in the program, the first N-line number in the program will take precedence than the second one.

Examples	%@MACRO
	#1 := 1;
	#2 := 10;
	IF(#1 = 1) THEN
	GOTO #2;
	END_IF;
	IF(#1 = 2) THEN
	GOTO 100;
	END_IF;
	N10;
	G01 G90 X50. Y0. F1000;
	M30;
	N100;
	G01 G90 X0. Y50. F1000;
	M30;
Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

5.3 CASE

CASE	
Syntax	**CASE** \<condition variable\> **OF**
	\<**variable**\>:
	\<statement List\>
	\<**variable**\>, \<**variable**\>:
	\<statement list\>
	\<**variable**\>, \<**variable**\>, \<**variable**\>:
	\<statement list\>
	ELSE
	\<statement list\>
	END_CASE;

Explanation	Multi-conditional judgment. CNC, according to the condition variable, executes different program blocks. Please note that the "**variable**" content must be an **integer** which is greater than or equal to zero.

Examples	%@MACRO
	#1 := 1;
	G01 G90 X0. Y0. F1000;
	CASE #1 OF
	1:
	    X(1.0*#1) Y(1.0*#1);
	2:
	    X(2.0*#1) Y(2.0*#1);
	3, 4, 5:
	    X(3.0*#1) Y(3.0*#1);
	ELSE
	    X(4.0*#1) Y(4.0*#1);
	END_CASE;
	M30;

5.4 IF

IF	

Syntax	**IF** <condition> **THEN**
	<statement list>
	ELSEIF <condition> **THEN**
	 <statement list>
	ELSE
	<statement list>
	END_IF;

Explanation	IF condition judgment

Examples	%@MACRO
	#1 := 3.0;
	G01 G90 X0. Y0. F1000;
	IF #1 = 1 THEN
	X(1.0*#1) Y(1.0*#1);
	ELSEIF #1 = 2 THEN
	X(2.0*#1) Y(2.0*#1);
	ELSEIF #1 = 3 THEN
	X(3.0*#1) Y(3.0*#1);
	ELSE
	X(4.0*#1) Y(4.0*#1);
	END_IF;
	M30;

5.5　Repeat

REPEAT	
Syntax	**REPEAT**
	<statement List>
	UNTIL <condition>
	END_REPEAT;
Explanation	REPEAT loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	REPEAT
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	UNTIL (#14 > #12) OR (#15 > #13) END_REPEAT;
	M30;
Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash. It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

5.6 While

WHILE	
Syntax	**WHILE** <condition> **DO** <statement list> **END_WHILE;**
Explanation	WHILE loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	WHILE (#14 <= #12) AND (#15 <= #13) DO
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_WHILE;
	M30;
Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

5.7 For

FOR	

Syntax	**FOR** <variable 1> := <Description 1> **TO** <Description 2> **BY** <Description 3> **DO**
	<statement list>
	END_FOR;
	Variable 1: The variable that controls the number of loops
	Description 1: The start number of the loop count, which can be a numerical value or an arithmetic expression
	Description 2: The terminated number of the loop count, which can be a numerical value or an arithmetic expression.
	Description 3: The added number to the current loop count after each loop, which can be a numerical value or an arithmetic expression.
	Statement list: execution in each loop
Explanation	FOR loop control
Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	FOR #6 := 0 TO 3 BY 1.0 DO
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15); G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_FOR;
	M30;

Remarks	1. When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	2. It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.
	3. Do NOT use the command whitch will jump out and in FOR loop (e.g: Complex Canned Cycle (G72-G78), using GOTO jump out loop and jump in again), 、M98 H_, it will cause the incorrect added number(<Description 3>). example:

```
// FOR #10 will add 5 in each loop cycle
% @MACRO   FOR # 10 := 1  TO  100  BY  1  DO
 GOTO  12 ;
 N13;
END_FOR;
M30;

N12;
M00;
@1 := @1 + 5 ;
GOTO  13 ;
M99;
```

5.8 EXIT

EXIT	
Syntax	EXIT
Explanation	Interrupt loop, jump out of loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	#16 := 1.0;
	G01 G90 X#12 Y#13 F1000;
	FOR #6 := 0 TO 3 BY 1.0 DO
	IF((#14 = 4) & (#16 = 1)) THEN
	EXIT;
	END_IF;
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_FOR;
	M30;

5.9 Program Annotation

Program Annotation	
Syntax	(* <statement list> *) // <statement list>
Ex	Program Annotation(comment)

Example 1
Single line annotation
	%@MACRO
	G00 G90 X0. Y0.; // homing
	M30;

Example 2	%@MACRO
Block annotation	(*
	This block is the annotation area
	Regardless of the content, it does not affect program execution.
	*)
	G00 G90 X0. Y0.;
	G00 G90 X10. Y0.;
	G00 G90 X10. Y10.;
	G00 G90 X0. Y10.;
	G00 G90 X0. Y0.;
	M30;
Remark	If a text that is an annotation is added to the statement list, system error may occur due to the limitation while interpreting the code. This misuse is not under the protection of the controller.

5.10 Area of Execution Program

Area of Execution Program	
Syntax	%
	Execution Program
	%
Explanation	1. While there is "%...%" in the program, the execution program between two % will be executed by CNC. For those program prior to the first % and after the second % will not be executed.
Example 1	G91 G00 X10.
	%
	G91 G00 Y10.
	%
	G91 G00 Z10.
	M30
	//After "Cycle Start", only Y coordinate move to "10.".

Example 2	G91 G00 X10. % G91 G00 Y10. G91 G00 Z10. M30 //After "Cycle Start", Y and Z coordinate move to "10." and X stay unmoved.

6 **MACRO Read/Process Flow**

Icon	Explanation

☐	The following explains the movements of each line of the parent program (main program): (the program in the left box)
	
	• N1: Set the coordinate system to G54 and move in absolute mode G90
	• N2: Call macro G0201 and get the content in argument X1 with GETARG function
	    • #1：=GETARG(X1)
	    After entering G0201, store the argument X1 into the local variable #1
	    • #10：=#1004
	    Back up state G90/G91 with #10
	    • G91 G00 Y#1
	    Y coordinate moves in increment of 10mm by G00
	    • G#10
	    Restore state G90/G91
	    • M99
	    Return to parent program
	• N3: Since the last interpolation mode before leaving G0201 is G00, in this block, the X coordinate still moves by G00
	• N4: Call the macro G0202 and the value of argument X will be stored in #24
	    • #1：=STD(#24,#1600)
	    After entering G0202, store the argument X into the local variable #1
	    • #10：=#1004
	    Back up state G90/G91 with #10
	    • G91 G00 Y#1
	    Y axis moves in increment of -10mm by G00
	    • "G#10"
	    Restore state G90/G91
	    • " #1000：=202"
	    Set the interpolation mode to 202
	    • M99
	    Return to the parent program
	• N5: Since the interpolation mode is stored as 202 before leaving G0202, the system will call G0202 again when this block is executed.
	• N6: End of program

7 **MACRO Writing Note**

- It is recommended to use local variables (Local Variables, #1 ~ #400) in one MACRO and to use global variable when user need to pass value between MACROs (Global Variables, @1 ~ @165535).
- When executing MACRO, the user's data is passed by arguments (A_, B_, …, Z_, X1=, Y1=, …), and the arguments are connected to the local variables. The following table relates the arguments and the local variables.
- For expansion argument address, such as X1, use the function "GETARG" to read the values.

Argument	Corresponded Variable	Argument	Corresponded Variable	Argument	Corresponded Variable
A	#1	J	#5	T	#20
B	#2	K	#6	U	#21
C	#3	L	#12	V	#22
D	#7	M	#13	W	#23
E	#8	P	#16	X	#24
F	#9	Q	#17	Y	#25
H	#11	R	#18	Z	#26
I	#4	S	#19	X1	GETARG(X1)

- Modal Variables (#2001 ~ #2100, #3001 ~ #3100) will return to the VACANT state when the system is reset, so it can be applied to the timing of data exchange between multiple MACROs to save the use of variable resources.
- If a default initial value is required for MACRO, Customer Parameter is recommneded(#4001 ~ #4100, #5001 ~#5100).
- When the MACRO sub-routine (sub-program) is executed, if the mode G code is changed (G91/G90, G40/G41/ G42, …, etc.), please back up the current state, and restore the original mode G state before leaving the MACRO.
- If user want to keep the current MACRO interpolation mode (#1000) after leaving MACRO, it is recommended to designate #1000 as the MACRO number before leaving MACRO. As long as there is single block of the axial displacement, the system will automatically call this MACRO without specifying it again.
  - The interpolation mode will be automatically rewritten when G00/G01/G02/G03/G31/G33 show up or #1000 change.
- For length or angle arguments, use the STD function to normalize the unit before operation to match the usage habits of machine tool.
- Change to the setting of coordinate system is strictly forbidden, such as G92/G54/G52 which are relevant to coordinate system. Otherwise, the simulation would be useless.
- When machining, the core will pre-interpret the MACRO content, so the MACRO progress is ahead of the practical G/M code. If the variable specification or the data reading needs to synchronize with the G/M code, please add WAIT function before the variable specification or the data reading to ensure the movement is

correct.

Note: The WAIT function is used to prevent errors occur from the premature data reading or writing, which is done by the pre-interpretation process, before the associated motion commands have been done by the controller. Thus, the WAIT function only guarantees that the pre-interpretation process won't take place until the previous G/M codes have already been done (M98, M99, and M198 are the exceptions), it is ineffective against other commands.

• The MACRO program must be added with "M99;" to return to the main program (parent program).

• Develop good habits, add more comments to the program, this will increase program readability, and help subsequent maintenance and troubleshooting.

7.1 Registered G Code MACRO

• Developers could, according to the needs, add G code macros other than the standard G code, and could also customize the standard G code.

• Use the [Pr3701~Pr3710 *G code Macro call registry] setting to register the standard G code user want to customize. When the corresponding G code is executed in the program, the customized G code macro will execute instead of the standard G code.

• The following table introduces the setting value of [Pr3701~Pr3710 *G code Macro call registry] and the customizable standard G code.

Pr3701	Standard G Code	File Name of G Code Macro
0	none	none
-1	G00	G0000
1	G01	G0001
2	G02	G0002
3	G03	G0003
4	G53	G0053
5	G40	G0040
6	G41	G0041
7	G42	G0042
8	G43	G0043
9	G44	G0044

| 10 | G49 | G0049 |
| 11 | G04 | G0004 |

- The following are the operating specifications for the G code macro
    - Macro feature is treated as G code macro feature
    - All G codes in the registered G code macro will be executed as the standard G codes.
    - Inheritance property of interpolation G codes such as G00, G01, G02, G03 still persists after these codes are registered as customized G codes.
        - Example:
          G00 X100.
          Y100.
          Where Y100. will also execute the G00 macro, and can read the occupied Y argument.
    - If the user changes the interpolation mode in the customized G code macro, the interpolation mode (#1000) should be restored before leaving the customized G code macro.
        - For example, the user sets G00 as the registered G code macro.
          In the macro G0000, if the user changes the interpolation mode to G01, the user needs to restore the interpolation mode to G00, #1000 should be 0 before leaving G0000, in order to avoid the state disorder.
    - The login G code macro will not work when it encounters the following instructions.
        - Lathe G7.1
        - Lathe G12.1
        - Lathe, A
        - Lathe, R
        - Lathe, C
        - Lathe All machining cycle instructions
        - Mill All machining cycle instructions
        - T code macro

- Version revision

Version	Revision
~Before	The interpretation order of the registered G codes is same as the G code macro.
10.114.50	- The interpretation order of the registered G codes is same as the standard G code. - Do not support changing the interpolation mode to 900000 in G code macro(#1000 := 900000).
10.116.16A、10.116.17	G53 is able to be replaced by customized MACRO (G0053)
10.118.22F、10.118.26	G40, G41, G42 is able to be replaced by customized MACRO (G0040, G0041, G0042)
10.118.45	G43, G44, G49 is able to be replaced by customized MACRO (G0043, G0044, G0049)
10.118.52D, 10.118.54	G04 is able to be replaced by customized MACRO (G0004)

8 **Function List**

Function	Explanation

ABS	Get the absolute value
	
	Example:
	
	``
	#10 := -1.1;
	#1 := ABS(#10); // #1 = 1.1
	#2 := ABS(-1.2); // #2 = 1.2
	``

ACOS	Calculate the acos of a value
	
	Example:
	
	``
	#10 := 1;
	#1 := ACOS(#10); // #1 = 0
	#2 := ACOS(-1); // #2 = 180
	``

ALARM	Call macro alarm
	
	Example:
	
	``
	ALARM(300); // 觸發巨集第300號警報
	ALARM(301, "ALARM 301 Content");
	``
	
	Remark: There is limit of string length in an alarm. For Mandarin, it's 19 words；For English, it's 39 alphabets.

ASIN	Calculate the asin of a value
	
	Example:
	
	``
	#10 := 1;
	#1 := ASIN(#10); // #1 = 90
	#2 := ASIN(-1); // #2 = -90
	``

ATAN	Calculate the atan of a value. The calculation result is between ±90°.
	
	Example:
	
	``
	#10 := 1;
	#1 := ATAN(#10); // #1 = 45
	#2 := ATAN(-1); // #2 = -45
	``

Function	Explanation

ATAN2(Y, X)	Calculate the four-quadrant atan value of Y/X. The calculation result is between ±180°.
	
	Example:
	
	`#10 := 1;`
	`#20 := -1`
	`#1 := ATAN2(#10, #20);  // #1 = 135`
	`#2 := ATAN2(#20, #10);  // #2 = -45`
	`#3 := ATAN2(1, 0);      // #3 = 90`
	
	Notes:
	
	1. Valid version: 10.118.29W, 10.118.40C, 10.118.42
	2. The argument X and Y must be numbers, or the alarm COR-023 【Semantic error】 will be issued.
	3. The argument X and Y can not be zero at the same time, or the alarm COR-004 【Operation domain error】 will be issued.
	
	Example of wrong cases:
	
	`@1 := ATAN2( "1", 1 );` // The first argument is not a number, issue COR-023 alarm.
	`@2 := ATAN2( 0, 0 );` // The arguments are both zero, issue COR-004 alarm.

AXID	Inquire the axis number corresponding to the axis name. If the axis name does not exist, the return value is blank (VACANT, #0)
	
	Example:
	
	Suppose the sixth axis name is Y2 (Pr326=202) and the second axis name is Y (Pr322=200).
	
	`#1 := AXID(Y); // #1 = 2`
	
	`#2 := AXID(Y2); // #2 = 6`

CEIL	Return the smallest integer greater than or equal to a certain value
	
	Example:
	
	`#10 := 1.4;`
	
	`#1 := CEIL(#10); // #1 = 2`
	
	`#2 := CEIL(1.5); // #2 = 2`

CLOSE	Close the file opened by the OPEN function, and the file will be automatically closed after the program ends. The PRINT function will fail if the file is already closed. Example: `CLOSE(); // close the file`

Function	Explanation

COS	Calculate the cosine of a value
	
	Example:
	
	#10 := 180;
	
	#1 := COS(#10); // #1 = -1
	
	#2 := COS(-180); // #2 = -1

DBLOAD	Read the data of specified index from the currently loaded XML. For related applications, please refer to the appendix.
	
	Example:
	
	DBOPEN("FLAT\\TAB01");
	
	// Load FLAT\\TAB01 data file
	
	DBLOAD(0);
	
	// read the 0th cycle
	
	DBLOAD(1);
	
	// read the 1st cycle
	
	Note: The file path to the XMLDB may be influenced by customized Action (CUSTOMFILE_CYCLE1~5), please refer to CE人机客制应用文件.

Function	Explanation

DBSAVE	Save the data of specified index from the currently loaded XML. For related applications, please refer to the appendix.
	
	EX：
	
	``
	1 DBOPEN("GrinderToolTable.cyc"); //
	    Load "GrinderToolTable.cyc"
	2
	3 DBLOAD(0); //
	    Load the 0-th datum from cyc file
	4 DBSAVE(0); //
	    Save the 0-th datum to cyc file
	``
	
	Notes：
	1. The file path for XMLDB can be changed by Customized Actions (CUSTOMFILE_CYCLE1~5). Fore more info, refer to CE人机客制应用文件-开放使用的Action列表 。
	2. Opening (DBOPEN) and loading (DBLOAD) must be done before saving (DBSAVE). If the user uses DBSAVE without using DBPEN and DBLOAD beforehand, the saving process will NOT be executed.
	3. Even if the Field Value is the same as the default value defined in the Schema, the data will still be written into the CYC data file.
	4. Versions: 10.118.39 and later.

DBOPEN	Load the specified XML. The XML should be in the GNCFILES specified by the user. For related applications, please refer to the appendix.
	
	Example:
	
	DBOPEN("Test.cyc");
	
	// Load GNCFILES\\Test.cyc
	
	Example:
	
	#1 = 51;
	
	DBOPEN("FLAT\\\\AB#1[3]ZZ.cyc");
	
	 // Load FLAT\\\\AB051ZZ.cyc, [3] indicates that the file name is in three valid digits
	
	Note1: The file path to the XMLDB may be influenced by customized action (CUSTOMFILE_CYCLE1~5), please refer to CE人机客制应用文件.
	
	Note2: For the requirement of reloading file, please reset system (or run BGnd Stop in Background Running Objects) before re-execute DBOPEN.

Function	Explanation
DRAWHOLE	Draw a circle based on the tool radius and the color defined by the SETDRAW function at the current position (only valid in the simulation, system will not add a circle in the actual path)
EXP	Calculate the exponential value with natural number as the base Example: #1:=EXP(1); // e^1 = 2.71828 Valid version: 10.116.16

FLOOR	Return the largest integer less than or equal to a certain value
	
	Example:
	
	    #10 := 1.4;
	
	    #1 := FLOOR(#10); // #1 = 1
	
	    #2 := FLOOR(1.5); // #2 = 1

GETARG	Read the arguments passed by the caller
	
	Example:
	
	    Assume that the main program content of O0001 is
	
	    G101 X30. Y40. Z1=40. Z2=50.;
	
	    The G0101 expansion macro program uses GETARG to read the argument content
	
	     #1 := GETARG(X); // Save the X argument 30. to #1
	
	     #2 := GETARG(Z1); // Save the Z1 argument 40., to #2
	
	     #3 := GETARG(W);
	
	     // Since W does not exist, #3 is (VACANT, #0)

Function	Explanation

GETTRAPARG	Read the argument content in Trap Block. Trap Block is the blocks between G66/ G66.1 and G67.
	
	Example:
	
	Assume that the main program content of O0001 is
	
	G66 P100 X100. Y100. // P100 means call sub-program O0100.
	
	G01 X20. // O0100 sub-program uses GETTRAPARG to read the argument content
	
	#1 := GETARG(X);
	
	// Save the X argument 100. to #1
	
	#2 := GETTRAPARG(X);
	
	// Save the X argument in the Trap block, 20. to #2
	
	Please refer to G66/G67：Call Modal Macro.

LN	Calculate the logarithm value with natural number as the base. Example: #2:=LN(100); // ln100 = 4.60517 Note: The argument has to be positive. Otherwise, the alarm shows up. Valid version: 10.116.16

MAX	Get the maximum value of two input values
	
	Example:
	
	#10 := 1.2;
	
	#20 := 4.5;
	
	#1 := MAX(#10, #20); // #1 = 4.5
	
	#2 := MAX(-1.2, -4.5); // #2 = -1.2

MIN	Get the minimum value of two input values
	
	Example:
	
	#10 := 1.2;
	
	#20 := 4.5;
	
	#1 := MIN(#10, #20); // #1 = 1.2
	
	#2 := MIN(-1.2, -4.5); // #2 = -4.5

Function	Explanation

MSG	Customize the message, please refer to the "MACRO Customized Message" for details.
	
	Example:
	
	      MSG(100); // message ID
	
	      MSG("bit lost"); // display message content
	
	      MSG(100, "bit lost"); // hint ID + displaye message content
	
	Remark: There is limit of string length in a message. For Mandarin, it's 19 words；For English, it's 39 alphabets.

Function	Explanation

OPEN ("file name") or
 	
OPEN ("file name", "writing mode")
	Open a text file name of which user specify, which is in the NcFiles folder (Folder path please refer the Pr3219). The PRINT function is valid only after the file is opened.
	
	If the file name is "COM", it means that the port RS232/RS485 is turned on, and its setting is determined by Pr3905.
	
	Example:
	
	      OPEN("COM"); // Open port RS232
	
	      PRINT("\\p"); // Output '%' character
	
	      FOR #1 = 1 TO 5000 DO
	
	          #30 := #1 * 10.;
	
	          PRINT("G01 X#30"); // Output G01 X10.0
	
	      END_FOR;
	
	      PRINT("\\p"); // Output '%' character
	
	      CLOSE(); // Close port
	
	The "writing mode" determines, when the file is opened, whether the original file content is retained or cleared. (valid version: 10.116.36I)
	
	(i) "a": Keep the original text and the text newly-output follows the original one.
	
	Example:
	
	      OPEN("PROBE.NC", "a");
	
	      // Open PROBE.NC and keep the text, and be ready for text output
	
	(ii) "w"/nothing: Clear the original text and output the new text in the file.
	
	Example:
	
	      OPEN("PROBE.NC");
	
	      // Open PROBE.NC and clear the text, and be ready for text output
	
	      OPEN("PROBE.NC", "w");
	
	      // Open PROBE.NC and clear the text, and be ready for text output
	
	(iii) Wrong writing mode: system issues alarm, COR-301 OPEN command format error.
	
	Example:
	
	      OPEN("PROBE.NC", "abc");
	
	      // Wrong writing mode, issues alarm, COR-301, so PROBE.NC is not opened for text output.
	
	(iv) Convert # or @ variable into a string, and the decimal digits is determined by Pr17 (valid version: 10.118.12C)

Function	Explanation
	(v) Convert # or @ variable into a string with [*] in the end, and the decimal digits is determined by this variable.

PARAM	Read the contents of system parameters
	
	Example:
	
	    #1 := PARAM(3204);
	
	    // Read the contents of Pr3204 (PLC scan time)

POP	Taking the data in STACK from top layer to bottom layer in sequence. User must pay attention to the total data in the stack. If there are 5 data, the maximum times to use POP is 5.
	
	Example:
	
	    PUSH(5); // Insert the number 5 into the stack
	
	    #1 := POP(); // Remove the topmost value in the stack (#1 = 5)

POW	Calculate the power of specified base Example: #3:=POW(16,0.5); // 16^0.5 = 4 Note: The base cannot be negative. Otherwise there will be alarm COR-122. Valid version: 10.116.16

Function	Explanation
PRINT	This function is used to output a string, and the variable in the output string will be replaced by the content of it.
	The character "\" is an escape character, and the related special characters are defined as follows:
	"\\": indicates "\" character
	"\@": indicates "@" character
	"\#": indicates "#" character
	"\p": indicates"%" character
	Convert # or @ variable into a string, and the decimal digits is determined by Pr17 (valid version: 10.118.12C)
	Convert # or @ variable into a string with [*] in the end, and the decimal digits is determined by this variable.
	Example:
	Assume that Pr17=2 in metric unit
	@53 = 20 ;
	#3 = 23.1234 ;
	PRINT("G01 X#3 Y@53 Z20.0") ;
	The output is G01 X23.123 Y20 Z20.0 ; // Display to thousandths place
	Example:
	@53 = 20 ;
	#3 = 23.1234 ;
	PRINT("G01 X#3[2] Y@53 Z20.0") ; // #3[2] means display to hundredths place
	The output is G01 X23.12 Y20 Z20.0 ; // Display to hundredths place
PUSH	Stuff data into the STACK, the data PUSH into the controller first will be stacked on the bottom layer, and the last data on the top one.
	Example:
	PUSH(#1); // Put variable #1 into the STACK
RANDOM	Generate a random number
	Example:
	#1 := RANDOM();

Function	Explanation

READDI
(I point number)
READDO
(O point number)
	The value of variable derives from the I/O point number in the parentheses READDI/READDO.
	
	Example:
	
	@52 := READDI(31); // Read the value of I31 and put it in @52
	
	#88 := READDO(11); // Read the value of O11 and put it in #88
	
	G90 G10 L1000 P4000 R READDI(15); // Read the value of I15 and put it in R4000
	
	Notes:
	
	1. Valid version: 10.116.23
	2. The I/O point is read during pre-interpreting, but it is processed when READDI / READDO is executed in order to avoid the error resulting from pre-interpreting I/O point. Because system await until READDI / READDO is executed, machine will decelerate to zero.
	3. The range of I/O point number is 0~511. If the number is out of the range, system issues alarm, COR-138 Read/write command format error at the I/O/ A point.

READABIT
 (A point number)
	The value of variable derives from the A point number in the parentheses of READBIT.
	
	Example:
	
	@52 := READABIT(31); // Read the value of A31 and put it in @52
	
	#88 := READABIT(11); // Read the value of A11 and put it in #88
	
	Notes:
	
	1. Valid version: 10.116.44
	2. The A point is read during pre-interpreting, but it is processed when READBIT is executed in order to avoid the error resulting from pre-interpreting A point. Because system await until READBIT is executed, machine will decelerate to zero.
	3. The range of A point number is 0~511. If the number is out of the range, system issues alarm, COR-138 Read/write command format error at the I/O/ A point.

Function	Explanation
READRREGBIT (Register number, specified Bit)	The value of variable derives from the register number and specified bit in the parenthesis of the READRREGBIT. Example: @52 := READRREGBIT(31,3); // Read the value of the third bit of R31 and put it in @52 Notes: 1. Valid version: 10.116.39 2. The register is read during pre-interpreting, but it is processed when READRREGBIT is executed in order to avoid the error resulting from pre-interpreting Register. Because system await until READBIT is executed, machine will decelerate to zero. 3. If register is less than 0 or greater than 65535, the system issues alarm, COR-135 Read/write command format error for R value. 4. If register is an incorrect character, system issues alarms, COR-5 Program loading failure and COM-8 absent statement ending character ';'. 5. If specified bit is less than 0 or greater than 31, system issues alarm, COR-135 Read/write command format error for R value. 6. If specified bit or both of register and specified bit is incorrect characters, system issues alarm, COR-5 Program loading failure and COM-9 wrong assignment character ':='.

ROUND	Return a rounding value.
	
	Example:
	
	 #10 := 1.4;
	
	 #1 := ROUND(#10); // #1 = 1
	
	 #2 := ROUND(1.5); // #2 = 2

SYNTEC

Function	Explanation

SCANTEXT	This function is used to read the contents of the string stored in global variable.
	
	When the string is stored in global variable, the controller translate it into ASCII first and save. User get wrong string if they output the value directly. To get the correct string, please make good use of this function.
	
	Example:
	
	    %@MACRO
	
	    @1:="12"; // 16 carry HEX=3231, 10 carry DEC=12849
	
	    #1:=SCANTEXT(1);
	
	    OPEN("NC");
	
	    PRINT("@1");
	
	    PRINT("#1");
	
	    CLOSE();
	
	    M30;
	
	    The result is @1 = 12849
	
	    #1 = 12

SETDO
(O point number,
 O point on or off)
	Determine O point number and the state (1: On, 0: Off) with 2 numbers in the parenthesis of SETDO.
	
	Example:
	
	    SETDO(3, 1); // Set O3 on
	
	    SETDO(8, 0); // Set O8 off
	
	Notes:
	
	1. Valid version: 10.116.23
	2. The writing of point O is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETDO should be avoided. For example, O1 is on by SETDO in MACRO, but off in PLC. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. The range of O point number is limited to 0~511. If the range is wrong, the system issues alarm, COR-138 Read/write command format error at the I/O/A point.

Function	Explanation

SETABIT
 	
(point A, point A on or off)
	Determine A point number and the state (1: On, 0: Off) with 2 numbers in the parenthesis of SETDO.
	
	Example:
	
	    SETABIT(3, 1); // Set A3 on
	
	    SETABIT(8, 0); // Set A8 off
	
	Notes:
	
	1. Valid version: 10.116.44
	2. The writing of point A is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETABIT should be avoided. For example, A1 is on by SETABIT in MACRO, but off in PLC. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. The range of A point number is limited to 0~511. If the range is wrong, the system issues alarm, COR-138 Read/write command format error at the I/O/A point.

SETRREGBIT
 	
(Register number, Bit number, on or off)
	Determine Register number, Bit number, and the state (1: On, 0: Off) with the 3 digits in the parenthesis of SETRREGBIT.
	
	Example:
	
	    SETRREGBIT(50,3,1); // Set the third R50 Bit on
	
	    SETRREGBIT(50,4,0); // Set the fourth R50 Bit off
	
	Notes:
	
	1. Valid version: 10.116.39
	2. The writing of Register is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETRREGBIT should be avoided. For example, first Bit of R50 is on by SETRREGBIT in MACRO, but is off in PLC off. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. If Register number is less than 0 or greater than 65535, the system issue alarm, COR-135 Read/write command format error for R value.
	5. If Bit number is less than 0 or greater than 31, system issues alarm, COR-135 Read/write command format error for R value.
	6. If the state is not 0 (off) or 1 (on), system issues alarm, COR-135 Read/write command format error for R value.
	7. If any argument is incorrect character, system issues alarms, COR-5 Program loading failure and COM-3 Syntax error.

Function	Explanation
SETDRAW (path color) or SETDRAW (path color, filled color, tool radius)	Define the drawing style of simulation: 1. Path Color: set the color of the outline, which can be set by BGR code or by color code in "Simu. Setting." 2. Fill color: set the color filled in circle drawn by DRAWHOLE, which can be set by BGR code or by color code in "Simu. Setting." 3. Tool radius: set the radius of circle drawn by DRAWHOLE, G code with tool radius compensation, such as G01, is influenced by this as well. Common BGR codes are as follows: Color Setting:

Color Code	BGR codes	Color Code	BGR codes
0	0	8	8421504
1	8388608	9	16711680
2	32768	10	65280
3	8421376	11	16776960
4	128	12	255
5	8388736	13	16711935
6	32896	14	65535
7	12632256	15	16777215

Note:

- BGR code is the same as RGB code, but with different endianness for B(Blue) and R(Red) value.

e.g.

Red color in RGB code is 0xFF0000. Converting it into BGR code will be 0x0000FF = 255. (as described in color code 12)

- SETDRAW sets path color and filled color at the same time. If user would like to make path color and filled color different, remember to change the path color with SETDRAW after DRAWHOLE is executed.

Function	Explanation
	e.g. %@MACRO #3:=SETDRAW(#1,#2,#18); // #3 records the original path color, #2 defines the filled color, #18 defines the circle radius DRAWHOLE(); SETDRAW(#3); // Change the path color after drawing the circle M99;

SIGN	Return the sign of a value, the negative number is -1, the positive number is 1, and 0 is 0.
	
	Example:
	
	    #10 := 4;
	
	    #1 := SIGN(#10); // #1 = 1
	
	    #2 := SIGN(-4); // #2 = -1
	
	    #3 := SIGN(0); // #3 = 0

SIN	Calculate the sine of a value
	
	Example:
	
	    #10 := 90;
	
	    #1 := SIN(#10); // #1 = 1
	
	    #2 := SIN(-90); // #2 = -1

SLEEP	Temporarily abandon the execution right of this macro loop, generally used in conjunction with the loop (FOR, WHILE..., etc.) to avoid entering the infinite loop, which causes the human-machine to crash. Example: SLEEP();

SQRT	Calculate the square root value of a value
	
	Example:
	
	    #10 := 4;
	
	    #1 := SQRT(#10); // #1 = 2
	
	    #2 := SQRT(9); // #2 = 3

Function	Explanation

STD (argument1, argument 2)	According to Pr17, the value is converted into the input unit (Input Unit, IU) set by the system at that time.
	
	1. The argument 1 is the value the unit of which is about to be changed.
	2. The argument 2 is a standard unit. Generally, it is #1600, and value of #1600 is from Pr17.
	
	Metric Unit:
	
	Example 1:
	
	  When Pr17=2, #1600 corresponds to LIU = 0.001mm
	
	  #9 := 100;
	
	  #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.1mm (100*0.001)
	
	Example 2:
	
	  When Pr17=3, #1600 corresponds to LIU = 0.0001mm
	
	  #9 := 100;
	
	  #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.01mm (100*0.0001)
	
	Imperial:
	
	Example 3:
	
	  When Pr17=2, #1600 corresponds to LIU = 0.0001inch
	
	  #9 := 100;
	
	  #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.01inch (100*0.0001)

STDAX (argument 1, argument 2)	Converts the value to the standard unit of the corresponding axis.
	
	The argument 1 is a variable, and the argument 2 is the name of the corresponding axis.
	
	Example:
	
	  #24 := STDAX(#24,X);
	
	  #3 := STDAX(#3,A);

Function	Explanation

STKTOP	Copy the data in the STACK.
	
	Example:
	
	PUSH(5); //Put the number 5 into the stack
	
	PUSH(6); //Put the number 6 into the stack
	
	PUSH(7); //Put the number 7 into the stack
	
	#1 := STKTOP[0] ; // #1 = 7
	
	#2 := STKTOP[1]; // #2 = 6
	
	#3 := STKTOP[2]; // #3 = 5

SYSVAR (Path identification code, system variable code)	Read the system variable in specific Path. Path identification code: 1 is the first Path, 2 is the second Path, and so on. System variable code: system variable number will be read Example: #1 := SYSVAR(1, 1000); // Read the system variable #1000 in the first Path (interpolation mode)

TAN	Calculate the tangent of a value.
	
	Example:
	
	#10 := 45;
	
	#1 := TAN(#10); // #1 = 1
	
	#2 := TAN(-45); // #2 = -1

WAIT	The system stops pre-interpreting until the instruction before WAIT is finished.
	
	Example:
	
	%@MACRO
	
	@50 := 1; // @50 equals to 1
	
	G90 G01 X100. F1000; // Assume to system is Reset at this time
	
	WAIT();
	
	@50 := 0; // @50 equals to 0
	
	M30;
	
	Assume that the system is reset when G01 is in execution. Since the block before WAIT is not finished, @50 equals to 1 after Reset.

Function	Explanation

CHKMN ("machinery code")	Check machinery code. 1: consistent, 0: does not match
	
	Example:
	
	    %@MACRO
	
	    #51 := CHKMN("5566"); //The value of #51 is the checking result .
	
	    IF #51=0 THEN
	
	        ALARM(501, "The manufacturer code is invalid."); //If machinery code does not match, system issues an alarm
	
	    END_IF;
	
	Target version: 10.116.6A

CHKSN ("Serial No.")	Check Serial Number. 1: consistent, 0: does not match
	
	Example:
	
	    %@MACRO
	
	    #52 := CHKSN("M9A0001"); //The value of #52 is the checking result
	
	    IF #52=0 THEN
	
	        ALARM(502, "The serial number is invalid."); //If the serial number does not match, system issues the alarm
	
	    END_IF;
	
	Target version: 10.116.6A

CHKMT ("Machine Type")	Check machine type. 1: consistent, 0: does not match
	
	Example:
	
	    %@MACRO
	
	    #53 := CHKMT("MILL"); //The value of #53 is the check result
	
	    IF #53=0 THEN
	
	        ALARM(503, "The machine type is invalid."); //If machine type does not match, system issues the alarm
	
	    END_IF ;
	
	Target version: 10.116.6A

Function	Explanation

CHKMI ("Model")	Check the controller model, 1: consistent, 0: does not match
	
	For SUPER series, please input 'S'. For other models, please input value according to the actual model. For example, 10B =>10B, 11A, => 11A.
	
	Example:
	
	    %@MACRO
	
	    #54 := CHKMI("S"); //#54 is the checking result
	
	    IF #54=0 THEN
	
	        ALARM(504, "The hardware type is invalid."); //If the model does not match, system issues the alarm
	
	    END_IF;
	
	Target version: 10.116.6A

Function	Explanation
CHKINF(category number, "code")	Check if the code corresponds to the category number. 1: consistent, 0: does not match The range of category numbers is 1~5, each corresponding code is: 1. Machinery code 2. Serial No. 3. Machine Type 4. Model 5. Industrial machine ID For SUPER series, please input 's'. For other models, please input value according to the actual model. For example, 10B =>10B, 11A, => 11A. Example:

Inside the Explanation cell, the example code:

```
%@MACRO
#51 := CHKINF(1, "5566"); // #51 is the checking result
IF #51=0 THEN // If the machineny code does not match, system issues an alarm
ALARM(501, "The manufacturer code is invalid.");
END_IF;

#52 := CHKINF(2, "M9A0001"); // #52 is the checking result
IF #52=0 THEN // If the serial No. does not match, system issues the alarm
ALARM(502, "The serial number is invalid.");
END_IF;

#53 := CHKINF(3, "MILL"); // #53 is the checking[ result
IF #53=0 THEN // If the machine type does not match, system issue the alarm
ALARM(503, "The machine type is invalid.");
END_IF;

        #54 := CHKINF(4, "S"); // #54 is the checking result
        IF #54=0 THEN // If the model does not match, system issues the alarm
        ALARM(504, "The hardware type is invalid.");
        END_IF;

#55 := CHKINF(5, "10"); // #55 is the checking result
IF #55=0 THEN // If the Industrial machine ID does not match, system issues the alarm
ALARM(505, "The industrial machine ID is invalid.");
END_IF;
```

If argument is incorrect, or the category number is out of the range, system issues the alarm, COR-353 【Invalid argument of CHKINF】

Example:

Function	Explanation

	%@MACRO
	#51 := CHKINF(60, "Mill"); // category number is out of range
	#51 := CHKINF("2", "Mill"); // argument 1 is incorrect
	#53 := CHKINF(5, 12345); // argument 2 is incorrect
	
	Target version: 10.118.22M、10.118.28B、10.118.30

STR2INT("string")	Convert a numeric string into an integer
	
	Example 1:
	
	%@MACRO
	
	@1:="5555";
	
	#1:= SCANTEXT(1); // #1 = String 5555
	
	#2:= STR2INT("#1"); // #2 = 5555
	
	Example 2:
	
	%@MACRO
	
	#1:=STR2INT("100"); // #1 = 100
	
	Example 3:
	
	%@MACRO
	
	#1:=STR2INT("123.456"); // #1 = 123
	
	Note: As long as there is character or alphabet in the string, STR2INT is not able to work.

SYNTEC

Function	Explanation

SYSDATA(system data number)	Read the system data number.
	
	EX：
	
	// if want to get system data D336、D77
	WAIT(); // stops pre-interpreting for getting newest value
	#1 := SYSDATA(336); // Axis board data exchange time(D336)
	#2 := SYSDATA(77); // Free hardware memory(D77)
	OPEN("DbgData.txt", "a"); // Open the file which name is "DbgData.txt"
	PRINT("#1 #2"); // print data to file
	CLOSE(); // close the file
	
	Note:
	
	1. Valid version: 10.118.23U, 10.118.28H, 10.118.33.
	2. For getting newest value, it is recommended to use WAIT() function blocking the pre-interpretation before using SYSDATA().
	3. The type of argument must be integer. If the type of argument is incorrect, it will cause function to operate abnormally.
	4. if the argument number is out of range of system data number, system will issues the alarm COR-016 【Illegal variable access】.
	
	EX：
	
	SYSDATA("77"); // the type of argument is string, issues the alarm COR-023
	SYSDATA(77.0); // the type of argument must be integer , issues the alarm COR-023
	SYSDATA(D77); // the argument is not a number, it cause syntax error, issue the allarm COM-008
	SYSDATA(10000); // the argument number is out of range of system data number, issues the alarm COR-016

Function	Explanation

DRVDATA(station_number, status_variables_No(Dec))
 	
DRVDATA(station_number, "status_variables_No(Hex)")
	Read Syntec drive status variables.
	
	status variables No. has two input format.
	
	`Decimal:Convert variable numbers to decimal representation.`
	
	`Hexadecimal:use format "xxxh", x=0~F, fill the status variables number( 3 digit ) and end with 'h' or 'H'.`
	
	e.g：
	
	    // assume want to get
	    // speed command(Pn-D26) of first axis (station_number=1000)
	    // Enc Internal Temperature(Pn-D61) of first spindle
	    (station_number=1003)
	    WAIT(); // To get newest value, block Pre-Interpretation
	    #1 := DRVDATA(1000, 3366); // speed command of first axis (Pn-D26，D26 convert to decimal format is 3366)
	    #2 := DRVDATA(1003, "D61h"); // Enc Internal Temperature of first spindle(Pn-D61，D61 convert to hexadecimal format is "D61h")
	    OPEN("DbgData.txt", "a"); // open file DbgData.txt
	    PRINT("Pn-D26: #1, Pn-D26: #2"); // print value
	    CLOSE(); // close file
	
	The file DbgData.txt content may be below.
	
	Pn-D26: 150, Pn-D26: 430
	
	Note:
	
	1. Valid version: 10.118.23U, 10.118.28.I, 10.118.34.
	2. For getting newest value, it is recommended to use WAIT() function blocking the pre-interpretation before using DRVDATA().
	3. The execution time of each function is 0.1~0.2s.
	4. First argument must be integer, system will issues the alarm COR-023 【**Semantic error**】 .
	5. If second argument is string type, must be hexadecimal format ("xxxh", x=0~F), system will issues the alarm COR-023 【**Semantic error**】 .
	6. second argument must be string or integer value, system will issues the alarm COM-003 【**Syntax error**】 .
	7. If either the drive or the controller does not support the specified status variable, the system will issue the alarm COR-016 【**Illegal variable access**】 .
	8. Visit the "Controllor Axis Info." page to check status variable accessibility. Only those shown are accessibile.
	9. If the controller supports specified status variable and the drive does not support, 0 will be returned.
	10. If no drive corresponding to station number or using non Syntec M3 drive, return VACANT.

Function	Explanation

	11. Since it will take some time to read the status variable after system be powered on, it is recommended to call this function one minute after powering on, or it might cause system issue the alarm COR-016 【**Illegal variable access**】 due to the variable are not read finish.
	
	Example of wrong case：
	
	// alarm COM-3
	DRVDATA(1003, D61h); // second argument must be string or integer value
	
	// alarm COR-023
	DRVDATA("1003", 3425); // first argument must be integer
	DRVDATA(1003, "G21h"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	DRVDATA(1003, "3425"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	DRVDATA(1003, "0D61h"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	
	// alarm COR-016
	DRVDATA(1003, "DFFh"); // drive is not suupot this status No.
	
	// return VACANT
	DRVDATA(9999, "D61h"); // no drive corresponding to station number
	DRVDATA(9999, "DFFh"); // if no drive corresponding to station number, will not check the status variables No.

9 **Call sub-Program**

9.1 Calling Methods

Syntax	Explanation	Calling Type	Local Variable	Example
M98 P_ H_ L_	Call sub-program P_Sub-program Name H_Starting of block sequence No. L_Repeated Counts	Sub-program	Inherit the local variables #1~#400 from main/parent program	M98 P10 L2; Explanation: Call O0010 twice
M198 P_ H_ L_ (If M198 is not logged in Pr3601~)	Call sub-program P_Sub-program Name H_Starting of block sequence No. L_Repeated Counts	Sub-program	Inherit the local variables #1~#400 from main/parent program	M198 P10 L2; Explanation: Call O0010 twice
G65 P_L_	Call Single Macro P_Subroutine Name L_Repeated Counts	Macro	Create new local variables #1~#400, and #1~#26 records the corresponding argument in the calling block	G65 P10 Lw X10.0 Y10.0 Explanation: Call O0010 twice, and input argument

Syntax	Explanation	Calling Type	Local Variable	Example
G66 P_L_	Use movement instruction to call mode macro P_Subroutine Name L_Repeated Counts	Mode Macro	Create an independent section of #1~#400 each time G66 is called. Local variables in this section will be shared until executing G67. After executing G67, local variables in this section will be retrieved and cleared. Note: The local variables in the section are shared with sub-program called by P argument (G66 P) only. They are different from the local variables in the program where G66/G66.1 is called.	G66 P10 X10.0 Y10.0; **X20.** **Y20.** Explanation: Moving instructions X20. and Y20. call O0010, and input arguments X10.0, Y10.0.
G66.1 P_ L_	Each block calls mode macro P_Sub-program Name L_Repeated Counts	Mode Macro	The same as G66.	G66.1 P10 X10.0 **X20.** **G04 X2.** **M30** Explanation: Each block calls O0010 and input argument X10.0.
G_L_	Call expansion G Code. L_Repeated Counts	Macro	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	G128 L3 X1.0 Explanation: Call G0128 three times.
G_	Call customized G Code (G00, G01, G02, G03, G53, G40, G41, G42) Must login Pr3701~ before using.	Macro	Create a new section of local variables #1~#400 each calling, and restoring local variables in main program when the Macro is finished.	G01A_B_C; Explanation: Call customized G01.

Syntax	Explanation	Calling Type	Local Variable	Example
T_	Call sub-program T0000 to change tool. The T code in sub-program is general T code, which does not call T0000.	If Pr3215=1, then it is sub-program.	Inherit local variables #1~#400 from main program	T3; Explanation: Call T0000.
		If Pr3215=2, then it is macro.	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	
M_	Call M code Macro. The M code in the macro is general M code, which does not call M code macro again. Must login Pr3601~ before using.	Macro	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	M13A_B_C; Explanation: Call M0013 macro.

Notes:

- If L argument above isn't assigned, the default value is 1.
- The life cycle of local vaiables (#1~#400) in above form, please refer to Macro Variable Specification.

Example of Variable Life Cycle:

Local variable is shared if same color

Main Program
Test.NC
G01...
G01...
G65 P10 L2
....

M30

O0010 (First-time)
Local variable is independent from Test.NC

G01...
G03...
M98 P11 H50 L2

O0010 (Second-time)
Local variable is independent from Test.NC

O0011 (First-time)
Share local variable with O0010

G66 P12 l1
G01...
G02...
G67

O0011 (Second-time)
Keep Sharing local variable with O0010

G66 P12 l1
G01...
G02...
G67

O0012 for G01
Local variable is independent from O0011
...

O0012 for G02
Share local variable with G01 executed earlier
...

O0012 for G01
Local variable is independent from O0011
...

O0012 for G02
Share local variable with G01 executed earlier
...

9.2   Return Methods

Syntax	Explanation	Example
M99	Return to main program.	M99
M99 P_	Return to the specified block sequence number in main program P_: specified **block sequence number**.	M99 P100; Return to N100 in the main program.
M99 Q_	Return to the sepcified row number in main program P_: specified **row number**.	M99 Q100; Return to row number, like 100, in the main program.
G67	Cancel G66	G67;

10 <u>**Variable Specification**</u>

- For the explanation of # and @ variable, please refer to Macro Variable Specification.
- Perform four arithmetic operations on variables of different types(Long and Double), it would be forced to convert to Double and then do the calculation. For example

```
@1  :=  1 ;           // assign the variable @1 as type of Long
@2  :=  2.0 ;         // assign the variable @2 as type of Double
@3  :=  @1  +  @2 ;   // the result is "@3 := 3.0" which type is Double

@4  :=  2 ;           // assign the variable @4 as type of Long
@5  :=  100 ;         // assign the variable @5 as type of Long
@6  :=  @4  *  @5 ;   // the result is "@6 := 200" which type is Long
```

11 **MACRO Customized Alarm**

11.1 **MACRO Alarm Trigger Syntax**

%@MACRO

ALARM(xxx);// xxx is the Alarm number

M30;

11.2 **DOS System Alarm Content Editor Explanation**

• File Path:

=> Traditional Chinese: C:\\CNC\\EXE\\APPDATA.RES\\CNCCHI.STR

=> English Version: C:\\CNC\\EXE\\APPDATA.RES\\CNCENG.STR

=> Others: C:\\CNC\\EXE\\APPDATA.RES\\CNCLOC.STR

• Content Format: **24**xxx=”**1;MSG**=**Alarm Content**”, xxx is the Alarm number. Please choose an unused number as the customized alarm number, and please note the identification number is **24**.

• Example:

=>CNCCHI.STR:

24003="1;MSG=**max chordal length of arc should be smaller or equal to 0**"

-> CNCENGSTR:

24003="1;MSG= **max arc length can not be negative**"

11.3 **WinCE System Alarm Content Editor Explanation**

• File Path:

=>Chinese Version: DiskC\\OCRes\\CHT\\String\\AlarmMacro_CHT.Xml

=>English Version: DiskC\\OCRes\\Common\\String\\AlarmMacro_Com.Xml

=>General: DiskC\\OCRes**L**\\String\\AlarmMacro_**L**.Xml.

L is the name of each language.

• File Format: <Message ID="AlarmMsg::**Macro**::ID=xxx" Content="**Alarm Content**" />.

xxx is Alarm number. Please choose an unused number as the customized alarm number.

Please note that the identification letter is **Macro**. Length of string in alarm content is 48 alphabets in English, or 31 characters in Chinese. Redundant string exceeds the alarm window.

• Example:

-> CusMacroAlarmMsg_CHT.Xml：

<Message ID="AlarmMsg::**Macro**::ID=3" Content="**max chordal length of arc should be smaller or equal to 0**" />

-> CusMacroAlarmMsg_Common.Xml：

<Message ID="AlarmMsg::**Macro**::ID=3" Content="**max arc length can not be negative**" />

11.4 **Edit Alarm String Through SI (SyntecIDE)**

MACRO alarm string editor is already integrated with SI, for related manual please refer to Macro Alarm String Editor.

12 MACRO Customized Message (MSG)

12.1 MSG Specification Explanation

- If MACRO alarm occurs, system must be reset to clear the alarm. While MSG is able to be cleared by clicking "ESC". MSG can be used for prompt simply. However, MSG vanishs when the program finishes.
- There is limit of string length of MSG. For Chinese, it's 19 characters; For English, it's 39 alphabets.

12.2 MSG Trigger Syntax

- MSG(100);// MSG ID

- MSG("Missing Drill");// Display MSG content

- MSG("100,Missing Drill");// MSG ID + Display Content

13 **Appendix**

13.1 Macro User Guide

13.1.1 Preface

- The built-in G, T, M code may not satisfy demand from all walks of life, so Syntec Corp. provides "customizing macro" for customer.
  Developer is able to, according to the machine properties, develop macros with special actions, which greatly promotes the machine value.
- Before introduction, the methods of calling other programs in main program are as below:
  - Call sub-program: execute sub-program, while reading or occupying argument is forbidden.
  - Call macro: execute macro, while reading and occupying argument is allowed.
  - For argument definition please refer to Argument Explanation.
- The following sections introduces related specification of macro, and the specification of calling sub-program will be skipped.

13.1.2 Macro Classification

- All Macro has to meet the following conditions:
  - Correct macro syntax
    - Program starts with %@MACRO.
    - Each row(block) ends with " ; " (a semicolon).
  - No file extension.
  - Program ends with M99 to return to main program that calls the macro.
- Basically, macro can be categorized into following types
  - G code macro
  - Non-mode call G code (G65)
  - Mode G code (G66/G66.1)
  - T code macro
  - M code macro

Type	Characteristic	Enable Condition	File Name Specification	File Name Range

G Code Macro	• G code macro developed by developer, which is known as expansion G code macro and in contrast to standard G code.	None	• The beginning of file name must be letter G. • No file extension • The file name of expansion G code macro can be separated into that with 4 digit number or 6 digit number.	• G200~G999 • If the number is out of the range, file is not guaranteed to work normally. • May conflict with standard G code.

4 digit number   6 digit number

• Without decimal digits in G code instruction.
• All zeros starting from the largest digit of number can be omitted.

Instruction	Filename	Explanation
• G200 • G0200	G0200	If there is no decimal part in G code instruction, then the File name = G + four digit number

4 digit number   **6 digit number**

• With decimal digit in G code instruction.
• Three digits starting from the left of number in file name corresponds to the integer part of G code instruction
• Last three digits correspond to the decimal part of G code instruction
• All zeros starting from the largest digit of number can be omitted.

Instruction	Filename	Explanation
G200.1	G200001	If there is decimal part in G code instruction, then the file name = G +six digit number
G200.001		
G200.10	G200010	
G200.010		
G200.100	G200100	

| Non-mode G code (G 65) | • Call assigned program through macro
• Must be the last G code in that row. | None | • The beginning of file name must be letter O.
• No file extension.
• In file name, except O, other characters must be number.
• Omit the O while calling.
    • E.g, for file name O0123, instruction is G65 P123. | • O0000~O09999
• If the number is out of the range, file is not guaranteed to work normally. |

Mode G code (G66 / G66.1)	Type	Effect	None	• The beginning of file name must be letter O. • No file extension. • In file name, except O, other characters must be number. • Omit the O while calling. • E.g, for file name O0123, instruction is G65 P123.	• O0000~ O9999 • If the number is out of the range, file is not guaranteed to work normally.
	G66	Call assigned program through macro when every block with moving instruction is finished.			

Type	Effect
G66.1	Call assigned program through macro when every block is finished.

- Must be the last G code in that row.

T Code Macro	Call T1000 file through macro.	Pr3215 Enable T code call mode must be 2.		• File name can only be T0000, other file names are not allowed. • No file extension.	T0000
		Pr 3 2 1 5	**Type**		
		0	• T code supple men tary cod e • doe s not call T00 00		
		1	• Call T00 00 thro ugh sub-pro gra m. • Doe s not read and occ upy any argu men t		

Pr 3215	Type
2	• Call T0000 through macro. • Read and occupy any argument

M code Macro	Different from M code, M code macro calls corresponding M code macro file through macro.	• M code must be logged in Pr3601~3610 "M code MACRO call registry". • Following M code is standard M code and can't be logged as M code macro.	• The beginning of file name must be letter M. • No file extension. • The file name must be "M+4 digit number". The 4 digit number corresponds to M code macro instruction. • Example	• M0000~ M9999 • If the number is out of the range, file is not guaranteed to work normally.

For the M code Macro row, the second column also contains this table:

M00	M30	M98
M01	M96	M99
M02	M97	

And the fourth column contains this table:

File Name	Pr 3601	Calling Instruction
M0123	123	• M123 • M0123

13.1.3 Macro Operation Process Explanation

- Next, G code macro is taken as example for introduction. Any difference of specification between G code macro and other macros will be pointed out.
- When G code is executed, actually the system is executing the content of G code macro.
- Write "M99" in the last row of G code macro. After M99 is executed, system returns to main program and keep processing.
- Instruction in G code macro may change the system state. For example, G90/G91 in G code macro change the G code mode.
  - In order to avoid influencing the main program or other macros/ sub-programs, usually, the G code mode is backed up first while executing G code macro and restore it before leaving G code macro.
- Example:
  - Execute G200 in main program, then execute G code macro G0200.
  - In G0200, backups G code mode first.
  - X coordinate increase in increment of 10.
  - Restore G code mode before leaving.

Example_Main

```
1   // Main
2   G90;
3   G01 X10. F100.:          // X Axis moving by G01    result X=10.
4   G200;                    // X Axis moving by G200   result X=20.
5   X- 20 .;                  // X Axis moving by G01    result
    X=-20.
6   M30;
```

Example_G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 1000 ;           // Backup #1000, G00/G01/G02/G03/
    G33/G34/G35
4   # 102  := # 1004 :           // Backup #1004, G90/G91
5   G91 G00 X10.:            // X coordinate increase in increment
    of 10 by G00.   result X=20.:
6   G# 101 ;                     // Restore #1000, G00/G01/G02/G03/
    G33/G34/G35
7   G# 102 :                     // Restore #1004, G90/G91
8   M99;                    // Return to main program
```

13.1.4 Introduction of Argument Usage

- Macro is able to execute the instructions designed by developers, from simple state-changing to complicated multi-processing.
- If a macro only deals with one process at the same time, then developers have to write uncountable macros for all sorts of situations.

- For example, the function of a macro is to cut 10*10 cm squares, if user wants to cut 20*20 cm squares then a new macro is needed.
- This kind of macro is too inflexible. However, if the content is able to be adjusted through arguments, for example, the edge of a square, then it is much more flexible in capability.
- Example:
  - In main program, system executes G201 to cut a 10*10 cm square.
  - In main program, system executes G201 to cut a 20*20 cm square.
  - In main program, system executes G202 to cut a square, length of edge of which is decided by argument.

Example_Main

```
1   // Example002_Main
2   G90 G00 X0. Y0.;
3   G200;
4   G201;
5   G202 X30.;
6   M30;
```

Example_G0200

```
1   // G0200
2   % @MACRO
3   # 101 :=# 1000 ;
4   # 102 :=# 1004 ;
5   G91 G01 X10.;
6   G91 G01 Y10.;
7   G91 G01 X- 10 .;
8   G91 G01 Y- 10 .;
9   G# 101 ;
10  G# 102 ;
11  M99;
```

Example_G0201

```
1   // G0201
2   % @MACRO
3   # 101 :=# 1000 ;
4   # 102 :=# 1004 ;
5   G91 G01 X20.;
6   G91 G01 Y20.;
7   G91 G01 X- 20 .;
8   G91 G01 Y- 20 .;
9   G# 101 ;
10  G# 102 ;
```

```
11   M99;
```

Example_G0202

```
 1   // G0202
 2   % @MACRO
 3   # 101 :=# 1000 :
 4   # 102 :=# 1004 ;
 5   # 103 :=# 24 :
 6   G91 G01 X# 103 :
 7   G91 G01 Y# 103 ;
 8   G91 G01 X-# 103 :
 9   G91 G01 Y-# 103 ;
10   G# 101 :
11   G# 102 ;
12   M99;
```

13.1.5 Arguments Explanation

- Arguments consist of 26 letters. Expect G/N/O, each letter has a corresponding local variable (# variable). They are shown in the table below.

Argument	#		Argument	#		Argument	#	
A	#1		J	#5		S	#19	
B	#2		K	#6		T	#20	
C	#3		L	#12		U	#21	
D	#7		M	#13		V	#22	
E	#8		N			W	#23	
F	#9		O			X	#24	
G			P	#16		Y	#25	
H	#11		Q	#17		Z	#26	
I	#4		R	#18				

Instruction			
Category	Axis Argument	Condition Argument	Special Argument	Exception Argument

- Argument will be read and occupied by macro.
- "Read" means there is # variable to which corresponds to the argument in the macro, so the macro reads the argument to start operation.
- "Occupied" means argument is not able to be read by other macro after it is read.
- It doesn't mean argument is occupied if it is read. The occupation depends on the macro characteristics. Please refer to the next 2 sections, "**Macro Explanation of process order**" & "**Macro Characteristics**".
- It doesn't mean argument will not be occupied if it is not read. The occupation depends on the characteristics of the argument. Please refer to the following introduction of this chapter.
- If there are repeated arguments in the same line, only the last argument is read, which means the next argument override the previous one.
- Basically, argument is separated into categories as below:

Category	Argument	Characteristic	Other Explanation
Axis Argument	X Y Z A B C I J K U V W	As long as one of the arguments is occupied by a macro, other axis arguments are also occupied by that macro.	• B code is defined by Pr3806 Second auxiliary code | Pr3806	Type	Explanation
---	---	---
0	Axis Argument	
1	Auxiliary B code	Put value of B code into R5	

Category	Argument	Characteristic	Other Explanation
Condition Argument	F S T D E H P Q R M B	Compared to axis argument, condition argument is independent from each other. If one of the condition argument is occupied, only that one is occupied instead of all condition arguments.	• T code is defined by Pr3215 Enable T code call mode.

• T code is defined by Pr3215 Enable T code call mode.

Pr3215	Type	Explanation
0	T code auxiliary code do not call T000	only put T code value into #1036 and the corresponding R value (each axis has different corresponding R value)
1	Call T0000 through sub-program	do not accept any argument. Don't regard as macro.
2	Call T0000 through macro	accept argument.

• M code is defined by Pr3601~3610 M code Macro call registry

Pr3601~3610	Type	Explanation
No login M code	M code auxiliary code	only put M code value into #1038 and the corresponding R value (each axis has different corresponding R value)

Category	Argument	Characteristic	Other Explanation		

			Pr3601~3610	**Type**	**Explanation**
			Login M code	M code macro	accept argument.

- B code is defined by Pr3806 Second auxiliary code

Pr3806	**Type**	**Explanation**
0	Axis Argument	
1	B code axuiliary code	put B code value into R5

Category	Argument	Characteristic	Other Explanation
Special Argument	L	L argument is setting up macro repeated counts. • For example: G200 L10, it means G200 will continuously execute ten times. • Even if the user doesn't input L argument when calling the macro, the system will automatically fill in L=1 to let macro execute once.	T code macro do not read L code. As a result, no matter what L code value is, T code macro always execute once.
Exception Argument	G N O	These three letters is the keyword in the controller syntax, so they are unable to be used as argument. That's why there are not corresponding # variables.	

Type	Explanation
G	Use as G code instruction or G code macro.
N	Use as program flag
O	Head of file name of normal processing program.

13.1.6 Interpreting Order of Macro

- Before explaining how macro reads and occupies arguments, user have to understand the interpreting order of macro.
- System starts interpreting from the first row and it is no problem if there is only one macro/instruction in each row.
- However, if there are several macro or instructions in a row, system interprets them according to the interpreting order of macro.
  Following table is the interpreting order of macro.

Order	Type	Particulars	Example
1	part of G code	G code macro	G73, G84, etc
		Modal G code	G15, G17, G70, etc
		One-shot G code	G65
		The interpreting order of above instructions/macro depends on the sequence of writing, from left to right.	
2	macro	M code macro	
		T code macro	
		The interpreting order of above instructions/macro depends on the sequence of writing, from left to right.	
3	S code		
4	F code		
5	H code		
6	D code		
7	T code		
8	M code		
9	B code		
10	Function G code		G04, G51, G68, etc

Order	Type	Particulars	Example
11	Interpolation G code		G00, G01, etc

- Normally, the interpreting order also represents the order of occupying argument, which means "the macro interpreted first occupies the arguments first."
Nonetheless, this is not absolutely right, because the occupying order also depends on macro characteristics.
Some macros are interpreted first but executed last. Hence, this situation does not comply to the rule, "earlier interpreted, earlier occupying."
- Example:
    a. Function G code is interpreted earlier than interpolation G code is, so function G code occupies axis argument first.
    Even though interpolation G code is on the left to the function G code in the same block, function G code still occupies axis argument earlier.
    b. In table above, interpreting order of both M code macro and T code macro are level 2, which means if if T code and M code are in the same row,
    interpreting order depends on the sequence, the code on the left is interpreted first.

13.1.7 Macro Characteristics

- When macro is reading occupied arguments, besides the basic operation explanation of argument above, there are some characteristics which is briefly explained as below:

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Activate Condition
G code and G code macro	None
T code macro	- Pr3215 Enable T code call mode. This parameter is set up as 2. After rebooting, system regard T code as T code macro. - If T code is considered T code macro, it only executes T code macro if T argument (#20) is not occupied.
M code macro	- Pr3601~3610 M code Macro call registry. M code is entered in this section of parameter. After rebooting, system regard M code as M code macro. - If M code (#13) argument is occupied, M code macro do not execute. - If any axis argument is occupied, M code macro do not execute.

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Character
G code and G code macro	• Inherit Function 　• If interpolation mode (#1000) is updated, G code inherit value from it. This is inherit function. 　• Because of inherit function, as long as axis instruction is inputted, G code is executed. 　• For example：System activate X, Z axes 　　G98 G83 Z-40.0 R-5.0 P0.0 Q10.0 F1.5; 　　X-3.; // execute G83 X-3. again
T code macro	• T code updates different variables according to the type of T code. Following are corresponding updating rules. **Table:** • If user want to capture T code value in a T code macro, please use T code variable (#1036). • If user want to capture T code value in other macros, please use T argument (#20). • Only when T code is auxiliary code, the corresponding R value of T code will be updated. If T code is used as macro or argument, the corresponding R value of T code is not updated.

T code condition update	#1036	#20	R value
Macro	O	X	X
Sub-program	O	X	X
Auxiliary Code	O	X	O
Argument	X	O	X

Category	Character
M code macro	• M code updates different variable according to the type of M code. Following are corresponding updating rules.

T code condition update	#13	R value
Macro	O	X
Auxiliary Code	X	O
Argument	O	X

• #13 is updated in M code macro or argument. Please note that this is different from specification of T code.
• Only when M code is auxiliary code, the corresponding R value of M code will be updated. If M code is used as macro or argument, the corresponding R value of M code is not updated.

Condition   Character   **Argument reading and occupation**   Multi same code instruction in a row

Other command in macro

Category	Argument reading and occupation
G code and G code macro	• While executing G code macro, argument read by G code macro is occupied. • If axis argument is read, all axis arguments are occupied simultaneously.
• Modal G code • One-shot G code	• It occupies all the argument except T code.
T code macro	• While executing T code macro, system read arguments which are needed to. No matter whether argument is occupied or not. • After executing T code macro, all arguments read by T code macro are occupied. If axis argument is read, all axis arguments are occupied simultaneously. • Repeated count of T code macro is not defined by L code, and T code macro always executes only one time. • The L code argument (#12) is not read by system, even if there is one in T code macro. System just puts 1 into #12.

M code macro	• After executing M code macro, system occupies all arguments which are read by M code macro.
	    ◦ If M code macro doesn't read T argument, then after finishes executing M code macro, T argument is not occupied.
	    ◦ If M code macro reads T argument, then after finishes executing M code macro, T argument is occupied.

Condition   Character   Argument reading and occupation   **Multi same code instruction in a row**

Other command in macro

Category	Multi same code instruction in a row				
G code and G code macro	• Being interpreted from left to right • Only the last G code reads and occupies argument. G code in the front do not read or occupy any argument. • The last G code can be sorted into situations as below				

Category		Explanation	Read occupied argument	Execution
• G code macro		Immediately	Read and occupy argument	Execute immediately
• Call modal macro	G66	Immediately	Read and occupy argument	Do not execute immediat System executes G code instruction is finished.
	G66.1	Immediately	Read and occupy argument	Do not execute immediat System executes G code instruction is finished.
• Interpolation G code • Function G code		Immediately	No	If other instructions in th corresponding argument

Category	Multi same code instruction in a row
T code macro	• If multi T code macros are in the same row. Each T code is interpreted sequentially and each T code macro can read a
M code macro	• If multi M code macro are in the same row, only the first M code macro is interpreted. • Pr3810 Parallel executing multiple M code in one block is for auxiliary M code not M code macro. Hence, even if Pr3810

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Command in macro
G code macro	• G code in G code macro can be instruction or macro. • M code in G code macro can be auxiliary code or macro. • T code in G code macro can be auxiliary code, macro or sub-program.
T code macro	• G code in T code macro can be instruction or macro. • M code in T code macro is only considered auxiliary code. • T code in T code macro is only considered auxiliary code.

Category	Command in macro
M code macro	• G code in M code macro can be instruction or macro. • M code in M code macro is only considered auxiliary code. • T code in M code macro is only considered auxiliary code.

13.1.8 Macro Calling Example

- In macro characteristics, many macro usage conditions have been mentioned. Next, example of macro calling argument will be provided

- If Doesn't especially mention any macro example. It is available to all macros.

01_Argument and program variable (local variable)

- Except the exception argument (G,N,O), all arguments correspond to a # variable (local variable)

Main	

```
1  // Main
2  G200 A1 B2 C3 D7 E8 F9 H11 I4 J5 K6 L12 M13 P16 Q17 R18 S19
   T20 U21 V22 W23 X24 Y25 Z26;
3  M30;
```

G0200	

```
1   // G0200
2   % @MACRO
3
4   //Axis Argument
5   @101  := # 1 :                              // A
6   @102  := # 2 :                              // B
7   @103  := # 3 :                              // C
8   @104  := # 4 :                              // I
9   @105  := # 5 :                              // J
10  @106  := # 6 ;                              // K
11  @121  := # 21 :                             // U
12  @122  := # 22 :                             // V
13  @123  := # 23 :                             // W
14  @124  := # 24 :                             // X
15  @125  := # 25 :                             // Y
16  @126  := # 26 ;                             // Z
17
18  //Condition Argument
19  @107  := # 7 ;                              // D
20  @108  := # 8 ;                              // E
```

```
21   @109  := # 9 ;                          // F
22   @111  := # 11 ;                         // H
23   @113  := # 13 ;                         // M
24   @116  := # 16 ;                         // P
25   @117  := # 17 ;                         // Q
26   @118  := # 18 ;                         // R
27   @119  := # 19 ;                         // S
28   @120  := # 20 ;                         // T
29
30   //Special Argument
31   @112  := # 12 ;                         // L
32
33   M00;                            // Switch the screen to
     [Diag.] → [Display Global] and [Display Coord.]
34   WAIT();
35   M99;
```

02_Repeated argument

- Only the last repeated argument is read.
- Because same argument put value into the same # variable, the last one overrides the previous one.

Main

```
1   // Main
2   G01 X0.;
3   G200 X10. X- 10 .;      // X argument has been written twice,
    only X-10 is read by G200
4                           // #24 is input 10 by X10 first
5                           // then input -10 by X-10.
6                           // G01 will be executed after completing
    G200,
7                           // because G01 can't read any argument X
    which is occupied by G200
8   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 100  := # 1000 :       // Backup interpolation mode
4   # 101  := # 24 :         // The #24 is -10.
5   G01 X# 101 ;             // Move to X-10.
6   M00;
7   WAIT();
8   G# 100 ;                 // Restore to interpolation mode
9   M99;
```

03_Axis argument and condition argument

- All the axis arguments are occupied if one of them is occupied.
- Condition arguments are independent from each other.

Main

```
1  // Main
2  G01 X0. Y0. Z0.;
3  G200 X10. Y20. Z30.;              // Though G200 only
   reads X argument
4                                    // Y, Z arguments are
   also occupied
5                                    // G01 will be executed
   after completing G200
6                                    // because G01 can't
   read any argument which is occupied by G200
7  G01 X0. Y0. Z0. F100.;
8  G201 P20. X10. Y20. Z30. F200.;   // Since G201 only reads
   P argument
9                                    // X, Y, Z, F arguments
   are not occupied
10                                   // G01 will be executed
   after completing G201 and
11                                   // read X10. Y20. Z30.
   F200.
12                                   // , and then execute
   the corresponding actions
13 M30;
```

G0200

```
1  // G0200
2  % @MACRO
3  @1 :=# 24 ;               // Only reads X argument
4  M00;
5  WAIT();
6  M99;
```

G0201

```
1  // G0201
2  % @MACRO
3  @1 :=# 16 ;               // Only reads P argument
4  M00;
5  WAIT();
```

```
6    M99;
```

04_H code as a condition argument

- H code is condition argument

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 X10. Y20. Z30. H40.;              // G200 only reads H
     argument
4                                           // H argument is a
     condition argument
5                                           // G01 will be executed
     after completing G200.
6                                           // G01 read and occupy
     X10. Y20. Z30.
7                                           // and then execute the
     corresponding actions
8    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 11 ;         // Only reads H argument
4    M00;
5    WAIT();
6    M99;
```

05_H code as an axis argument

- If Pr3809 "*Are UVW incremental command of XYZ axes" is set to 1,H code is not only an axis argument but also a condition argument.
- Nevertheless, H argument is regarded as condition argument and read in macro. Therefore, after completed the execution of G code macro, the H argument is occupied again as an axis argument and system does the corresponding action.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 X10. Y20. Z30. H40.;              // G200 only reads H
     argument
```

```
4                                        // Since Pr3809=1, the H
argument is regarded as both axis argument and condition
argument
5                                        // However, it is
occupied as a condition argument in MACRO
6                                        // G01 is executed after
completing G200
7                                        // G01 reads and
occupies X10. Y20. Z30. H40., and
8                                        // system executes the
corresponding action
9                                        // Since the axis
argument of H40. is not occupied, it is then occupied as an
axis argument by
10                                       // G01 again.
11   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 11 ;        // only reads H argument
4   M00;
5   WAIT();
6   M99;
```

- When reading axis arguments in MACRO, all the axis arguments will be occupied if one of them is occupied.
- As a result, the axis argument of H argument is occupied as well.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. H40.;          // G200 only reads X
argument
4                                        // Since Pr3809=1, the H
argument is regarded as both axis argument and condition
argument
5                                        // The axis argument of
H argument is also occupied
6                                        // G01 will be executed
after completing G200
7                                        // G01 can't read any
argument since the axis arguments are all occupied by G200.
8                                        // The axis argument of
the H argument is also included and won't be read.
9   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 24 ;        // Only reads X argument
4   M00:
5   WAIT();
6   M99;
```

06_B code as an axis argument

- If Pr3806 Second auxiliary code is set to 0, B code is regarded as an axis argument.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. B40.;            // G200 only reads B
    argument
4                                        // Since Pr3806=0, B
    argument is regarded as an axis argument
5                                        // X, Y, Z arguments is
    occupied by G200
6                                        // G01 is executed after
    completing G200
7                                        // and G01 can't read
    any argument since all arguments are occupied by G200
8   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 2 ;        // Only reads B argument
4   M00:
5   WAIT();
6   M99;
```

07_B code as a condition argument

- If Pr3806 Second auxiliary code is set to 1, B code is regarded as a condition argument.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 X10. Y20. Z30. B40.;           // G200 only reads B
     argument
4                                        // Since Pr3806=1, B
     argument is seen as a condition argument
5                                        // X, Y, Z arguments are
     not occupied by G200
6                                        // G01 is executed after
     completing G200
7                                        // G01 reads and
     occupies X10. Y20. Z30., and
8                                        // executes the
     corresponding actions
9    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 2 ;        // Only reads B argument
4    M00;
5    WAIT();
6    M99;
```

08_Multiple G code macros in the same line

- When multiple G codes are written in the same line, no matter G code command or G code macro, system interprets all of them in order.
- However, only the G code macro written last reads and occupies arguments. The former G codes do not read or occupy any argument even though they are interpreted first.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 G201 X10. P20.;        // Since 2 G code macros are
     written in the same line
4                                // only the last G code macro is
     able to read the argument
5                                // Though G201 didn't read the X
     argument
6                                // G200 still can't read the X
     argument, because X argument is axis argument
```

```
 7                                // G201 is executed after
    completing G200, and
 8                                // G01 is be executed after G201
    read and occupied P argument
 9                                // G01 reads and occupies X10.,
    and
10                                // then executes the
    corresponding actions
11   M30;
```

G200

```
 1   // G0200
 2   % @MACRO
 3   # 100  := # 1000 :              // Backup interpolation mode
 4   # 101 :=# 24 ;                  // Read X argument which is
    unable to be read
 5   G01 X# 101 ;                    // Since #101 has no value,
    this line won't be executed
 6   G01 X0.;                  // This line will be executed
 7   M00:
 8   WAIT();
 9   G# 100 ;                        // Restore interpolation mode
10   M99;
```

G201

```
 1   // G0201
 2   % @MACRO
 3   # 100  := # 1000 ;              // Backup interpolation mode
 4   # 101 :=# 16 ;                  // Read P argument which is
    able to be read
 5   G01 X# 101 ;                    // This line will be executed
 6   G01 X0.;
 7   M00;
 8   WAIT();
 9   G# 100 ;                        // Restore interpolation mode
10   M99;
```

09_G code instruction and G code macro in the same line

- When multiple G codes or G code macros are written in the same line, they are interpreted in order.
- Only the G code macro written last can read and occupy arguments. Though the former G codes are interpreted first, they do not read and occupy any argument.

Main

```
1   // Main
2   G90;
3   G00 X0. Y0. Z0.;                // G00, spindle moves to X0 Y0
    Z0
4   G200 G01 X10. F100.;            // Since 2 G codes are in the
    same line.
5                                   // only the last G code can read
    the argument
6                                   // G200 is executed but no
    arguments are read
7                                   // G01 X10. F100. is executed
    after completing G200
8
9   G00 X0. Y0. Z0.;                // G00, spindle moves to X0 Y0
    Z0
10  G01 G201 X10. F100.;            // Since 2 G codes are in the
    same line.
11                                  // G01 is interpreted first,
    interpolation mode changes to G01.
12                                  // G201 is interpreted secondly.
13                                  // In G201, system executes G01
    X10. F100.
14                                  // Since G201 reads and occupies
    the axis argument, all the axis arguments are occupied.
15                                  // Therefore, there is no moving
    instruction left for G01, so no movement.
16
17  G00 X0. Y0. Z0.;                // G00, spindle moves to X0 Y0
    Z0
18  G01 G202 X10. F100.;            // Since 2 G codes are in the
    same line
19                                  // G01 is explained first,
    interpolation mode changes to G01
20                                  // G202 is interpreted
    secondely.
21                                  // In G202, no action is
    executed and no arguments are read or occupied.
22                                  // Therefore, there are axis
    arguments left for G01, so system executes G01 X10. F100.
23  M30;
```

G200

```
1   // G0200
2   % @MACRO
3   # 100  := # 1000 ;              // Backup interpolation mode
```

```
4    # 101  := # 24 ;                // Read X argument but it's
     unable to read
5    G# 100  X# 101  F# 9 ;          // This line is not
     executed since #101 has no value. However, since #100=0, the
     interpolation mode changes to G00.
6    M00;
7    WAIT();
8    M99;
```

G201

```
1    // G0201
2    % @MACRO
3    # 100  := # 1000 :             // Backup interpolation mode
4    # 101  := # 24 ;               // Read X argument and read
     X=10.
5    G# 100  X# 101  F# 9 ;         // Because #100 = 1,
     system executes G01.
6    M00;
7    WAIT();
8    M99;
```

G202

```
1    // G0202
2    % @MACRO
3    M00;
4    WAIT();
5    M99;
```

10_T code macro

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- Executing T code macro if T argument is not occupied.
- #20 and R3 have no value, but #1036 shows the value of T code.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 X10. Y20. Z30.;       // T argument is not occupied and
     Pr3215=2, so
4                              // system executes T code macro
5                              // With X argument being read in T
     code macro,
```

```
6                              // Y, Z arguments are occupied by T
code macro because all of them are axis arguments.
7                              // G01 is executed after completing
T01
8                              // G01 can't read any argument since
the arguments in the line are all occupied by T coda macro.
9    M30;
```

T0000

```
1    // T0000
2    % @MACRO
3    # 101  := # 24 ;           // only reads X argument
4    M00;                       // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
5                               // Observe #20/#1036/#101
6    WAIT();
7    M99;
```

11_G code macro and T code macro in the same line

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- When G code macro and T code macro are in the same line, G code macro is interpreted first then the T code macro.
- If the T code is occupied by G code macro then T code macro is not executed.
- If the T code is not occupied by G code macro then T code macro is executed after G code macro is completed.
- There is only one T0000 in system, but in the example, there are two T code macros named after T0000_T01 and T0000_T02.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 G200 X10. Y20. Z30. F1000;     // According to the
interpretation order, G code macro is interpreted first.
4                                       // There is a T argument
read in G code macro (G0200), so
5                                       // T argument is
occupied and T code macro is not executed.
6                                       // G01 is executed after
completing G code macro (G0200).
7                                       // G01 reads and
occupies X10. Y20. Z30. F1000, and
8                                       // executes the
corresponding actions
9
```

```
10   T02 G201 X10. Y20. Z30. F1000.;     // According to the
     interpretation order, G code macro is interpreted first.
11                                       // There is no T
     argument read in G code macro (G0201), so
12                                       // T argument is not
     occupied, and
13                                       // T code macro is
     executed after G code macro (G0201) is completed
14                                       // Because T02 reads all
     arguments, G01 can't read any argument
15   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 20 ;          // Only reads T argument
4   M00;
5   WAIT();
6   M99;
```

T0000_01

```
1    // Not executing this process
2    // T0000
3    % @MACRO
4    # 201  := # 1000 ;
5    # 202  := # 1004 ;
6    # 101  := # 24 ;
7    # 102  := # 25 ;
8    # 103  := # 26 ;
9    # 104  := # 9 ;
10   M00;               // Switch the screen to [Diag.] →
     ［Display Global] and [Display Coord.]
11   WAIT();
12   G91 G01 X# 101  Y# 102  Z# 103  F# 104 ;
13
14   G# 201 ;
15   G# 202
16   M99;
```

G0201

```
1    // G0201
2    % @MACRO
```

```
3    # 101  := # 24 ;           // Only reads X argument
4    M00;                       // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
5    WAIT();
6    M99;
```

T0000_02

```
1    // T0000
2    % @MACRO
3    # 201  := # 1000 ;
4    # 202  := # 1004 ;
5    # 101  := # 24 ;
6    # 102  := # 25 ;
7    # 103  := # 26 ;
8    # 104  := # 9 ;
9    M00;                       // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
10   WAIT();
11   G91 G01 X# 101  Y# 102  Z# 103  F# 104 ;     // T code macro
     can still read the arguments and execute the corresponding
     actions
12
13   G# 201 ;
14   G# 202
15   M99;
```

12_T code macro is not affected by L Argument

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- T code macro does not read or occupy L argument

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 L2;                // Though T0000 reads #12,
4                           // #12 is equal to "1", not "2" given by L
     argument
5                           // Execute G01 after completed T01
6                           // G01 reads and occupies L2, and
7                           // executes the corresponding actions (G01
     L2 is meaningless)
8    M30;
```

T0000

```
1   // T0000
2   % @MACRO
3   # 101  := # 12 ;          // Only reads L argument
4   M00;                      // Switch the screen to [Diag.] →
    「Display Global] and [Display Coord.]
5   WAIT();
6   M99;
```

13_Multiple T code macros in a line

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- When multiple T code macros are written in the same line, T code is interpreted in order and every T code macro can read the arguments.
- There is only one T0000 in system, but in the example, there are two T code macros named after T0000_T01 and T0000_T02.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   T01 T02 X10. Y20. Z30.;     // This line is executed twice
4                               // T01 is executed first then
    T02
5                               // Both T code read X, Y, Z
    arguments successfully
6                               // If there is any other macro
    going to read X, Y, Z,
7                               // it reads nothing since
    arguments are already occupied by T code macro.
8                               // G01 is executed after
    completing T02 with all argument occupied.
9   M30;
```

T0000_T01

```
1   // T0000_T01
2   % @MACRO
3   # 100  := # 1000 ;      // Backup interpolation mode
4   # 101  := # 24 ;        // Read X argument
5   # 102  := # 25 ;        // Read Y argument
6   # 103  := # 26 ;        // Read Z argument
7   IF # 1036 =  1  THEN    // T01 → #1036=1
8      G01 X# 101 ;         // X axis moving
9      G01 X0.;
```

```
10    END_IF;
11
12    IF # 1036  =  2  THEN     // The section is not executed
13        G01 Y# 102 ;
14        G01 Y0.;
15    END_IF;
16
17    M00;                      // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
18    WAIT();
19    G# 100 ;                  // Restore interpolation mode
20    M99;
```

T0000_T02

```
1    // T0000_T02
2    % @MACRO
3    # 100  := # 1000 :        // Backup interpolation mode
4    # 101  := # 24 ;          // Read X argument
5    # 102  := # 25 ;          // Read Y argument
6    # 103  := # 26 ;          // Read Z argument
7    IF # 1036  =  1  THEN     // The section is not executed
8        G01 X# 101 ;
9        G01 X0.;
10   END_IF;
11
12   IF # 1036  =  2  THEN     // T02 → #1036=2
13       G01 Y# 102 ;          // Y axis moving
14       G01 Y0.;
15   END_IF;
16
17   M00;                      // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
18   WAIT();
19   G# 100 ;                  // Restore interpolation mode
20   M99;
```

14_M code macro

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.
- If M code is registered as an M code macro, it is executed only when the M code and all the other axis arguments are not occupied.
- While M code macro is executed, all the arguments are occupied except T.
- M argument (#13) is updated whether the M code is registered as M code macro or not.
- The corresponding R value of M code is updated only if M code is regarded as auxiliary code. R value is not updated if M code is regarded as macro or argument.

Main

```
1    // Main
2    G01 X0. Y0. Z0. F100.;
3    M123 X10. Y20. Z30. F300.;    // No M argument and no axis
     argument is occupied, so
4                                  // M code macro M0123 is
     executed.
5                                  // While M code macro is
     executed, aside from the arguments read by the M code macro,
6                                  // all other arguments are
     occupied as well except T argument.
7                                  // Though the F argument is not
     read by M0123, it is occupied, then the feedrate is not
     updated.
8                                  // G01, executed after
     completing M123,
9                                  // can't read any argument since
     the arguments are all occupied.
10
11   G01 X0. Y0. Z0.;              // The F in this line is F100,
     since the argument in previous line is occupied by M code
     macro
12   M30;
```

M0123

```
1    // M0123
2    % @MACRO
3    # 100  := # 1000 ;                 // Backup interpolation mode
4    # 101  := # 24 ;                   // Read X argument
5    # 102  := # 25 ;                   // Read Y argument
6    # 103  := # 26 ;                   // Read Z argument
7    G00 X# 101  Y# 102  Z# 103 ;     // G00 moving along X, Y,
     Z axis
8    M00;
9    WAIT();                  // Switch the screen to [Diag.]
     → [Display Global] and [Display Coord.]
10                                     // user should found #13=123
11                                     // Switch the screen to [PLC
     Status] → [PLC Register]
12                                     // User should found R1=0
13   G# 100 ;                          // Restore interpolation mode
14   M99;
```

15_G code macro and M code macro in the same line

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.

- When G code macro and M code macro are written in the same line, G code macro is interpreted before M code macro.
- If M code is occupied, the M code macro is not executed.
- On the opposite, if M code is not occupied, the M code macro is executed after G code macro completed.
- All the arguments except T will be occupied after executing the M code macro.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G1301 M1301 X10. Y20. Z30. P2 F1000.;     // G code macro is
    interpreted first.
4                                             // The M argument is
    read by G code macro (G1301)
5                                             // The M argument is
    regarded occupied, M code macro is executed.
6                                             // Since the X, Y, Z
    arguments is not occupied by G code macro (G1301).
7                                             // G01 is executed
    after completing the G code macro (G1301).
8                                             // G01 reads and
    occupies X10. Y20. Z30. P2 F1000., and
9                                             // then executes the
    corresponding instructions.
10
11  G1302 M1301 X10. Y20. Z30. F1000.;        // G code macro is
    interpreted first
12                                            // Only P argument
    is read by G code macro (G1302)
13                                            // The M argument is
    regarded unoccupied, and
14                                            // no other axis
    arguments are occupied
15                                            // The M code macro
    is executed after completing the G code macro (G1302)
16                                            // The M code macro
    (M1301) do not read any argument,
17                                            // but all arguments
    except T are regarded occupied after the M code macro is
    executed
18                                            // G01, executed
    after completing the M code macro,
19                                            // can't read any
    argument since the arguments are all occupied
20  M30;
```

G1301

```
1   // G1301
```

```
2   % @MACRO
3   # 101  := # 13 ;              // Only reads M argument
4   M00;
5   WAIT();
6   M99;
```

M1301_01

```
1   // The program won't be executed
2   // M1301
3   % @MACRO
4   M00:
5   WAIT();
6   M99;
```

G1302

```
1   // G1302
2   % @MACRO
3   # 101  := # 16 ;              // Only reads P argument
4   M00:
5   WAIT();
6   M99;
```

M1301_02

```
1   // M1301
2   % @MACRO
3   M00:
4   WAIT();                 // The program does not read any
    argument
5                           // but occupies all the arguments except
    T
6   M99;
```

16_G code macro and M code macro are in the same line, and G code macro reads and occupies axis argument.

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.
- When G code macro and M code macro are written in the same line, G code macro is interpreted before M code macro.
- If axis arguments are occupied by G code macro, M code macro is not executed.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 M123 X10. Y20. Z30.;      // G code macro is interpreted
     first
4                                   // Because X argument is read by
     G code macro (G0200),
5                                   // all axis arguments are
     occupied by G200.
6                                   // M code macro therefore is
     executed since axis arguments are occupied.
7                                   // G01, executed after
     completing G code macro (G0200),
8                                   // can't read any argument since
     the arguments in the line are all occupied
9    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 24 ;          // Only reads X argument
4    M00;
5    WAIT();
6    M99;
```

M0123

```
1    //This program won't be executed
2    // M0123
3    % @MACRO
4    # 101  := # 24 ;
5    M00;
6    WAIT();
7    M99;
```

17_Multiple M code macros in a line

- Pr3601 is set to 123, and M0123 is registered as M code macro.
- Pr3602 is set to 124, and M0124 is registered as M code macro.
- When multiple M codes macro are written in the same line, only the first M code macro is interpreted.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    M123 M124 X10. Y20. Z30.;      // M123 is interpreted first,
     and only M123 is executed.
4                                   // X10. Y20. Z30. are read by M
     code macro (M0123)
5                                   // All arguments except T are
     occupied after executing the M code macro.
6                                   // G01, executed after
     completing M code macro,
7                                   // can't read any argument since
     the arguments are all occupied.
8    M30;
```

M0123

```
1    // M0123
2    % @MACRO
3    # 100  := # 1000 ;              // Backup interpolation mode
4    # 101  := # 24 ;                // Read X argument
5    # 102  := # 25 ;                // Read Y argument
6    # 103  := # 26 ;                // Read Z argument
7    G00 X# 101  Y# 102  Z# 103 ;     // G00 moving along X, Y,
     Z axis
8    M00;
9    WAIT();                         // Switch the screen to [Diag.]
     → [Display Global] and [Display Coord.]
10                                   // User should found #13=124
11                                   // Since M124 is regarded as an
     argument, which overwrote #13
12   G# 100 ;                        // Restore interpolation mode
13   M99;
```

M0124

```
1    // This program won't be executed
2    // M0124
3    % @MACRO
4    # 101  := # 24 ;
5    M00;
6    WAIT();
7    M99;
```

18_T code macro and M code macro in the same line

- Pr3215 is set to 2, T0000 is the T code macro
- Pr3601 *M code Macro call registry is set to 123/124/125, M0123 / M0124 / M0125 are registered as M coed macros
- If T code macro and M code macro are written in the same line, the macro on the left is interpreted first.
- Whether the macro on the right is executed or not is decided by the rules in previous section [Macro Characteristics]

Main

```
 1   // Main
 2   G01 X0. Y0. Z0.;
 3    T01 M1601 X10. Y20. Z30.;     // According to the interpreting
     order, T01 is interpreted first, then M1601.
 4                                  // No argument is occupied by T
     code macro, so M code macro (M1601) executes X=10.
 5                                  // Although M code macro (M1601)
     does not read any argument,
 6                                  // all the arguments except T
     are occupied
 7                                  // G01, executed after
     completing M code macro (M1601),
 8                                  // can't read any argument since
     the arguments are all occupied.
 9
10    M1602 T02 X10. Y20. Z30.;     // According to the interpreting
     order, M1602 is interpreted first.
11                                  // M code macro (M1602) does not
     reading any argument.
12                                  // Because M code macro (M1602)
     occupies all arguments except T,
13                                  // T code macro (T0000_02) is
     able to be executed since T argument is not occupied by M
     code macro.
14                                  // G01, executed after
     completing T code macro,
15                                  // can't read any argument since
     the arguments in the line are all occupied
16
17    M1603 T03 X10. Y20. Z30.;     // According to the interpreting
     order, M1603 is interpreted first.
18                                  // M code macro (M1603) only
     reads the T argument
19                                  // M code macro (M1603) will
     occupy all the arguments after the execution, including T
     argument.
20                                  // Then, T code macro (T0000_03)
     is executed since T code here is regarded as an argument.
21                                  // G01 can't read any argument
     since the arguments in the line are all occupied.
```

```
22    M30;
```

T0000_01

```
1     // T0000
2     % @MACRO
3     # 100  := # 1000 :        // Backup interpolation mode
4     # 101  := # 1004 ;        // Backup absolute/increment command
      mode
5
6     G91 G00 X10.;             // X axis moving in increment of 10.
7
8     M00;                      // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
9     WAIT();
10    G# 100 :                  // Restore the interpolation mode
11    G# 101 :                  // Restore the absolute/increment
      command mode
12    M99;
```

M1601

```
1     // M1601
2     % @MACRO
3     # 100  := # 1000 :        // Backup interpolation mode
4     # 101  := # 1004 ;        // Backup absolute/increment command
      mode
5
6     G91 G00 Y10.;             // Y axis moving in increment of 10.
7
8     M00;                      // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
9     WAIT();
10    G# 100 :                  // Restore the interpolation mode
11    G# 101 :                  // Restore the absolute/increment
      command mode
12    M99;
```

M1602

```
1     // M1602
2     % @MACRO
3     # 100  := # 1000 :        // Backup interpolation mode
4     # 101  := # 1004 ;        // Backup absolute/increment command
      mode
```

```
 5
 6    G91 G00 Y10.;           // Y axis moving in increment of 10.
 7
 8    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
 9    WAIT();
10    G# 100 ;                // Restore the interpolation mode
11    G# 101 ;                // Restore the absolute/increment
      command mode
12    M99;
```

T0000_02

```
 1    // T0000
 2    % @MACRO
 3    # 100  := # 1000 ;       // Backup interpolation mode
 4    # 101  := # 1004 ;       // Backup absolute/increment command
      mode
 5
 6    G91 G00 X10.;           // X axis moving in increment of 10.
 7
 8    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
 9    WAIT();
10    G# 100 ;                // Restore the interpolation mode
11    G# 101 ;                // Restore the absolute/increment
      command mode
12    M99;
```

M1603

```
 1    // M1603
 2    % @MACRO
 3    # 100  := # 1000 ;       // Backup interpolation mode
 4    # 101  := # 1004 ;       // Backup absolute/increment command
      mode
 5    # 102  := # 20 ;         // Read data of #20, the line will
      occupy the T argument
 6
 7    G91 G00 Y10.;           // Y axis moving in increment of 10.
 8
 9    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
10    WAIT();
11    G# 100 ;                // Restore the interpolation mode
12    G# 101 ;                // Restore the absolute/increment
      command mode
```

```
13    M99;
```

T0000_03

```
1     // This program won't be executed
2     // T0000
3     % @MACRO
4     # 100  := # 1000 ;        // Backup interpolation mode
5     # 101  := # 1004 ;        // Backup absolute/increment command
      mode
6
7     G91 G00 X10.;          // X axis moving in increment of 10.
8
9     M00:                    // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
10    WAIT();
11    G# 100 :                // Restore the interpolation mode
12    G# 101 :                // Restore the absolute/increment
      command mode
13    M99;
```

19_One-shot macro calling (G65)

- G65 executes the macro file assigned by the P argument

Main

```
1     // Main
2     G01 X0. Y0. Z0.;
3     G65 P1 X10. Y20. Z30.;          // Execute O0001
4     M30;
```

O0001

```
1     // O0001
2     % @MACRO
3     # 101  := # 24 ;
4     # 102  := # 25 ;
5     # 103  := # 26 ;
6     M00;
7     WAIT();
8     M99;
```

20_Modal macro calling (G66)

- G66 is executed every time a movement block is completed.

Main

```
1   // Main
2   G01 X- 5 . Y- 5 . Z- 5 .;
3   G200 G66 P1 X10. Y20. Z30. ;      // G200 in this line can't
    read X argument
4   G200 X10.;                        // G200 in this line can
    read X argument
5                                     // There are 2 lines of G01
    movement blocks in G200
6                                     // G66 P1 X10. Y20. Z30. is
    executed every time G01 movement block is finished.
7   G67;
8   M30;
```

G0200_01

```
1   // G0200_01
2   % @MACRO
3   # 100  := # 1000 ;        // Backup interpolation mode
4   # 101  := # 24 ;          // Executing G200 for the first
    time, G200 can't read X argument, so
5   G01 X# 101 ;              // this line is not executed.
6   G01 X0.;
7   M00;
8   WAIT();
9   G# 100 ;                  // Restore the interpolation mode
10  M99;
```

G0200_02

```
1   // G0200_02
2   % @MACRO
3   # 100  := # 1000 ;        // Backup interpolation mode
4   # 101  := # 24 ;          // Executing G200 for the second
    time, G200 is able to read X argument, so
5   G01 X# 101 ;               // this line is executed
6                             // This line is a movement block, after
    finished, G66 P1 X10. Y20. Z30. is executed.
7   G01 X0.;                  // This line is a movement block, after
    finished, G66 P1 X10. Y20. Z30. is executed.
8   M00;
9   WAIT();
```

```
10    G# 100 ;                // Restore the interpolation mode
11    M99;
```

O0001

```
1     // O0001
2     % @MACRO
3     # 100  := # 1000 ;        // Backup interpolation mode
4     # 101  := # 24 :
5     # 102  := # 25 :
6     # 103  := # 26 :
7     G91 G01 X# 101  Y# 102  Z# 103 ;
8     M00;
9     WAIT();
10    G# 100 ;                // Restore the interpolation mode
11    M99;
```

21_Non modal call macro (G66.1)

- G66.1 is executed every time after a block is completed.

Main

```
1     // Main
2     G01 X- 5 . Y- 5 . Z- 5 .;
3     G66. 1  P1 X10. Y20. Z30. ;
4     G200 X10.;                      // G200 in this line can read
      the X argument
5                                     // G66.1 P1 X10. Y20. Z30. is
      executed every time the block in G200 is executed.
6     G67;                            // G66 P1 X10. Y20. Z30. is
      executed after this block is executed
7     M30;
```

G0200

```
1     // G0200
2     % @MACRO
3     # 100  := # 1000 ;        // Backup interpolation mode
4     # 101  := # 24 :          // Executing G200 for the second
      time, G200 is able to read X argument
5     G01 X# 101 ;              // This line is executed
6                              // G66 P1 X10. Y20. Z30. is executed
      after this block is executed
```

```
 7   G01 X0.;                // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
 8   M00;                    // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
 9   WAIT();                 // This line is not a block
10   G# 100 ;                // Restore the interpolation mode
11                           // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
12   M99;                    // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
```

O0001

```
 1   // O0001
 2   % @MACRO
 3   # 100  := # 1000 ;        // Backup interpolation mode
 4   # 101  := # 1004 ;        // Backup absolute/increment command
     mode
 5   # 111  := # 24 ;
 6   # 112  := # 25 ;
 7   # 113  := # 26 ;
 8   G91 G01 X# 111  Y# 112  Z# 113 ;
 9   M00;
10   WAIT();
11   G# 100 ;                  // Restore the interpolation mode
12   G# 101 ;                  // Restore the absolute/increment
     command mode
13   M99;
```

22_G65 and G66/G66.1 must be the last G code in the line

- One-shot G code macro calling (G65) and modal G code macro calling (G66/G66.1) must be the last G code in the line.

Main

```
 1   // Main
 2   G01 X0. Y0. Z0.;
 3   G65 P1 X10. Y20. Z30. G200;  // Alarm COR-013 shows up owing
     to wrong syntax in this line, because
 4                               // G65 and G200 are in the same
     line
 5                               // but G65 is not the last G
     code.
 6   M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 24 ;
4    M00:
5    WAIT();
6    M99;
```

O0001

```
1    // O0001
2    % @MACRO
3    # 101  := # 24 ;
4    # 102  := # 25 ;
5    # 103  := # 26 ;
6    M00:
7    WAIT();
8    M99;
```

13.2 MACRO XML Data Application

- MACRO is able to read xml files with special functions, which, respectively, are DBLOAD and DBOPEN. DBOPEN is used to load xml file and DBLOAD is used to read the data content.

- Application example：Following is a customized HMI, which automatically produces xml file recording the related machining data. The content of the xml file will be read and taken as reference when planning movements in macro afterwards.

尺寸編輯	模式關閉		前座外徑	12.98	線徑	1.80	方向	右旋	材質	SW-A	▼	間隙	直軸	切刀	一般
No	X 間隙	Y 送線	Z 右旋	A 上切	B 左旋	C 下切	R	K	T	起始速度
1	0.00	17.63	12.98	267.54						
2	2.26	21.34	13.20	300.87						
3	2.26	91.19	13.20	443.29						
4	0.00	21.55	12.98	116.95						
5	-6.05	21.16	12.98	150.00						

-> The customized HMI first outputs the the user-defined contents to an xml file, and save the xml file to the GNCFILES file path of which is assigned by user (refer to Pr3219).

The syntax format is defined as below,：

```xml
<?xml version="1.0" encoding="UTF-16"?>
<CycleFile>

    <Cycle Name="Cycle_HerdonProg">  ← The beginning of first data

        <Field Name="Col_Y" Value="17.63"/>
        <Field Name="Col_Z" Value="12.98"/>
        <Field Name="Col_X" Value="0.00"/>
        <Field Name="Col_A" Value="267.54"/>

    </Cycle>  ← The end of first data
    <Cycle Name="Cycle_HerdonProg">  ← The beginning of second data

        <Field Name="Col_Y" Value="21.34"/>
        <Field Name="Col_Z" Value="13.20"/>
        <Field Name="Col_X" Value="2.26"/>
        <Field Name="Col_A" Value="300.87"/>

    </Cycle>  ← The end of second data
    <Cycle Name="Cycle_HerdonProg">  ← The beginning of third data

        <Field Name="Col_Y" Value="91.19"/>
        <Field Name="Col_Z" Value="13.20"/>
        <Field Name="Col_X" Value="2.26"/>
        <Field Name="Col_A" Value="443.29"/>

    </Cycle>  ← The end of third data
    <Cycle Name="Cycle_HerdonProg">  ← The beginning of fourth data

        <Field Name="Col_Y" Value="21.55"/>
        <Field Name="Col_Z" Value="12.98"/>
        <Field Name="Col_X" Value="0.00"/>
        <Field Name="Col_A" Value="116.95"/>

    </Cycle>  ← The end of fourth data
    <Cycle Name="Cycle_HerdonProg">  ← The beginning of fifth data

        <Field Name="Col_Y" Value="21.16"/>
        <Field Name="Col_Z" Value="12.98"/>
        <Field Name="Col_X" Value="-6.05"/>
        <Field Name="Col_A" Value="150.00"/>

    </Cycle>  ← The end of fifth data

</CycleFile>
```

-> User have to write the configuration file of XML file by their own.

   Configuration file(schema file) defines the data to be read and the variable where data is put while DBLOAD is used.

   The syntax format is defined as below, and the configuration file should be saved in OCRes\\Common\\Schema\\

   If Dipole is enable, the schema file should be in the folder OCRes\\Common\\Schema\\ in controller.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<Schema>

    <Cycle name="Cycle_HerdonProg">
```

```xml
<Field>

    <Name>Col_X</Name>
    <InputStorage>@1200</InputStorage>   ← The variable Col_X saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
<Field>

    <Name>Col_Y</Name>
    <InputStorage>@1201</InputStorage>   ← The variable Col_Y saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
<Field>

    <Name>Col_Z</Name>
    <InputStorage>@1202</InputStorage>   ← The variable Col_Z saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
<Field>

    <Name>Col_A</Name>
    <InputStorage>@1203</InputStorage>   ← The variable Col_A saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
<Field>

    <Name>Col_B</Name>
    <InputStorage>@1204</InputStorage>   ← The variable Col_B saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
<Field>

    <Name>Col_C</Name>
    <InputStorage>@1205</InputStorage>   ← The variable Col_C saves in
    <InputFormat>Variant</InputFormat>
    <DefaultValue></DefaultValue>

</Field>
    </Cycle>
</Schema>
```

-> MACRO example

```
// Load GNCFILES\\Number of test data, total 5 data, therefore @1:=5;
@1:=DBOPEN("Test");
```

```
// Load the first data, DBLOAD argument is 0
// @1200=0.00 @1201=17.63 @1202=12.98 @1203=267.54
DBLOAD( 0 );


// Load the second data, DBLOAD argument is 1
// @1200=2.26 @1201=21.34 @1202=13.20 @1203=300.87
DBLOAD( 1 );
```

14 **Preface**

In order to increase the flexibility of the controller application, Syntec controller provides MACRO program editing functions. When the processing program is declared in the MACRO format, specific mathematical functions are applicable like other programming languages. In this way, in addition to the original movement and compensation command functions, logical judgment and mathematical calculation functions are also included.

15 **File Format**

The first line of the program should be declared as the title line with "**%**" and add the keyword "**@MACRO**". Otherwise, the file will be regarded as a normal ISO format file, and user is not able to to use the full functionality of the MACRO. In addition, each line of the program content must be followed by a **semicolon** "**;**", but there are exceptions to some of the syntax, refer to the syntax description.

ISO format	MACRO format
Do not support MACRO syntax. (While there are syntax can not be complied correctly, there may be alarm "COM-003."	Support full functionality of MACRO syntax.
With or without semicolon at the end of line is acceptable.	1. Except the special syntax, every line should end with semicolon. 2. If the line does not end with semicolon, CNC combine it together with the next line when checking the syntax. If there is no alarm, the NC program run normally; otherwise, there comes the alarm "COM-008," which is "line does not end with semicolon." Example: _see table below_

	Original	what CNC read	Description
Alarm	%@MACRO #1:=SIN(100) G01 Y100.; M99;	%@MACRO #1:=SIN(100) G01 Y100.; M99;	Due to wrong syntax after combined, there is alarm, "COM-008."
No alarm	%@MACRO G01 X100. G01 Y100.; M99;	%@MACRO G01 X100. G01 Y100.; M99;	Due to correct syntax even after combined, there is no alarm. The spindle moves to X100. Y100.

ISO format	MACRO format
While using "(...)", the ... is regarded as comment.	While using "(*...*)", the ... is regarded as comment.
Programming according to Pr3201.	Programming according to Lathe C-Type.

Note:

1. It is recommended that multi-path style(including $1 and $2) might not be used in NC Program which is called as sub-program or a macro.
   Should it be inevitable, the size of the program **must be** less than **60KB(60000bytes)**. Otherwise, there comes the alarm COR-203 Illegal NC file format
2. Not support the sub-program, which is in MACRO format, using the multi-path style (including $1, $2).
3. If the size of file is larger than 60KB(60000bytes), it would not support the syntax like IF, CASE, REPEAT, FOR, WHILE, which are longer than one line. If these syntax are used, there will be alarms.(COR-204 File size too large)
4. Only ASCII characters are acceptable in NC files. Using non-ASCII characters is thus unacceptable and will trigger COM-027 Invalid character.
   Note: Listed below are the special cases in which non-ASCII characters are considered acceptable:
   a. Comment.
   b. Arguments of MACRO functions that are of string type.

16 **Block Format**

The writing format of the block (one line) is instructed as follows:

/	N	G	X	Y	Z	A	B	C	I	J	K	F	S	T	D	M

/	Block selective jump function. C41 in PLC must coordinate as well; does not support "function syntax" and "variable calculation."
N	The block sequence No., which must be written at the head of a block, and the MACRO command or variable designation cannot be written in the same line.
G	Function specification code, need to be written after N code
X	The X axis movement command, or the expansion of the G code, must be written after the G code.
Y	The Y axis movement command, or the expansion of the G code, must be written after the G code.
Z	The Z axis movement command, or the expansion of the G code, must be written after the G code.
A	The A axis movement command, or the expansion of the G code, must be written after the G code.
B	The B axis movement command, or the expansion of the G code, must be written after the G code.
C	The C axis movement command, or the expansion of the G code, must be written after the G code.
I	The radius command in the X direction or the argument of the expansion G code, must be written after the G code.
J	The radius command in the Y direction or the argument of the expansion G code, must be written after the G code.
K	The radius command in the Z direction or the argument of the expansion G code, must be written after the G code.
F	Block feed rate, or the argument of expansion G code.

S	Spindle rotation speed, or the argument of expansion G code.
T	Tool selection function, or the argument of expansion G code.
D	Tool compensation function, or the argument of expansion G code.
M	Auxiliary, or the argument of expansion G code.

Core interpretation processing order (1. first ~10. last):

1. Modal G code(G15、 G17、 G70…), expansion G code macro(G73、G84…)
2. M code macro, T code macro
3. S code
4. F code
5. H code
6. D code
7. T code
8. M code
9. B code
10. Interpolated G code（G0、G1…）, functional G code（G4、G51、G68…）

Note:

1. The format not mentioned above is introduced by the relevant G code in the form of an argument.
2. Generally, the "GETARG" function is used in the sub-program to read the argument. The rules, instructing the form of argument in the main program (parent program), are as follows:
   a. For parameter D, E, H, I, J, K, L, M, P, Q, R, T, argument should be attached directly, such as "G101 X30.Y40. D50. ;". If parameter is followed by a symbol first then argument, such as "G101 X30. Y40. D1=50. ;", there is an alarm.
   b. For parameter A, B, C, F, S, U, V, W, X, Y, Z, in addition to attaching argument directly, user can also attach a number before the argument, for example "G101 X30. Y40 . Z1=50. ;"
   c. Right after the above action instructions, only the value or the variable stored as numeric can be used. Otherwise, the system error may be caused by the coding limitation from program interpretation. This misuse is not under the range of protection.

17 **Operator**

Operator	Sign	Operating order
Brackets	( ) [ ]	1
Function assignment	Identifier	2
Negative	-	3
Complement	NOT	3
Multiplication sign	*	4
Divisor	/	4
Modulus (remainder)	MOD	4
Plus	+	5
Minus	-	5
Comparison	<,>,<=,>=	6
Equal	=	7
Not equal	<>	8
Boolean "and"	&,AND	9
Boolean "Mutually exclusive"	XOR	10
Boolean "or"	OR	11

Note 1:

When using the "/" component (division), be aware that if the numerator and denominator are integers, the result is still an integer. The difference between an integer and a non-integer result is whether user adds the decimal point or not.

example:

- The numerator is a non-integer: 1. / 2 = 0.5
- The denominator is a non-integer: 1 / 2.0 = 0.5
- The numerator and denominator are integers: 1/2 = 0

- The number in the bracket is an integer: (1/2)*1.0 = 0

Note 2:

The MOD operator (modulo) is only applicable to the numeric type "Long". If the numeric type is "Double", the following alarm message shows up.

example:

%@MACRO

 @1:= 4. MOD 3;

 M99;

18 **Language Instructions**

18.1 Variable Designation

Variable Designation	
Syntax	<**variable**> := <**description**>;
Explanation	Designate variable content

Example 1:
Direct setting
	@1 := 123;
	#1 := 456;
	#10 := "12"; // The **local variable** #10 content is 12
	@10 := "12"; // **public variable** @10 content is 12849

Example 2:
Indirect setting
	#1:= 123;
	@[#1] := 567; // @123=567
	@[#1+7]:=890; // @130=890

Remarks	1. The "12" in the first example is a string, indicating that the string is stored in the variable. When the public variable is stored, the controller will translate the string into ASCII. For the local variable, the translation will not be executed. 2. To correctly read the contents of the string stored in the public variable, use the **SCANTEXT function**. 3. In the example 2, please notice that it's the "square bracket"

18.2 GOTO

GOTO	
Syntax	**GOTO n**;
Explanation	The use of GOTO should be paired up with block sequence code(n). CNC would jump to the specified N-number and execute from that line. If there are two same N-numbers in the program, the first N-line number in the program will take precedence than the second one.

Examples	%@MACRO
	
	#1 := 1;
	
	#2 := 10;
	
	IF(#1 = 1) THEN
	
	  GOTO #2;
	
	END_IF;
	
	IF(#1 = 2) THEN
	
	  GOTO 100;
	
	END_IF;
	
	N10;
	
	G01 G90 X50. Y0. F1000;
	
	M30;
	
	N100;
	
	G01 G90 X0. Y50. F1000;
	
	M30;

Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash. It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

18.3 CASE

CASE	
Syntax	**CASE** <condition variable> **OF** <**variable**>: <statement List> <**variable**>, <**variable**>: <statement list> <**variable**>, <**variable**>, <**variable**>: <statement list> **ELSE** <statement list> **END_CASE**;

Explanation	Multi-conditional judgment. CNC, according to the condition variable, executes different program blocks. Please note that the "**variable**" content must be an **integer** which is greater than or equal to zero.

Examples	%@MACRO
	#1 := 1;
	
	G01 G90 X0. Y0. F1000;
	
	CASE #1 OF
	
	1:
	
	    X(1.0*#1) Y(1.0*#1);
	
	2:
	
	    X(2.0*#1) Y(2.0*#1);
	
	3, 4, 5:
	
	    X(3.0*#1) Y(3.0*#1);
	
	ELSE
	
	    X(4.0*#1) Y(4.0*#1);
	
	END_CASE;
	
	M30;

18.4 IF

IF	

Syntax	**IF** \<condition> **THEN**
	\<statement list>
	ELSEIF \<condition> **THEN**
	 \<statement list>
	ELSE
	\<statement list>
	END_IF;

Explanation	IF condition judgment

Examples	%@MACRO
	#1 := 3.0;
	G01 G90 X0. Y0. F1000;
	IF #1 = 1 THEN
	X(1.0*#1) Y(1.0*#1);
	ELSEIF #1 = 2 THEN
	X(2.0*#1) Y(2.0*#1);
	ELSEIF #1 = 3 THEN
	X(3.0*#1) Y(3.0*#1);
	ELSE
	X(4.0*#1) Y(4.0*#1);
	END_IF;
	M30;

18.5 Repeat

REPEAT	
Syntax	**REPEAT**
	<statement List>
	UNTIL <condition>
	END_REPEAT;
Explanation	REPEAT loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	REPEAT
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	UNTIL (#14 > #12) OR (#15 > #13) END_REPEAT;
	M30;
Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

18.6 While

WHILE	
Syntax	**WHILE** <condition> **DO**
	<statement list>
	END_WHILE;
Explanation	WHILE loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	WHILE (#14 <= #12) AND (#15 <= #13) DO
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_WHILE;
	M30;
Remarks	When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.

18.7 For

FOR	

Syntax	**FOR** <variable 1> := <Description 1> **TO** <Description 2> **BY** <Description 3> **DO**
	<statement list>
	END_FOR;
	Variable 1: The variable that controls the number of loops
	Description 1: The start number of the loop count, which can be a numerical value or an arithmetic expression
	Description 2: The terminated number of the loop count, which can be a numerical value or an arithmetic expression.
	Description 3: The added number to the current loop count after each loop, which can be a numerical value or an arithmetic expression.
	Statement list: execution in each loop
Explanation	FOR loop control
Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	G01 G90 X#12 Y#13 F1000;
	FOR #6 := 0 TO 3 BY 1.0 DO
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15); G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_FOR;
	M30;

Remarks	1. When using the loop function such as REPEAT/WHILE/FOR/GOTO, user should pay attention to the problem of infinite loop. When this occurs, the human machine interface, which is screen, may be locked or the machining program may crash.
	2. It is recommended to add the **SLEEP() function** avoiding the crash resulting from the infinite loop. With SLEEP() function, there is still chance to operate the human-machine interface to stop the program execution.
	3. Do NOT use the command whitch will jump out and in FOR loop (e.g: Complex Canned Cycle (G72-G78), using GOTO jump out loop and jump in again)、 M98 H_, it will cause the incorrect added number(<Description 3>). example:

```
// FOR #10 will add 5 in each loop cycle
% @MACRO   FOR # 10 := 1   TO   100   BY   1   DO
 GOTO   12 ;
 N13;
END_FOR;
M30;

N12;
M00;
@1 := @1 + 5 ;
GOTO   13 ;
M99;
```

18.8 EXIT

EXIT	
Syntax	EXIT
Explanation	Interrupt loop, jump out of loop control

Examples	%@MACRO
	#10 := 30.;
	#11 := 22.5.;
	#12 := #10/2;
	#13 := #11/2;
	#14 := 2.0;
	#15 := 1.5;
	#16 := 1.0;
	G01 G90 X#12 Y#13 F1000;
	FOR #6 := 0 TO 3 BY 1.0 DO
	IF((#14 = 4) & (#16 = 1)) THEN
	EXIT;
	END_IF;
	G00 X(#12+#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13-#15);
	G01 X(#12-#14) Y(#13+#15);
	G01 X(#12+#14) Y(#13+#15);
	#14 := #14 + 2.0;
	#15 := #15 + 1.5;
	END_FOR;
	M30;

18.9 Program Annotation

Program Annotation	
Syntax	(* <statement list> *)
	// <statement list>
Ex	Program Annotation(comment)
Example 1 **Single line annotation**	%@MACRO
	G00 G90 X0. Y0.; // homing
	M30;

Example 2 **Block annotation**	%@MACRO (* This block is the annotation area Regardless of the content, it does not affect program execution. *) G00 G90 X0. Y0.; G00 G90 X10. Y0.; G00 G90 X10. Y10.; G00 G90 X0. Y10.; G00 G90 X0. Y0.; M30;
Remark	If a text that is an annotation is added to the statement list, system error may occur due to the limitation while interpreting the code. This misuse is not under the protection of the controller.

18.10 Area of Execution Program

Area of Execution Program	
Syntax	% Execution Program %
Explanation	1. While there is "%...%" in the program, the execution program between two % will be executed by CNC. For those program prior to the first % and after the second % will not be executed.
Example 1	G91 G00 X10. % G91 G00 Y10. % G91 G00 Z10. M30 //After "Cycle Start", only Y coordinate move to "10.".

Example 2	G91 G00 X10. % G91 G00 Y10. G91 G00 Z10. M30 //After "Cycle Start", Y and Z coordinate move to "10." and X stay unmoved.

19 **MACRO Read/Process Flow**

Icon	Explanation
☐	The following explains the movements of each line of the parent program (main program): (the program in the left box) • N1: Set the coordinate system to G54 and move in absolute mode G90 • N2: Call macro G0201 and get the content in argument X1 with GETARG function • #1：=GETARG(X1) After entering G0201, store the argument X1 into the local variable #1 • #10：=#1004 Back up state G90/G91 with #10 • G91 G00 Y#1 Y coordinate moves in increment of 10mm by G00 • G#10 Restore state G90/G91 • M99 Return to parent program • N3: Since the last interpolation mode before leaving G0201 is G00, in this block, the X coordinate still moves by G00 • N4: Call the macro G0202 and the value of argument X will be stored in #24 • #1：=STD(#24,#1600) After entering G0202, store the argument X into the local variable #1 • #10：=#1004 Back up state G90/G91 with #10 • G91 G00 Y#1 Y axis moves in increment of -10mm by G00 • "G#10" Restore state G90/G91 • " #1000：=202" Set the interpolation mode to 202 • M99 Return to the parent program • N5: Since the interpolation mode is stored as 202 before leaving G0202, the system will call G0202 again when this block is executed. • N6: End of program

20 **MACRO Writing Note**

- It is recommended to use local variables (Local Variables, #1 ~ #400) in one MACRO and to use global variable when user need to pass value between MACROs (Global Variables, @1 ~ @165535).
- When executing MACRO, the user's data is passed by arguments (A_, B_, ..., Z_, X1=, Y1=, ...), and the arguments are connected to the local variables. The following table relates the arguments and the local variables.
- For expansion argument address, such as X1, use the function "GETARG" to read the values.

Argument	Corresponded Variable	Argument	Corresponded Variable	Argument	Corresponded Variable
A	#1	J	#5	T	#20
B	#2	K	#6	U	#21
C	#3	L	#12	V	#22
D	#7	M	#13	W	#23
E	#8	P	#16	X	#24
F	#9	Q	#17	Y	#25
H	#11	R	#18	Z	#26
I	#4	S	#19	X1	GETARG(X1)

- Modal Variables (#2001 ~ #2100, #3001 ~ #3100) will return to the VACANT state when the system is reset, so it can be applied to the timing of data exchange between multiple MACROs to save the use of variable resources.
- If a default initial value is required for MACRO, Customer Parameter is recommneded(#4001 ~ #4100, #5001 ~#5100).
- When the MACRO sub-routine (sub-program) is executed, if the mode G code is changed (G91/G90, G40/G41/G42, ..., etc.), please back up the current state, and restore the original mode G state before leaving the MACRO.
- If user want to keep the current MACRO interpolation mode (#1000) after leaving MACRO, it is recommended to designate #1000 as the MACRO number before leaving MACRO. As long as there is single block of the axial displacement, the system will automatically call this MACRO without specifying it again.
  - The interpolation mode will be automatically rewritten when G00/G01/G02/G03/G31/G33 show up or #1000 change.
- For length or angle arguments, use the STD function to normalize the unit before operation to match the usage habits of machine tool.
- Change to the setting of coordinate system is strictly forbidden, such as G92/G54/G52 which are relevant to coordinate system. Otherwise, the simulation would be useless.
- When machining, the core will pre-interpret the MACRO content, so the MACRO progress is ahead of the practical G/M code. If the variable specification or the data reading needs to synchronize with the G/M code,

please add WAIT function before the variable specification or the data reading to ensure the movement is correct.
- The MACRO program must be added with "M99;" to return to the main program (parent program).
- Develop good habits, add more comments to the program, this will increase program readability, and help subsequent maintenance and troubleshooting.

20.1 Registered G Code MACRO

- Developers could, according to the needs, add G code macros other than the standard G code, and could also customize the standard G code.
- Use the [Pr3701~Pr3710 *G code Macro call registry] setting to register the standard G code user want to customize. When the corresponding G code is executed in the program, the customized G code macro will execute instead of the standard G code.
- The following table introduces the setting value of [Pr3701~Pr3710 *G code Macro call registry] and the customizable standard G code.

Pr3701	Standard G Code	File Name of G Code Macro
0	none	none
-1	G00	G0000
1	G01	G0001
2	G02	G0002
3	G03	G0003
4	G53	G0053
5	G40	G0040
6	G41	G0041
7	G42	G0042
8	G43	G0043
9	G44	G0044
10	G49	G0049
11	G04	G0004

- The following are the operating specifications for the G code macro

- Macro feature is treated as G code macro feature
- All G codes in the registered G code macro will be executed as the standard G codes.
- Inheritance property of interpolation G codes such as G00, G01, G02, G03 still persists after these codes are registered as customized G codes.
    - Example:
    G00 X100.
    Y100.
    Where Y100. will also execute the G00 macro, and can read the occupied Y argument.
- If the user changes the interpolation mode in the customized G code macro, the interpolation mode (#1000) should be restored before leaving the customized G code macro.
    - For example, the user sets G00 as the registered G code macro.
    In the macro G0000, if the user changes the interpolation mode to G01, the user needs to restore the interpolation mode to G00, #1000 should be 0 before leaving G0000, in order to avoid the state disorder.
- The login G code macro will not work when it encounters the following instructions.
    - Lathe G7.1
    - Lathe G12.1
    - Lathe, A
    - Lathe, R
    - Lathe, C
    - Lathe All machining cycle instructions
    - Mill All machining cycle instructions
    - T code macro

- Version revision

Version	Revision
~Before	The interpretation order of the registered G codes is same as the G code macro.
10.114.50	- The interpretation order of the registered G codes is same as the standard G code. - Do not support changing the interpolation mode to 900000 in G code macro(#1000 := 900000).
10.116.16A、10.116.17	G53 is able to be replaced by customized MACRO (G0053)
10.118.22F、10.118.26	G40, G41, G42 is able to be replaced by customized MACRO (G0040, G0041, G0042)
10.118.45	G43, G44, G49 is able to be replaced by customized MACRO (G0043, G0044, G0049)
10.118.52D, 10.118.54	G04 is able to be replaced by customized MACRO (G0004)

21 **Function List**

Function	Explanation

ABS	Get the absolute value
	
	Example:
	
	#10 := -1.1;
	#1 := ABS(#10); // #1 = 1.1
	#2 := ABS(-1.2); // #2 = 1.2

ACOS	Calculate the acos of a value
	
	Example:
	
	#10 := 1;
	#1 := ACOS(#10); // #1 = 0
	#2 := ACOS(-1); // #2 = 180

ALARM	Call macro alarm
	
	Example:
	
	ALARM(300); // 觸發巨集第300號警報
	ALARM(301, "ALARM 301 Content");
	
	Remark: There is limit of string length in an alarm. For Mandarin, it's 19 words；For English, it's 39 alphabets.

ASIN	Calculate the asin of a value
	
	Example:
	
	#10 := 1;
	#1 := ASIN(#10); // #1 = 90
	#2 := ASIN(-1); // #2 = -90

ATAN	Calculate the atan of a value. The calculation result is between ±90°.
	
	Example:
	
	#10 := 1;
	#1 := ATAN(#10); // #1 = 45
	#2 := ATAN(-1); // #2 = -45

Function	Explanation

ATAN2(Y, X)	Calculate the four-quadrant atan value of Y/X. The calculation result is between ±180°.
	
	Example:
	
	    #10 := 1;
	    #20 := -1
	    #1 := ATAN2(#10, #20); // #1 = 135
	    #2 := ATAN2(#20, #10); // #2 = -45
	    #3 := ATAN2(1, 0); // #3 = 90
	
	Notes:
	
	1. Valid version: 10.118.29W, 10.118.40C, 10.118.42
	2. The argument X and Y must be numbers, or the alarm COR-023【Semantic error】will be issued.
	3. The argument X and Y can not be zero at the same time, or the alarm COR-004【Operation domain error】will be issued.
	
	Example of wrong cases:
	
	    @1 := ATAN2("1", 1); // The first argument is not a number, issue COR-023 alarm.
	    @2 := ATAN2(0, 0); // The arguments are both zero, issue COR-004 alarm.

AXID	Inquire the axis number corresponding to the axis name. If the axis name does not exist, the return value is blank (VACANT, #0)
	
	Example:
	
	    Suppose the sixth axis name is Y2 (Pr326=202) and the second axis name is Y (Pr322=200).
	
	    #1 := AXID(Y); // #1 = 2
	
	    #2 := AXID(Y2); // #2 = 6

CEIL	Return the smallest integer greater than or equal to a certain value
	
	Example:
	
	    #10 := 1.4;
	
	    #1 := CEIL(#10); // #1 = 2
	
	    #2 := CEIL(1.5); // #2 = 2

CLOSE	Close the file opened by the OPEN function, and the file will be automatically closed after the program ends. The PRINT function will fail if the file is already closed. Example: CLOSE(); // close the file

Function	Explanation

COS	Calculate the cosine of a value
	
	Example:
	
	#10 := 180;
	
	#1 := COS(#10); // #1 = -1
	
	#2 := COS(-180); // #2 = -1

DBLOAD	Read the data of specified index from the currently loaded XML. For related applications, please refer to the appendix.
	
	Example:
	
	DBOPEN("FLAT\\TAB01");
	
	// Load FLAT\\TAB01 data file
	
	DBLOAD(0);
	
	// read the 0th cycle
	
	DBLOAD(1);
	
	// read the 1st cycle
	
	Note: The file path to the XMLDB may be influenced by customized Action (CUSTOMFILE_CYCLE1~5), please refer to CE人机客制应用文件.

Function	Explanation
DBSAVE	Save the data of specified index from the currently loaded XML. For related applications, please refer to the appendix. EX： ```\n1 DBOPEN("GrinderToolTable.cyc"); //\n Load "GrinderToolTable.cyc"\n2\n3 DBLOAD(0); //\n Load the 0-th datum from cyc file\n4 DBSAVE(0); //\n Save the 0-th datum to cyc file\n``` Notes： 1. The file path for XMLDB can be changed by Customized Actions (CUSTOMFILE_CYCLE1~5). Fore more info, refer to CE人机客制应用文件-开放使用的Action列表 。 2. Opening (DBOPEN) and loading (DBLOAD) must be done before saving (DBSAVE). If the user uses DBSAVE without using DBPEN and DBLOAD beforehand, the saving process will NOT be executed. 3. Even if the Field Value is the same as the default value defined in the Schema, the data will still be written into the CYC data file. 4. Versions: 10.118.39 and later.

DBOPEN	Load the specified XML. The XML should be in the GNCFILES specified by the user. For related applications, please refer to the appendix.
	
	Example:
	
	DBOPEN("Test.cyc");
	
	// Load GNCFILES\\Test.cyc
	
	Example:
	
	#1 = 51;
	
	DBOPEN("FLAT\\\\\\AB#1[3]ZZ.cyc");
	
	 // Load FLAT\\\\\AB051ZZ.cyc, [3] indicates that the file name is in three valid digits
	
	Note1: The file path to the XMLDB may be influenced by customized action (CUSTOMFILE_CYCLE1~5), please refer to CE人机客制应用文件.
	
	Note2: For the requirement of reloading file, please reset system (or run BGnd Stop in Background Running Objects) before re-execute DBOPEN.

Function	Explanation
DRAWHOLE	Draw a circle based on the tool radius and the color defined by the SETDRAW function at the current position (only valid in the simulation, system will not add a circle in the actual path)
EXP	Calculate the exponential value with natural number as the base Example: #1:=EXP(1); // e^1 = 2.71828 Valid version: 10.116.16

FLOOR	Return the largest integer less than or equal to a certain value
	
	Example:
	
	    #10 := 1.4;
	
	    #1 := FLOOR(#10); // #1 = 1
	
	    #2 := FLOOR(1.5); // #2 = 1

GETARG	Read the arguments passed by the caller
	
	Example:
	
	    Assume that the main program content of O0001 is
	
	    G101 X30. Y40. Z1=40. Z2=50.;
	
	    The G0101 expansion macro program uses GETARG to read the argument content
	
	    #1 := GETARG(X); // Save the X argument 30. to #1
	
	    #2 := GETARG(Z1); // Save the Z1 argument 40., to #2
	
	    #3 := GETARG(W);
	
	    // Since W does not exist, #3 is (VACANT, #0)

Function	Explanation

GETTRAPARG	Read the argument content in Trap Block. Trap Block is the blocks between G66/G66.1 and G67.
	
	Example:
	
	Assume that the main program content of O0001 is
	
	G66 P100 X100. Y100. // P100 means call sub-program O0100.
	
	G01 X20. // O0100 sub-program uses GETTRAPARG to read the argument content
	
	#1 := GETARG(X);
	
	// Save the X argument 100. to #1
	
	#2 := GETTRAPARG(X);
	
	// Save the X argument in the Trap block, 20. to #2
	
	Please refer to G66/G67：Call Modal Macro.

LN	Calculate the logarithm value with natural number as the base. Example: #2:=LN(100); // ln100 = 4.60517 Note: The argument has to be positive. Otherwise, the alarm shows up. Valid version: 10.116.16

MAX	Get the maximum value of two input values
	
	Example:
	
	#10 := 1.2;
	
	#20 := 4.5;
	
	#1 := MAX(#10, #20); // #1 = 4.5
	
	#2 := MAX(-1.2, -4.5); // #2 = -1.2

MIN	Get the minimum value of two input values
	
	Example:
	
	#10 := 1.2;
	
	#20 := 4.5;
	
	#1 := MIN(#10, #20); // #1 = 1.2
	
	#2 := MIN(-1.2, -4.5); // #2 = -4.5

Function	Explanation
MSG	Customize the message, please refer to the "MACRO Customized Message" for details.
	Example:
	MSG(100); // message ID
	MSG("bit lost"); // display message content
	MSG(100, "bit lost"); // hint ID + displaye message content
	Remark: There is limit of string length in a message. For Mandarin, it's 19 words；For English, it's 39 alphabets.

Function	Explanation

OPEN ("file name") or
 	
OPEN ("file name", "writing mode")
	Open a text file name of which user specify, which is in the NcFiles folder (Folder path please refer the Pr3219). The PRINT function is valid only after the file is opened.
	
	If the file name is "COM", it means that the port RS232/RS485 is turned on, and its setting is determined by Pr3905.
	
	Example:
	
	      OPEN("COM"); // Open port RS232
	
	      PRINT("\\p"); // Output '%' character
	
	      FOR #1 = 1 TO 5000 DO
	
	          #30 := #1 * 10.;
	
	          PRINT("G01 X#30"); // Output G01 X10.0
	
	      END_FOR;
	
	      PRINT("\\p"); // Output '%' character
	
	      CLOSE(); // Close port
	
	The "writing mode" determines, when the file is opened, whether the original file content is retained or cleared. (valid version: 10.116.36I)
	
	(i) "a": Keep the original text and the text newly-output follows the original one.
	
	Example:
	
	      OPEN("PROBE.NC", "a");
	
	      // Open PROBE.NC and keep the text, and be ready for text output
	
	(ii) "w"/nothing: Clear the original text and output the new text in the file.
	
	Example:
	
	      OPEN("PROBE.NC");
	
	      // Open PROBE.NC and clear the text, and be ready for text output
	
	      OPEN("PROBE.NC", "w");
	
	      // Open PROBE.NC and clear the text, and be ready for text output
	
	(iii) Wrong writing mode: system issues alarm, COR-301 OPEN command format error.
	
	Example:
	
	      OPEN("PROBE.NC", "abc");
	
	      // Wrong writing mode, issues alarm, COR-301, so PROBE.NC is not opened for text output.
	
	(iv) Convert # or @ variable into a string, and the decimal digits is determined by Pr17 (valid version: 10.118.12C)

Function	Explanation
	(v) Convert # or @ variable into a string with [*] in the end, and the decimal digits is determined by this variable.

PARAM	Read the contents of system parameters
	
	Example:
	
	    #1 := PARAM(3204);
	
	    // Read the contents of Pr3204 (PLC scan time)

POP	Taking the data in STACK from top layer to bottom layer in sequence. User must pay attention to the total data in the stack. If there are 5 data, the maximum times to use POP is 5.
	
	Example:
	
	    PUSH(5); // Insert the number 5 into the stack
	
	    #1 := POP(); // Remove the topmost value in the stack (#1 = 5)

POW	Calculate the power of specified base Example: #3:=POW(16,0.5); // 16^0.5 = 4 Note: The base cannot be negative. Otherwise there will be alarm COR-122. Valid version: 10.116.16

Function	Explanation

PRINT	This function is used to output a string, and the variable in the output string will be replaced by the content of it.
	
	The character "\" is an escape character, and the related special characters are defined as follows:
	
	"\\": indicates "\" character
	
	"\@": indicates "@" character
	
	"\#": indicates "#" character
	
	"\p": indicates"%" character
	
	Convert # or @ variable into a string, and the decimal digits is determined by Pr17 (valid version: 10.118.12C)
	
	Convert # or @ variable into a string with [*] in the end, and the decimal digits is determined by this variable.
	
	Example:
	
	    Assume that Pr17=2 in metric unit
	
	    @53 = 20 ;
	
	    #3 = 23.1234 ;
	
	    PRINT("G01 X#3 Y@53 Z20.0") ;
	
	    The output is G01 X23.123 Y20 Z20.0 ; // Display to thousandths place
	
	Example:
	
	    @53 = 20 ;
	
	    #3 = 23.1234 ;
	
	    PRINT("G01 X#3[2] Y@53 Z20.0") ; // #3[2] means display to hundredths place
	
	    The output is G01 X23.12 Y20 Z20.0 ; // Display to hundredths place

PUSH	Stuff data into the STACK, the data PUSH into the controller first will be stacked on the bottom layer, and the last data on the top one. Example: PUSH(#1); // Put variable #1 into the STACK
RANDOM	Generate a random number Example: #1 := RANDOM();

Function	Explanation

READDI
 	
(I point number)
 	
READDO
 	
(O point number)
	The value of variable derives from the I/O point number in the parentheses READDI/READDO.
	
	Example:
	
	@52 := READDI(31); // Read the value of I31 and put it in @52
	
	#88 := READDO(11); // Read the value of O11 and put it in #88
	
	G90 G10 L1000 P4000 R READDI(15); // Read the value of I15 and put it in R4000
	
	Notes:
	
	1. Valid version: 10.116.23
	2. The I/O point is read during pre-interpreting, but it is processed when READDI / READDO is executed in order to avoid the error resulting from pre-interpreting I/O point. Because system await until READDI / READDO is executed, machine will decelerate to zero.
	3. The range of I/O point number is 0~511. If the number is out of the range, system issues alarm, COR-138 Read/write command format error at the I/O/ A point.

READABIT
 	
 (A point number)
	The value of variable derives from the A point number in the parentheses of READBIT.
	
	Example:
	
	@52 := READABIT(31); // Read the value of A31 and put it in @52
	
	#88 := READABIT(11); // Read the value of A11 and put it in #88
	
	Notes:
	
	1. Valid version: 10.116.44
	2. The A point is read during pre-interpreting, but it is processed when READBIT is executed in order to avoid the error resulting from pre-interpreting A point. Because system await until READBIT is executed, machine will decelerate to zero.
	3. The range of A point number is 0~511. If the number is out of the range, system issues alarm, COR-138 Read/write command format error at the I/O/ A point.

Function	Explanation
READRREGBIT (Register number, specified Bit)	The value of variable derives from the register number and specified bit in the parenthesis of the READRREGBIT. Example: @52 := READRREGBIT(31,3); // Read the value of the third bit of R31 and put it in @52 Notes: 1. Valid version: 10.116.39 2. The register is read during pre-interpreting, but it is processed when READRREGBIT is executed in order to avoid the error resulting from pre-interpreting Register. Because system await until READBIT is executed, machine will decelerate to zero. 3. If register is less than 0 or greater than 65535, the system issues alarm, COR-135 Read/write command format error for R value. 4. If register is an incorrect character, system issues alarms, COR-5 Program loading failure and COM-8 absent statement ending character ';'. 5. If specified bit is less than 0 or greater than 31, system issues alarm, COR-135 Read/write command format error for R value. 6. If specified bit or both of register and specified bit is incorrect characters, system issues alarm, COR-5 Program loading failure and COM-9 wrong assignment character ':='.

ROUND	Return a rounding value.
	
	Example:
	
	#10 := 1.4;
	
	#1 := ROUND(#10); // #1 = 1
	
	#2 := ROUND(1.5); // #2 = 2

Function	Explanation
SCANTEXT	This function is used to read the contents of the string stored in global variable. When the string is stored in global variable, the controller translate it into ASCII first and save. User get wrong string if they output the value directly. To get the correct string, please make good use of this function. Example: %@MACRO @1:="12"; // 16 carry HEX=3231, 10 carry DEC=12849 #1:=SCANTEXT(1); OPEN("NC"); PRINT("@1"); PRINT("#1"); CLOSE(); M30; The result is @1 = 12849 #1 = 12

SETDO
(O point number,
 O point on or off)
	Determine O point number and the state (1: On, 0: Off) with 2 numbers in the parenthesis of SETDO.
	
	Example:
	
	SETDO(3, 1); // Set O3 on
	
	SETDO(8, 0); // Set O8 off
	
	Notes:
	
	1. Valid version: 10.116.23
	2. The writing of point O is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETDO should be avoided. For example, O1 is on by SETDO in MACRO, but off in PLC. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. The range of O point number is limited to 0~511. If the range is wrong, the system issues alarm, COR-138 Read/write command format error at the I/O/A point.

Function	Explanation

SETABIT
 	
(point A, point A on or off)
	Determine A point number and the state (1: On, 0: Off) with 2 numbers in the parenthesis of SETDO.
	
	Example:
	
	SETABIT(3, 1); // Set A3 on
	
	SETABIT(8, 0); // Set A8 off
	
	Notes:
	
	1. Valid version: 10.116.44
	2. The writing of point A is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETABIT should be avoided. For example, A1 is on by SETABIT in MACRO, but off in PLC. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. The range of A point number is limited to 0~511. If the range is wrong, the system issues alarm, COR-138 Read/write command format error at the I/O/A point.

SETRREGBIT
 	
(Register number, Bit number, on or off)
	Determine Register number, Bit number, and the state (1: On, 0: Off) with the 3 digits in the parenthesis of SETRREGBIT.
	
	Example:
	
	SETRREGBIT(50,3,1); // Set the third R50 Bit on
	
	SETRREGBIT(50,4,0); // Set the fourth R50 Bit off
	
	Notes:
	
	1. Valid version: 10.116.39
	2. The writing of Register is in the interpolation stage, so it is not necessary to decelerate to 0 during execution. However, in the MACRO processing in pre-interpreting stage, the developer should decide whether to use WAIT, which makes machine decelerate to 0.
	3. Mixed use of PLC and SETRREGBIT should be avoided. For example, first Bit of R50 is on by SETRREGBIT in MACRO, but is off in PLC off. Even though the previous order is overridden by the next one, it is common to confuse while using both, so it is recommended that use one of them at a time.
	4. If Register number is less than 0 or greater than 65535, the system issue alarm, COR-135 Read/write command format error for R value.
	5. If Bit number is less than 0 or greater than 31, system issues alarm, COR-135 Read/write command format error for R value.
	6. If the state is not 0 (off) or 1 (on), system issues alarm, COR-135 Read/write command format error for R value.
	7. If any argument is incorrect character, system issues alarms, COR-5 Program loading failure and COM-3 Syntax error.

Function	Explanation
SETDRAW (path color) or SETDRAW (path color, filled color, tool radius)	Define the drawing style of simulation: 1. Path Color: set the color of the outline, which can be set by BGR code or by color code in "Simu. Setting." 2. Fill color: set the color filled in circle drawn by DRAWHOLE, which can be set by BGR code or by color code in "Simu. Setting." 3. Tool radius: set the radius of circle drawn by DRAWHOLE, G code with tool radius compensation, such as G01, is influenced by this as well. Common BGR codes are as follows: Color Setting:

Color Code	BGR codes	Color Code	BGR codes
0	0	8	8421504
1	8388608	9	16711680
2	32768	10	65280
3	8421376	11	16776960
4	128	12	255
5	8388736	13	16711935
6	32896	14	65535
7	12632256	15	16777215

Note:

- BGR code is the same as RGB code, but with different endianness for B(Blue) and R(Red) value.

e.g.

Red color in RGB code is 0xFF0000. Converting it into BGR code will be 0x0000FF = 255. (as described in color code 12)

- SETDRAW sets path color and filled color at the same time. If user would like to make path color and filled color different, remember to change the path color with SETDRAW after DRAWHOLE is executed.

Function	Explanation
	e.g. %@MACRO #3:=SETDRAW(#1,#2,#18); // #3 records the original path color, #2 defines the filled color, #18 defines the circle radius DRAWHOLE(); SETDRAW(#3); // Change the path color after drawing the circle M99;

SIGN	Return the sign of a value, the negative number is -1, the positive number is 1, and 0 is 0.
	
	Example:
	
	#10 := 4;
	
	#1 := SIGN(#10); // #1 = 1
	
	#2 := SIGN(-4); // #2 = -1
	
	#3 := SIGN(0); // #3 = 0

SIN	Calculate the sine of a value
	
	Example:
	
	#10 := 90;
	
	#1 := SIN(#10); // #1 = 1
	
	#2 := SIN(-90); // #2 = -1

SLEEP	Temporarily abandon the execution right of this macro loop, generally used in conjunction with the loop (FOR, WHILE..., etc.) to avoid entering the infinite loop, which causes the human-machine to crash. Example: SLEEP();

SQRT	Calculate the square root value of a value
	
	Example:
	
	#10 := 4;
	
	#1 := SQRT(#10); // #1 = 2
	
	#2 := SQRT(9); // #2 = 3

Function	Explanation

STD (argument1, argument 2)	According to Pr17, the value is converted into the input unit (Input Unit, IU) set by the system at that time.
	
	1. The argument 1 is the value the unit of which is about to be changed.
	2. The argument 2 is a standard unit. Generally, it is #1600, and value of #1600 is from Pr17.
	
	Metric Unit:
	
	Example 1:
	
	    When Pr17=2, #1600 corresponds to LIU = 0.001mm
	
	    #9 := 100;
	
	    #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.1mm (100*0.001)
	
	Example 2:
	
	    When Pr17=3, #1600 corresponds to LIU = 0.0001mm
	
	    #9 := 100;
	
	    #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.01mm (100*0.0001)
	
	Imperial:
	
	Example 3:
	
	    When Pr17=2, #1600 corresponds to LIU = 0.0001inch
	
	    #9 := 100;
	
	    #10 := STD(#9,#1600); // #9 is 100 BLU, so #10 is 0.01inch (100*0.0001)

STDAX (argument 1, argument 2)	Converts the value to the standard unit of the corresponding axis.
	
	The argument 1 is a variable, and the argument 2 is the name of the corresponding axis.
	
	Example:
	
	    #24 := STDAX(#24,X);
	
	    #3 := STDAX(#3,A);

Function	Explanation

STKTOP	Copy the data in the STACK.
	
	Example:
	
	    PUSH(5); //Put the number 5 into the stack
	
	    PUSH(6); //Put the number 6 into the stack
	
	    PUSH(7); //Put the number 7 into the stack
	
	    #1 := STKTOP[0] ; // #1 = 7
	
	    #2 := STKTOP[1]; // #2 = 6
	
	    #3 := STKTOP[2]; // #3 = 5

SYSVAR
 	
(Path identification code, system variable code)
	Read the system variable in specific Path.
	
	Path identification code: 1 is the first Path, 2 is the second Path, and so on.
	
	System variable code: system variable number will be read
	
	Example:
	
	    #1 := SYSVAR(1, 1000);
	
	    // Read the system variable #1000 in the first Path (interpolation mode)

TAN	Calculate the tangent of a value.
	
	Example:
	
	    #10 := 45;
	
	    #1 := TAN(#10); // #1 = 1
	
	    #2 := TAN(-45); // #2 = -1

WAIT	The system stops pre-interpreting until the instruction before WAIT is finished.
	
	Example:
	
	    %@MACRO
	
	    @50 := 1; // @50 equals to 1
	
	    G90 G01 X100. F1000; // Assume to system is Reset at this time
	
	    WAIT();
	
	    @50 := 0; // @50 equals to 0
	
	    M30;
	
	Assume that the system is reset when G01 is in execution. Since the block before WAIT is not finished, @50 equals to 1 after Reset.

Function	Explanation

CHKMN ("machinery code")	Check machinery code. 1: consistent, 0: does not match
	
	Example:
	
	%@MACRO
	
	#51 := CHKMN("5566"); //The value of #51 is the checking result .
	
	IF #51=0 THEN
	
	ALARM(501, "The manufacturer code is invalid."); //If machinery code does not match, system issues an alarm
	
	END_IF;
	
	Target version: 10.116.6A

CHKSN ("Serial No.")	Check Serial Number. 1: consistent, 0: does not match
	
	Example:
	
	%@MACRO
	
	#52 := CHKSN("M9A0001"); //The value of #52 is the checking result
	
	IF #52=0 THEN
	
	ALARM(502, "The serial number is invalid."); //If the serial number does not match, system issues the alarm
	
	END_IF;
	
	Target version: 10.116.6A

CHKMT ("Machine Type")	Check machine type. 1: consistent, 0: does not match
	
	Example:
	
	%@MACRO
	
	#53 := CHKMT("MILL"); //The value of #53 is the check result
	
	IF #53=0 THEN
	
	ALARM(503, "The machine type is invalid."); //If machine type does not match, system issues the alarm
	
	END_IF ;
	
	Target version: 10.116.6A

Function	Explanation

CHKMI ("Model")	Check the controller model, 1: consistent, 0: does not match
	
	For SUPER series, please input 'S'. For other models, please input value according to the actual model. For example, 10B =>10B, 11A, => 11A.
	
	Example:
	
	%@MACRO
	
	#54 := CHKMI("S"); //#54 is the checking result
	
	IF #54=0 THEN
	
	ALARM(504, "The hardware type is invalid."); //If the model does not match, system issues the alarm
	
	END_IF;
	
	Target version: 10.116.6A

Function	Explanation
CHKINF(category number, "code")	Check if the code corresponds to the category number. 1: consistent, 0: does not match

The range of category numbers is 1~5, each corresponding code is:

1. Machinery code
2. Serial No.
3. Machine Type
4. Model
5. Industrial machine ID

For SUPER series, please input 's'. For other models, please input value according to the actual model. For example, 10B =>10B, 11A, => 11A.

Example:

```
%@MACRO
#51 := CHKINF(1, "5566"); // #51 is the checking result
IF #51=0 THEN // If the machineny code does not match, system issues an alarm
ALARM(501, "The manufacturer code is invalid.");
END_IF;

#52 := CHKINF(2, "M9A0001"); // #52 is the checking result
IF #52=0 THEN // If the serial No. does not match, system issues the alarm
ALARM(502, "The serial number is invalid.");
END_IF;

#53 := CHKINF(3, "MILL"); // #53 is the checking[ result
IF #53=0 THEN // If the machine type does not match, system issue the alarm
ALARM(503, "The machine type is invalid.");
END_IF;

    #54 := CHKINF(4, "S"); // #54 is the checking result
    IF #54=0 THEN // If the model does not match, system issues the alarm
    ALARM(504, "The hardware type is invalid.");
    END_IF;

#55 := CHKINF(5, "10"); // #55 is the checking result
IF #55=0 THEN // If the Industrial machine ID does not match, system issues the alarm
ALARM(505, "The industrial machine ID is invalid.");
END_IF;
```

If argument is incorrect, or the category number is out of the range, system issues the alarm, COR-353 【Invalid argument of CHKINF】

Example:

Function	Explanation

	%@MACRO
	#51 := CHKINF(60, "Mill"); // category number is out of range
	#51 := CHKINF("2", "Mill"); // argument 1 is incorrect
	#53 := CHKINF(5, 12345); // argument 2 is incorrect
	
	Target version: 10.118.22M、 10.118.28B、 10.118.30

STR2INT("string")	Convert a numeric string into an integer
	
	Example 1:
	    %@MACRO
	    @1:="5555";
	    #1:= SCANTEXT(1); // #1 = String 5555
	    #2:= STR2INT("#1"); // #2 = 5555
	Example 2:
	    %@MACRO
	    #1:=STR2INT("100"); // #1 = 100
	Example 3:
	    %@MACRO
	    #1:=STR2INT("123.456"); // #1 = 123
	
	Note: As long as there is character or alphabet in the string, STR2INT is not able to work.

Function	Explanation

SYSDATA(system data number)	Read the system data number.
	
	EX：
	
	    // if want to get system data D336、D77
	    WAIT(); // stops pre-interpreting for getting newest value
	    #1 := SYSDATA(336); // Axis board data exchange time(D336)
	    #2 := SYSDATA(77); // Free hardware memory(D77)
	    OPEN("DbgData.txt", "a"); // Open the file which name is "DbgData.txt"
	    PRINT("#1 #2"); // print data to file
	    CLOSE(); // close the file
	
	Note:
	
	1. Valid version: 10.118.23U, 10.118.28H, 10.118.33.
	2. For getting newest value, it is recommended to use WAIT() function blocking the pre-interpretation before using SYSDATA().
	3. The type of argument must be integer. If the type of argument is incorrect, it will cause function to operate abnormally.
	4. if the argument number is out of range of system data number, system will issues the alarm COR-016 【Illegal variable access】.
	
	EX：
	
	    SYSDATA("77"); // the type of argument is string, issues the alarm COR-023
	    SYSDATA(77.0); // the type of argument must be integer , issues the alarm COR-023
	    SYSDATA(D77); // the argument is not a number, it cause syntax error, issue the allarm COM-008
	    SYSDATA(10000); // the argument number is out of range of system data number, issues the alarm COR-016

Function	Explanation

DRVDATA(station_number, status_variables_No(Dec))
 	
DRVDATA(station_number, "status_variables_No(Hex)")
	Read Syntec drive status variables.
	
	status variables No. has two input format.
	
	`Decimal:Convert variable numbers to decimal representation.`
	
	`Hexadecimal:use format "xxxh", x=0~F, fill the status variables number( 3 digit ) and end with 'h' or 'H'.`
	
	e.g：
	
	// assume want to get
	// speed command(Pn-D26) of first axis (station_number=1000)
	// Enc Internal Temperature(Pn-D61) of first spindle (station_number=1003)
	WAIT(); // To get newest value, block Pre-Interpretation
	#1 := DRVDATA(1000, 3366); // speed command of first axis (Pn-D26，D26 convert to decimal format is 3366)
	#2 := DRVDATA(1003, "D61h"); // Enc Internal Temperature of first spindle(Pn-D61，D61 convert to hexadecimal format is "D61h")
	OPEN("DbgData.txt", "a"); // open file DbgData.txt
	PRINT("Pn-D26: #1, Pn-D26: #2"); // print value
	CLOSE(); // close file
	
	The file DbgData.txt content may be below.
	
	Pn-D26: 150, Pn-D26: 430
	
	Note:
	
	1. Valid version: 10.118.23U, 10.118.28.I, 10.118.34.
	2. For getting newest value, it is recommended to use WAIT() function blocking the pre-interpretation before using DRVDATA().
	3. The execution time of each function is 0.1~0.2s.
	4. First argument must be integer, system will issues the alarm COR-023 【Semantic error】.
	5. If second argument is string type, must be hexadecimal format ("xxxh", x=0~F), system will issues the alarm COR-023 【Semantic error】.
	6. second argument must be string or integer value, system will issues the alarm COM-003 【Syntax error】.
	7. If either the drive or the controller does not support the specified status variable, the system will issue the alarm COR-016 【Illegal variable access】.
	8. Visit the "Controllor Axis Info." page to check status variable accessibility. Only those shown are accessibile.
	9. If the controller supports specified status variable and the drive does not support, 0 will be returned.
	10. If no drive corresponding to station number or using non Syntec M3 drive, return VACANT.

Function	Explanation

	11. Since it will take some time to read the status variable after system be powered on, it is recommended to call this function one minute after powering on, or it might cause system issue the alarm COR-016 【**Illegal variable access**】 due to the variable are not read finish.
	
	Example of wrong case：
	
	// alarm COM-3
	DRVDATA(1003, D61h); // second argument must be string or integer value
	
	// alarm COR-023
	DRVDATA("1003", 3425); // first argument must be integer
	DRVDATA(1003, "G21h"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	DRVDATA(1003, "3425"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	DRVDATA(1003, "0D61h"); // if second argument is string type, must be hexadecimal format ("xxxh", x=0~F)
	
	// alarm COR-016
	DRVDATA(1003, "DFFh"); // drive is not suupot this status No.
	
	// return VACANT
	DRVDATA(9999, "D61h"); // no drive corresponding to station number
	DRVDATA(9999, "DFFh"); // if no drive corresponding to station number, will not check the status variables No.

22 **Call sub-Program**

22.1 Calling Methods

Syntax	Explanation	Calling Type	Local Variable	Example
M98 P_ H_ L_	Call sub-program P_Sub-program Name H_Starting of block sequence No. L_Repeated Counts	Sub-program	Inherit the local variables #1~#400 from main/parent program	M98 P10 L2; Explanation: Call O0010 twice
M198 P_ H_ L_ (If M198 is not logged in Pr3601~)	Call sub-program P_Sub-program Name H_Starting of block sequence No. L_Repeated Counts	Sub-program	Inherit the local variables #1~#400 from main/parent program	M198 P10 L2; Explanation: Call O0010 twice
G65 P_L_	Call Single Macro P_Subroutine Name L_Repeated Counts	Macro	Create new local variables #1~#400, and #1~#26 records the corresponding argument in the calling block	G65 P10 Lw X10.0 Y10.0 Explanation: Call O0010 twice, and input argument

Syntax	Explanation	Calling Type	Local Variable	Example
G66 P_L_	Use movement instruction to call mode macro P_Subroutine Name L_Repeated Counts	Mode Macro	Create an independent section of #1~#400 each time G66 is called. Local variables in this section will be shared until executing G67. After executing G67, local variables in this section will be retrieved and cleared. Note: The local variables in the section are shared with sub-program called by P argument (G66 P) only. They are different from the local variables in the program where G66/G66.1 is called.	G66 P10 X10.0 Y10.0; **X20.** **Y20.** Explanation: Moving instructions X20. and Y20. call O0010, and input arguments X10.0, Y10.0.
G66.1 P_ L_	Each block calls mode macro P_Sub-program Name L_Repeated Counts	Mode Macro	The same as G66.	G66.1 P10 X10.0 **X20.** **G04 X2.** **M30** Explanation: Each block calls O0010 and input argument X10.0.
G_L_	Call expansion G Code. L_Repeated Counts	Macro	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	G128 L3 X1.0 Explanation: Call G0128 three times.
G_	Call customized G Code (G00, G01, G02, G03, G53, G40, G41, G42) Must login Pr3701~ before using.	Macro	Create a new section of local variables #1~#400 each calling, and restoring local variables in main program when the Macro is finished.	G01A_B_C; Explanation: Call customized G01.

Syntax	Explanation	Calling Type	Local Variable	Example
T_	Call sub-program T0000 to change tool. The T code in sub-program is general T code, which does not call T0000.	If Pr3215=1, then it is sub-program.	Inherit local variables #1~#400 from main program	T3; Explanation: Call T0000.
		If Pr3215=2, then it is macro.	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	
M_	Call M code Macro. The M code in the macro is general M code, which does not call M code macro again. Must login Pr3601~ before using.	Macro	Create a new section of local variables #1~#400 each calling, and restore local variables in main program when the Macro is finished.	M13A_B_C; Explanation: Call M0013 macro.

Notes:

- If L argument above isn't assigned, the default value is 1.
- The life cycle of local vaiables (#1~#400) in above form, please refer to Macro Variable Specification.

Example of Variable Life Cycle:

Local variable is shared if same color

Main Program
Test.NC
G01…
G01…
G65 P10 L2
….

M30

O0010 (First-time)
Local variable is independent from Test.NC

G01…
G03…
M98 P11 H50 L2

O0010 (Second-time)
Local variable is independent from Test.NC

O0011 (First-time)
Share local variable with O0010

G66 P12 L1
G01…
G02…
G67

O0011 (Second-time)
Keep Sharing local variable with O0010

G66 P12 L1
G01…
G02…
G67

O0012 for G01
Local variable is independent from O0011
…

O0012 for G02
Share local variable with G01 executed earlier
…

O0012 for G01
Local variable is independent from O0011
…

O0012 for G02
Share local variable with G01 executed earlier
…

22.2   Return Methods

Syntax	Explanation	Example
M99	Return to main program.	M99
M99 P_	Return to the specified block sequence number in main program P_: specified **block sequence number**.	M99 P100; Return to N100 in the main program.
M99 Q_	Return to the sepcified row number in main program P_: specified **row number**.	M99 Q100; Return to row number, like 100, in the main program.
G67	Cancel G66	G67;

23 **Variable Specification**

For the explanation of # and @ variable, please refer to Macro Variable Specification.

24 **MACRO Customized Alarm**

24.1 **MACRO Alarm Trigger Syntax**

%@MACRO

ALARM(xxx);// xxx is the Alarm number

M30;

24.2 **DOS System Alarm Content Editor Explanation**

- File Path:

=> Traditional Chinese: C:\\CNC\\EXE\\APPDATA.RES\\CNCCHI.STR

=> English Version: C:\\CNC\\EXE\\APPDATA.RES\\CNCENG.STR

=> Others: C:\\CNC\\EXE\\APPDATA.RES\\CNCLOC.STR

- Content Format: **24**xxx="**1;MSG**=**Alarm Content**", xxx is the Alarm number. Please choose an unused number as the customized alarm number, and please note the identification number is **24**.

- Example:

=>CNCCHI.STR:

24003="1;MSG=**max chordal length of arc should be smaller or equal to 0**"

-> CNCENGSTR:

24003="1;MSG= **max arc length can not be negative**"

24.3 **WinCE System Alarm Content Editor Explanation**

- File Path:

=>Chinese Version: DiskC\\OCRes\\CHT\\String\\AlarmMacro_CHT.Xml

=>English Version: DiskC\\OCRes\\Common\\String\\AlarmMacro_Com.Xml

=>General: DiskC\\OCRes**L**\\String\\AlarmMacro_**L**.Xml.

 L is the name of each language.

- File Format: <Message ID="AlarmMsg::**Macro**::ID=xxx" Content="**Alarm Content**" />.

xxx is Alarm number. Please choose an unused number as the customized alarm number.

Please note that the identification letter is **Macro**. Length of string in alarm content is 48 alphabets in English, or 31 characters in Chinese. Redundant string exceeds the alarm window.

- Example:

-> CusMacroAlarmMsg_CHT.Xml：

<Message ID="AlarmMsg::**Macro**::ID=3" Content="**max chordal length of arc should be smaller or equal to 0**" />

-> CusMacroAlarmMsg_Common.Xml：

<Message ID="AlarmMsg::**Macro**::ID=3" Content="**max arc length can not be negative**" />

24.4 Edit Alarm String Through SI (SyntecIDE)

MACRO alarm string editor is already integrated with SI, for related manual please refer to Macro Alarm String Editor.

25 MACRO Customized Message (MSG)

25.1 MSG Specification Explanation

- If MACRO alarm occurs, system must be reset to clear the alarm. While MSG is able to be cleared by clicking "ESC". MSG can be used for prompt simply. However, MSG vanishs when the program finishes.
- There is limit of string length of MSG. For Chinese, it's 19 characters; For English, it's 39 alphabets.

25.2 MSG Trigger Syntax

- MSG(100);// MSG ID

- MSG("Missing Drill");// Display MSG content

- MSG("100,Missing Drill");// MSG ID + Display Content

26 **Appendix**

26.1 Macro User Guide

26.1.1 Preface

- The built-in G, T, M code may not satisfy demand from all walks of life, so Syntec Corp. provides "customizing macro" for customer.
  Developer is able to, according to the machine properties, develop macros with special actions, which greatly promotes the machine value.
- Before introduction, the methods of calling other programs in main program are as below:
  - Call sub-program: execute sub-program, while reading or occupying argument is forbidden.
  - Call macro: execute macro, while reading and occupying argument is allowed.
  - For argument definition please refer to Argument Explanation.
- The following sections introduces related specification of macro, and the specification of calling sub-program will be skipped.

26.1.2 Macro Classification

- All Macro has to meet the following conditions:
  - Correct macro syntax
    - Program starts with %@MACRO.
    - Each row(block) ends with " ; " (a semicolon).
  - No file extension.
  - Program ends with M99 to return to main program that calls the macro.
- Basically, macro can be categorized into following types
  - G code macro
  - Non-mode call G code (G65)
  - Mode G code (G66/G66.1)
  - T code macro
  - M code macro

Type	Characteristic	Enable Condition	File Name Specification	File Name Range

| G Code Macro | • G code macro developed by developer, which is known as expansion G code macro and in contrast to standard G code. | None | • The beginning of file name must be letter G.
• No file extension
• The file name of expansion G code macro can be separated into that with 4 digit number or 6 digit number. | • G200~G999
• If the number is out of the range, file is not guaranteed to work normally.
• May conflict with standard G code. |

4 digit number   6 digit number

• Without decimal digits in G code instruction.
• All zeros starting from the largest digit of number can be omitted.

Instruction	Filename	Explanation
• G200 • G0200	G0200	If there is no decimal part in G code instruction, then the File name = G + four digit number

4 digit number   **6 digit number**

• With decimal digit in G code instruction.
• Three digits starting from the left of number in file name corresponds to the integer part of G code instruction
• Last three digits correspond to the decimal part of G code instruction
• All zeros starting from the largest digit of number can be omitted.

Instruction	Filename	Explanation
G200.1	G200001	If there is decimal part in G code instruction, then the file name = G +six digit number
G200.001		
G200.10	G200010	
G200.010		
G200.100	G200100	

| Non-mode G code (G65) | • Call assigned program through macro
• Must be the last G code in that row. | None | • The beginning of file name must be letter O.
• No file extension.
• In file name, except O, other characters must be number.
• Omit the O while calling.
    • E.g, for file name O0123, instruction is G65 P123. | • O0000~O09999
• If the number is out of the range, file is not guaranteed to work normally. |

Mode G code (G66 / G66.1)	Type	Effect	None	• The beginning of file name must be letter O. • No file extension. • In file name, except O, other characters must be number. • Omit the O while calling. • E.g, for file name O0123, instruction is G65 P123.	• O0000~ O9999 • If the number is out of the range, file is not guaranteed to work normally.
	G66	Call assigned program through macro when every block with moving instruction is finished.			

Type	Effect
G66.1	Call assigned program through macro when every block is finished.

- Must be the last G code in that row.

T Code Macro	Call T1000 file through macro.	Pr3215 Enable T code call mode must be 2.		• File name can only be T0000, other file names are not allowed. • No file extension.	T0000
		Pr 3 2 1 5	**Type**		
		0	• T code supplementary code • does not call T0000		
		1	• Call T0000 through sub-program. • Does not read and occupy any argument		

Pr 3215	Type
2	• Call T0000 through macro. • Read and occupy any argument

M code Macro	Different from M code, M code macro calls corresponding M code macro file through macro.	• M code must be logged in Pr3601~3610 "M code MACRO call registry". • Following M code is standard M code and can't be logged as M code macro. | M 00	M 30	M 98
M 01	M 96	M 99
M 02	M 97			• The beginning of file name must be letter M. • No file extension. • The file name must be "M+4 digit number". The 4 digit number corresponds to M code macro instruction. • Example	• M0000~ M9999 • If the number is out of the range, file is not guaranteed to work normally.

File Name	Pr 3601	Calling Instruction
M0123	123	• M123 • M0123

26.1.3 Macro Operation Process Explanation

- Next, G code macro is taken as example for introduction. Any difference of specification between G code macro and other macros will be pointed out.
- When G code is executed, actually the system is executing the content of G code macro.
- Write "M99" in the last row of G code macro. After M99 is executed, system returns to main program and keep processing.
- Instruction in G code macro may change the system state. For example, G90/G91 in G code macro change the G code mode.
    - In order to avoid influencing the main program or other macros/ sub-programs, usually, the G code mode is backed up first while executing G code macro and restore it before leaving G code macro.
- Example:
    - Execute G200 in main program, then execute G code macro G0200.
    - In G0200, backups G code mode first.
    - X coordinate increase in increment of 10.
    - Restore G code mode before leaving.

Example_Main

```
1  // Main
2  G90;
3  G01 X10. F100.;          // X Axis moving by G01    result X=10.
4  G200;                    // X Axis moving by G200   result X=20.
5  X- 20 .;                  // X Axis moving by G01    result X=-20.
6  M30;
```

Example_G0200

```
1  // G0200
2  % @MACRO
3  # 101  := # 1000 ;          // Backup #1000, G00/G01/G02/G03/G33/G34/G35
4  # 102  := # 1004 ;          // Backup #1004, G90/G91
5  G91 G00 X10.;               // X coordinate increase in increment of 10 by G00.   result X=20.;
6  G# 101 ;                    // Restore #1000, G00/G01/G02/G03/G33/G34/G35
7  G# 102 ;                    // Restore #1004, G90/G91
8  M99;                        // Return to main program
```

26.1.4 Introduction of Argument Usage

- Macro is able to execute the instructions designed by developers, from simple state-changing to complicated multi-processing.
- If a macro only deals with one process at the same time, then developers have to write uncountable macros for all sorts of situations.

- For example, the function of a macro is to cut 10*10 cm squares, if user wants to cut 20*20 cm squares then a new macro is needed.
- This kind of macro is too inflexible. However, if the content is able to be adjusted through arguments, for example, the edge of a square, then it is much more flexible in capability.
- Example:
    - In main program, system executes G201 to cut a 10*10 cm square.
    - In main program, system executes G201 to cut a 20*20 cm square.
    - In main program, system executes G202 to cut a square, length of edge of which is decided by argument.

Example_Main

```
1   // Example002_Main
2   G90 G00 X0. Y0.;
3   G200;
4   G201;
5   G202 X30.;
6   M30;
```

Example_G0200

```
1    // G0200
2    % @MACRO
3    # 101 :=# 1000 ;
4    # 102 :=# 1004 ;
5    G91 G01 X10.;
6    G91 G01 Y10.;
7    G91 G01 X- 10 .;
8    G91 G01 Y- 10 .;
9    G# 101 ;
10   G# 102 ;
11   M99;
```

Example_G0201

```
1    // G0201
2    % @MACRO
3    # 101 :=# 1000 ;
4    # 102 :=# 1004 ;
5    G91 G01 X20.;
6    G91 G01 Y20.;
7    G91 G01 X- 20 .;
8    G91 G01 Y- 20 .;
9    G# 101 ;
10   G# 102 ;
```

```
11    M99;
```

Example_G0202

```
 1    // G0202
 2    % @MACRO
 3    # 101 :=# 1000 :
 4    # 102 :=# 1004 ;
 5    # 103 :=# 24 :
 6    G91 G01 X# 103 :
 7    G91 G01 Y# 103 ;
 8    G91 G01 X-# 103 :
 9    G91 G01 Y-# 103 ;
10    G# 101 :
11    G# 102 ;
12    M99;
```

26.1.5 Arguments Explanation

- Arguments consist of 26 letters. Expect G/N/O, each letter has a corresponding local variable (# variable). They are shown in the table below.

Argument	#	Argument	#	Argument	#	
A	#1	J	#5	S	#19	
B	#2	K	#6	T	#20	
C	#3	L	#12	U	#21	
D	#7	M	#13	V	#22	
E	#8	N		W	#23	
F	#9	O		X	#24	
G		P	#16	Y	#25	
H	#11	Q	#17	Z	#26	
I	#4	R	#18			

Instruction			
Category	Axis Argument	Condition Argument	Special Argument	Exception Argument

- Argument will be read and occupied by macro.
- "Read" means there is # variable to which corresponds to the argument in the macro, so the macro reads the argument to start operation.
- "Occupied" means argument is not able to be read by other macro after it is read.
- It doesn't mean argument is occupied if it is read. The occupation depends on the macro characteristics. Please refer to the next 2 sections, "**Macro Explanation of process order**" & "**Macro Characteristics**".
- It doesn't mean argument will not be occupied if it is not read. The occupation depends on the characteristics of the argument. Please refer to the following introduction of this chapter.
- If there are repeated arguments in the same line, only the last argument is read, which means the next argument override the previous one.
- Basically, argument is separated into categories as below:

Category	Argument	Characteristic	Other Explanation
Axis Argument	X Y Z A B C I J K U V W	As long as one of the arguments is occupied by a macro, other axis arguments are also occupied by that macro.	• B code is defined by Pr3806 Second auxiliary code | Pr3806	Type	Explanation
---	---	---
0	Axis Argument	
1	Auxiliary B code	Put value of B code into R5	

Category	Argument	Characteristic	Other Explanation
Condition Argument	F S T D E H P Q R M B	Compared to axis argument, condition argument is independent from each other. If one of the condition argument is occupied, only that one is occupied instead of all condition arguments.	• T code is defined by Pr3215 Enable T code call mode.

• T code is defined by Pr3215 Enable T code call mode.

Pr3215	Type	Explanation
0	T code auxiliary code do not call T000	only put T code value into #1036 and the corresponding R value (each axis has different corresponding R value)
1	Call T0000 through sub-program	do not accept any argument. Don't regard as macro.
2	Call T0000 through macro	accept argument.

• M code is defined by Pr3601~3610 M code Macro call registry

Pr3601~ 3610	Type	Explanation
No login M code	M code auxiliary code	only put M code value into #1038 and the corresponding R value (each axis has different corresponding R value)

Category	Argument	Characteristic	Other Explanation
			Pr3601~3610 / **Type** / **Explanation** Login M code / M code macro / accept argument. • B code is defined by Pr3806 Second auxiliary code **Pr3806** / **Type** / **Explanation** 0 / Axis Argument / 1 / B code axuiliary code / put B code value into R5
Special Argument	L	L argument is setting up macro repeated counts. • For example: G200 L10, it means G200 will continuously execute ten times. • Even if the user doesn't input L argument when calling the macro, the system will automatically fill in L=1 to let macro execute once.	T code macro do not read L code. As a result, no matter what L code value is, T code macro always execute once.
Exception Argument	G N O	These three letters is the keyword in the controller syntax, so they are unable to be used as argument. That's why there are not corresponding # variables.	**Type** / **Explanation** G / Use as G code instruction or G code macro. N / Use as program flag O / Head of file name of normal processing program.

26.1.6 Interpreting Order of Macro

- Before explaining how macro reads and occupies arguments, user have to understand the interpreting order of macro.
- System starts interpreting from the first row and it is no problem if there is only one macro/instruction in each row.
- However, if there are several macro or instructions in a row, system interprets them according to the interpreting order of macro.
  Following table is the interpreting order of macro.

Order	Type	Particulars	Example
1	part of G code	G code macro	G73, G84, etc
		Modal G code	G15, G17, G70, etc
		One-shot G code	G65
		The interpreting order of above instructions/macro depends on the sequence of writing, from left to right.	
2	macro	M code macro	
		T code macro	
		The interpreting order of above instructions/macro depends on the sequence of writing, from left to right.	
3	S code		
4	F code		
5	H code		
6	D code		
7	T code		
8	M code		
9	B code		
10	Function G code		G04, G51, G68, etc

Order	Type	Particulars	Example
11	Interpolation G code		G00, G01, etc

- Normally, the interpreting order also represents the order of occupying argument, which means "the macro interpreted first occupies the arguments first."
  Nonetheless, this is not absolutely right, because the occupying order also depends on macro characteristics.
  Some macros are interpreted first but executed last. Hence, this situation does not comply to the rule, "earlier interpreted, earlier occupying."
- Example:
  a. Function G code is interpreted earlier than interpolation G code is, so function G code occupies axis argument first.
     Even though interpolation G code is on the left to the function G code in the same block, function G code still occupies axis argument earlier.
  b. In table above, interpreting order of both M code macro and T code macro are level 2, which means if if T code and M code are in the same row,
     interpreting order depends on the sequence, the code on the left is interpreted first.

26.1.7 Macro Characteristics

- When macro is reading occupied arguments, besides the basic operation explanation of argument above, there are some characteristics which is briefly explained as below:

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Activate Condition
G code and G code macro	None
T code macro	• Pr3215 Enable T code call mode. This parameter is set up as 2. After rebooting, system regard T code as T code macro. • If T code is considered T code macro, it only executes T code macro if T argument (#20) is not occupied.
M code macro	• Pr3601~3610 M code Macro call registry. M code is entered in this section of parameter. After rebooting, system regard M code as M code macro. • If M code (#13) argument is occupied, M code macro do not execute. • If any axis argument is occupied, M code macro do not execute.

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Character

G code and G code macro	• Inherit Function
	  • If interpolation mode (#1000) is updated, G code inherit value from it. This is inherit function.
	  • Because of inherit function, as long as axis instruction is inputted, G code is executed.
	  • For example：System activate X, Z axes
	    G98 G83 Z-40.0 R-5.0 P0.0 Q10.0 F1.5;
	    X-3.; // execute G83 X-3. again

T code macro	• T code updates different variables according to the type of T code. Following are corresponding updating rules. | T code condition update	#1036	#20	R value | |---|---|---|---| | Macro | O | X | X | | Sub-program | O | X | X | | Auxiliary Code | O | X | O | | Argument | X | O | X | • If user want to capture T code value in a T code macro, please use T code variable (#1036). • If user want to capture T code value in other macros, please use T argument (#20). • Only when T code is auxiliary code, the corresponding R value of T code will be updated. If T code is used as macro or argument, the corresponding R value of T code is not updated.

Category	Character
M code macro	• M code updates different variable according to the type of M code. Following are corresponding updating rules.

T code condition update	#13	R value
Macro	O	X
Auxiliary Code	X	O
Argument	O	X

- #13 is updated in M code macro or argument. Please note that this is different from specification of T code.
- Only when M code is auxiliary code, the corresponding R value of M code will be updated. If M code is used as macro or argument, the corresponding R value of M code is not updated.

Condition   Character   **Argument reading and occupation**   Multi same code instruction in a row

Other command in macro

Category	Argument reading and occupation
G code and G code macro	• While executing G code macro, argument read by G code macro is occupied. • If axis argument is read, all axis arguments are occupied simultaneously.
• Modal G code • One-shot G code	• It occupies all the argument except T code.
T code macro	• While executing T code macro, system read arguments which are needed to. No matter whether argument is occupied or not. • After executing T code macro, all arguments read by T code macro are occupied. If axis argument is read, all axis arguments are occupied simultaneously. • Repeated count of T code macro is not defined by L code, and T code macro always executes only one time. • The L code argument (#12) is not read by system, even if there is one in T code macro. System just puts 1 into #12.

M code macro	• After executing M code macro, system occupies all arguments which are read by M code macro.
	    • If M code macro doesn't read T argument, then after finishes executing M code macro, T argument is not occupied.
	    • If M code macro reads T argument, then after finishes executing M code macro, T argument is occupied.

Condition   Character   Argument reading and occupation   **Multi same code instruction in a row**

Other command in macro

Category	Multi same code instruction in a row
G code and G code macro	• Being interpreted from left to right • Only the last G code reads and occupies argument. G code in the front do not read or occupy any argument. • The last G code can be sorted into situations as below

Category		Explanation	Read occupied argument	Execution
• G code macro		Immediately	Read and occupy argument	Execute immediately
• Call modal macro	G66	Immediately	Read and occupy argument	Do not execute immediat System executes G code instruction is finished.
	G66.1	Immediately	Read and occupy argument	Do not execute immediat System executes G code instruction is finished.
• Interpolation G code • Function G code		Immediately	No	If other instructions in th corresponding argument

Category	Multi same code instruction in a row
T code macro	• If multi T code macros are in the same row. Each T code is interpreted sequentially and each T code macro can read a
M code macro	• If multi M code macro are in the same row, only the first M code macro is interpreted. • Pr3810 Parallel executing multiple M code in one block is for auxiliary M code not M code macro. Hence, even if Pr3810

Condition   Character   Argument reading and occupation   Multi same code instruction in a row

Other command in macro

Category	Command in macro
G code macro	• G code in G code macro can be instruction or macro. • M code in G code macro can be auxiliary code or macro. • T code in G code macro can be auxiliary code, macro or sub-program.
T code macro	• G code in T code macro can be instruction or macro. • M code in T code macro is only considered auxiliary code. • T code in T code macro is only considered auxiliary code.

Category	Command in macro
M code macro	• G code in M code macro can be instruction or macro. • M code in M code macro is only considered auxiliary code. • T code in M code macro is only considered auxiliary code.

26.1.8 Macro Calling Example

- In macro characteristics, many macro usage conditions have been mentioned. Next, example of macro calling argument will be provided

- If Doesn't especially mention any macro example. It is available to all macros.

01_Argument and program variable (local variable)

- Except the exception argument (G,N,O), all arguments correspond to a # variable (local variable)

Main

```
1   // Main
2   G200 A1 B2 C3 D7 E8 F9 H11 I4 J5 K6 L12 M13 P16 Q17 R18 S19
    T20 U21 V22 W23 X24 Y25 Z26;
3   M30;
```

G0200

```
1    // G0200
2    % @MACRO
3
4    //Axis Argument
5    @101   := # 1 :                          // A
6    @102   := # 2 :                          // B
7    @103   := # 3 :                          // C
8    @104   := # 4 :                          // I
9    @105   := # 5 :                          // J
10   @106   := # 6 ;                          // K
11   @121   := # 21 :                         // U
12   @122   := # 22 :                         // V
13   @123   := # 23 :                         // W
14   @124   := # 24 :                         // X
15   @125   := # 25 :                         // Y
16   @126   := # 26 ;                         // Z
17
18   //Condition Argument
19   @107   := # 7 :                          // D
20   @108   := # 8 ;                          // E
```

```
21   @109  := # 9 ;                              // F
22   @111  := # 11 ;                             // H
23   @113  := # 13 ;                             // M
24   @116  := # 16 ;                             // P
25   @117  := # 17 ;                             // Q
26   @118  := # 18 ;                             // R
27   @119  := # 19 ;                             // S
28   @120  := # 20 ;                             // T
29
30   //Special Argument
31   @112  := # 12 ;                             // L
32
33   M00;                          // Switch the screen to
     [Diag.] → [Display Global] and [Display Coord.]
34   WAIT();
35   M99;
```

02_Repeated argument

- Only the last repeated argument is read.
- Because same argument put value into the same # variable, the last one overrides the previous one.

Main

```
1   // Main
2   G01 X0.;
3   G200 X10. X- 10 .;      // X argument has been written twice,
    only X-10 is read by G200
4                           // #24 is input 10 by X10 first
5                           // then input -10 by X-10.
6                           // G01 will be executed after completing
    G200,
7                           // because G01 can't read any argument X
    which is occupied by G200
8   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 100  := # 1000 :       // Backup interpolation mode
4   # 101  := # 24 :         // The #24 is -10.
5   G01 X# 101 ;            // Move to X-10.
6   M00;
7   WAIT();
8   G# 100 ;                // Restore to interpolation mode
9   M99;
```

03_Axis argument and condition argument

- All the axis arguments are occupied if one of them is occupied.
- Condition arguments are independent from each other.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30.;            // Though G200 only
    reads X argument
4                                   // Y, Z arguments are
    also occupied
5                                   // G01 will be executed
    after completing G200
6                                   // because G01 can't
    read any argument which is occupied by G200
7   G01 X0. Y0. Z0. F100.;
8   G201 P20. X10. Y20. Z30. F200.;     // Since G201 only reads
    P argument
9                                   // X, Y, Z, F arguments
    are not occupied
10                                  // G01 will be executed
    after completing G201 and
11                                  // read X10. Y20. Z30.
    F200.
12                                  // , and then execute
    the corresponding actions
13  M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   @1 :=# 24 ;               // Only reads X argument
4   M00;
5   WAIT();
6   M99;
```

G0201

```
1   // G0201
2   % @MACRO
3   @1 :=# 16 ;               // Only reads P argument
4   M00;
5   WAIT();
```

```
6   M99;
```

04_H code as a condition argument

- H code is condition argument

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. H40.;            // G200 only reads H
    argument
4                                        // H argument is a
    condition argument
5                                        // G01 will be executed
    after completing G200.
6                                        // G01 read and occupy
    X10. Y20. Z30.
7                                        // and then execute the
    corresponding actions
8   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 11 ;        // Only reads H argument
4   M00;
5   WAIT();
6   M99;
```

05_H code as an axis argument

- If Pr3809 "*Are UVW incremental command of XYZ axes" is set to 1,H code is not only an axis argument but also a condition argument.
- Nevertheless, H argument is regarded as condition argument and read in macro. Therefore, after completed the execution of G code macro, the H argument is occupied again as an axis argument and system does the corresponding action.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. H40.;            // G200 only reads H
    argument
```

```
4                                    // Since Pr3809=1，the H
   argument is regarded as both axis argument and condition
   argument
5                                    // However, it is
   occupied as a condition argument in MACRO
6                                    // G01 is executed after
   completing G200
7                                    // G01 reads and
   occupies X10. Y20. Z30. H40., and
8                                    // system executes the
   corresponding action
9                                    // Since the axis
   argument of H40. is not occupied, it is then occupied as an
   axis argument by
10                                   // G01 again.
11   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 11 ;        // only reads H argument
4   M00;
5   WAIT();
6   M99;
```

- When reading axis arguments in MACRO, all the axis arguments will be occupied if one of them is occupied.
- As a result, the axis argument of H argument is occupied as well.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. H40.;        // G200 only reads X
   argument
4                                    // Since Pr3809=1，the H
   argument is regarded as both axis argument and condition
   argument
5                                    // The axis argument of
   H argument is also occupied
6                                    // G01 will be executed
   after completing G200
7                                    // G01 can't read any
   argument since the axis arguments are all occupied by G200.
8                                    // The axis argument of
   the H argument is also included and won't be read.
9   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 24 ;        // Only reads X argument
4   M00:
5   WAIT();
6   M99;
```

06_B code as an axis argument

- If Pr3806 Second auxiliary code is set to 0, B code is regarded as an axis argument.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. B40.;            // G200 only reads B
    argument
4                                        // Since Pr3806=0, B
    argument is regarded as an axis argument
5                                        // X, Y, Z arguments is
    occupied by G200
6                                        // G01 is executed after
    completing G200
7                                        // and G01 can't read
    any argument since all arguments are occupied by G200
8   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 2 ;        // Only reads B argument
4   M00:
5   WAIT();
6   M99;
```

07_B code as a condition argument

- If Pr3806 Second auxiliary code is set to 1, B code is regarded as a condition argument.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 X10. Y20. Z30. B40.;          // G200 only reads B
     argument
4                                        // Since Pr3806=1, B
     argument is seen as a condition argument
5                                        // X, Y, Z arguments are
     not occupied by G200
6                                        // G01 is executed after
     completing G200
7                                        // G01 reads and
     occupies X10. Y20. Z30., and
8                                        // executes the
     corresponding actions
9    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 2 ;        // Only reads B argument
4    M00;
5    WAIT();
6    M99;
```

08_Multiple G code macros in the same line

- When multiple G codes are written in the same line, no matter G code command or G code macro, system interprets all of them in order.
- However, only the G code macro written last reads and occupies arguments. The former G codes do not read or occupy any argument even though they are interpreted first.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    G200 G201 X10. P20.;          // Since 2 G code macros are
     written in the same line
4                                  // only the last G code macro is
     able to read the argument
5                                  // Though G201 didn't read the X
     argument
6                                  // G200 still can't read the X
     argument, because X argument is axis argument
```

```
 7                                // G201 is executed after
    completing G200，and
 8                                // G01 is be executed after G201
    read and occupied P argument
 9                                // G01 reads and occupies X10.,
    and
10                                // then executes the
    corresponding actions
11   M30;
```

G200

```
 1   // G0200
 2   % @MACRO
 3   # 100  := # 1000 :              // Backup interpolation mode
 4   # 101 :=# 24 ;                  // Read X argument which is
    unable to be read
 5   G01 X# 101 ;                    // Since #101 has no value,
    this line won't be executed
 6   G01 X0.;                   // This line will be executed
 7   M00:
 8   WAIT();
 9   G# 100 ;                        // Restore interpolation mode
10   M99;
```

G201

```
 1   // G0201
 2   % @MACRO
 3   # 100  := # 1000 :              // Backup interpolation mode
 4   # 101 :=# 16 ;                  // Read P argument which is
    able to be read
 5   G01 X# 101 ;                    // This line will be executed
 6   G01 X0.;
 7   M00:
 8   WAIT();
 9   G# 100 ;                        // Restore interpolation mode
10   M99;
```

09_G code instruction and G code macro in the same line

- When multiple G codes or G code macros are written in the same line, they are interpreted in order.
- Only the G code macro written last can read and occupy arguments. Though the former G codes are interpreted first, they do not read and occupy any argument.

Main

```
1    // Main
2    G90;
3    G00 X0. Y0. Z0.;              // G00, spindle moves to X0 Y0 Z0
4    G200 G01 X10. F100.;          // Since 2 G codes are in the same line.
5                                  // only the last G code can read the argument
6                                  // G200 is executed but no arguments are read
7                                  // G01 X10. F100. is executed after completing G200
8
9    G00 X0. Y0. Z0.;              // G00, spindle moves to X0 Y0 Z0
10   G01 G201 X10. F100.;          // Since 2 G codes are in the same line.
11                                 // G01 is interpreted first, interpolation mode changes to G01.
12                                 // G201 is interpreted secondly.
13                                 // In G201, system executes G01 X10. F100.
14                                 // Since G201 reads and occupies the axis argument, all the axis arguments are occupied.
15                                 // Therefore, there is no moving instruction left for G01, so no movement.
16
17   G00 X0. Y0. Z0.;              // G00, spindle moves to X0 Y0 Z0
18   G01 G202 X10. F100.;          // Since 2 G codes are in the same line
19                                 // G01 is explained first, interpolation mode changes to G01
20                                 // G202 is interpreted secondely.
21                                 // In G202, no action is executed and no arguments are read or occupied.
22                                 // Therefore, there are axis arguments left for G01, so system executes G01 X10. F100.
23   M30;
```

G200

```
1    // G0200
2    % @MACRO
3    # 100  := # 1000 ;            // Backup interpolation mode
```

```
4   # 101  := # 24 ;                 // Read X argument but it's
    unable to read
5    G# 100  X# 101  F# 9 ;          // This line is not
    executed since #101 has no value. However, since #100=0, the
    interpolation mode changes to G00.
6    M00;
7    WAIT();
8    M99;
```

G201

```
1    // G0201
2    % @MACRO
3    # 100  := # 1000 :              // Backup interpolation mode
4    # 101  := # 24 ;                 // Read X argument and read
    X=10.
5    G# 100  X# 101  F# 9 ;          // Because #100 = 1,
    system executes G01.
6    M00;
7    WAIT();
8    M99;
```

G202

```
1    // G0202
2    % @MACRO
3    M00;
4    WAIT();
5    M99;
```

10_T code macro

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- Executing T code macro if T argument is not occupied.
- #20 and R3 have no value, but #1036 shows the value of T code.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 X10. Y20. Z30.;        // T argument is not occupied and
    Pr3215=2, so
4                                // system executes T code macro
5                                // With X argument being read in T
    code macro,
```

```
6                              // Y, Z arguments are occupied by T
code macro because all of them are axis arguments.
7                              // G01 is executed after completing
T01
8                              // G01 can't read any argument since
the arguments in the line are all occupied by T coda macro.
9    M30;
```

T0000

```
1    // T0000
2    % @MACRO
3    # 101  := # 24 ;            // only reads X argument
4    M00;                       // Switch the screen to [Diag.] →
 [Display Global] and [Display Coord.]
5                              // Observe #20/#1036/#101
6    WAIT();
7    M99;
```

11_G code macro and T code macro in the same line

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- When G code macro and T code macro are in the same line, G code macro is interpreted first then the T code macro.
- If the T code is occupied by G code macro then T code macro is not executed.
- If the T code is not occupied by G code macro then T code macro is executed after G code macro is completed.
- There is only one T0000 in system, but in the example, there are two T code macros named after T0000_T01 and T0000_T02.

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 G200 X10. Y20. Z30. F1000;      // According to the
interpretation order, G code macro is interpreted first.
4                                      // There is a T argument
read in G code macro (G0200), so
5                                      // T argument is
occupied and T code macro is not executed.
6                                      // G01 is executed after
completing G code macro (G0200).
7                                      // G01 reads and
occupies X10. Y20. Z30. F1000, and
8                                      // executes the
corresponding actions
9
```

```
10    T02 G201 X10. Y20. Z30. F1000.;      // According to the
      interpretation order, G code macro is interpreted first.
11                                          // There is no T
      argument read in G code macro (G0201), so
12                                          // T argument is not
      occupied, and
13                                          // T code macro is
      executed after G code macro (G0201) is completed
14                                          // Because T02 reads all
      arguments, G01 can't read any argument
15    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 101  := # 20 ;            // Only reads T argument
4    M00;
5    WAIT();
6    M99;
```

T0000_01

```
1    // Not executing this process
2    // T0000
3    % @MACRO
4    # 201  := # 1000 ;
5    # 202  := # 1004 ;
6    # 101  := # 24 ;
7    # 102  := # 25 ;
8    # 103  := # 26 ;
9    # 104  := # 9 ;
10   M00;                // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
11   WAIT();
12   G91 G01 X# 101  Y# 102  Z# 103  F# 104 ;
13
14   G# 201 ;
15   G# 202
16   M99;
```

G0201

```
1    // G0201
2    % @MACRO
```

```
3    # 101  := # 24 ;          // Only reads X argument
4    M00;                      // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
5    WAIT();
6    M99;
```

T0000_02

```
1    // T0000
2    % @MACRO
3    # 201  := # 1000 ;
4    # 202  := # 1004 ;
5    # 101  := # 24 ;
6    # 102  := # 25 ;
7    # 103  := # 26 ;
8    # 104  := # 9 ;
9    M00;                      // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
10   WAIT();
11   G91 G01 X# 101  Y# 102  Z# 103  F# 104 ;      // T code macro
     can still read the arguments and execute the corresponding
     actions
12
13   G# 201 ;
14   G# 202
15   M99;
```

12_T code macro is not affected by L Argument

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- T code macro does not read or occupy L argument

Main

```
1    // Main
2    G01 X0. Y0. Z0.;
3    T01 L2;              // Though T0000 reads #12,
4                         // #12 is equal to "1", not "2" given by L
     argument
5                         // Execute G01 after completed T01
6                         // G01 reads and occupies L2, and
7                         // executes the corresponding actions (G01
     L2 is meaningless)
8    M30;
```

T0000

```
1   // T0000
2   % @MACRO
3   # 101  := # 12 ;           // Only reads L argument
4   M00;                       // Switch the screen to [Diag.] →
    「Display Global] and [Display Coord.]
5   WAIT();
6   M99;
```

13_Multiple T code macros in a line

- Pr3215 Enable T code call mode is set to 2, T0000 is the T code macro.
- When multiple T code macros are written in the same line, T code is interpreted in order and every T code macro can read the arguments.
- There is only one T0000 in system, but in the example, there are two T code macros named after T0000_T01 and T0000_T02.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   T01 T02 X10. Y20. Z30.;      // This line is executed twice
4                                // T01 is executed first then
    T02
5                                // Both T code read X, Y, Z
    arguments successfully
6                                // If there is any other macro
    going to read X, Y, Z,
7                                // it reads nothing since
    arguments are already occupied by T code macro.
8                                // G01 is executed after
    completing T02 with all argument occupied.
9   M30;
```

T0000_T01

```
1   // T0000_T01
2   % @MACRO
3   # 100  := # 1000 ;        // Backup interpolation mode
4   # 101  := # 24 ;          // Read X argument
5   # 102  := # 25 ;          // Read Y argument
6   # 103  := # 26 ;          // Read Z argument
7   IF # 1036 = 1  THEN     // T01 → #1036=1
8      G01 X# 101 ;          // X axis moving
9      G01 X0.;
```

```
10    END_IF;
11
12    IF # 1036  =  2  THEN    // The section is not executed
13        G01 Y# 102 ;
14        G01 Y0.;
15    END_IF;
16
17    M00;                    // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
18    WAIT();
19    G# 100 ;                // Restore interpolation mode
20    M99;
```

T0000_T02

```
1    // T0000_T02
2    % @MACRO
3    # 100  := # 1000 ;      // Backup interpolation mode
4    # 101  := # 24 ;        // Read X argument
5    # 102  := # 25 ;        // Read Y argument
6    # 103  := # 26 ;        // Read Z argument
7    IF # 1036  =  1  THEN    // The section is not executed
8        G01 X# 101 ;
9        G01 X0.;
10   END_IF;
11
12   IF # 1036  =  2  THEN    // T02 → #1036=2
13       G01 Y# 102 ;         // Y axis moving
14       G01 Y0.;
15   END_IF;
16
17   M00;                    // Switch the screen to [Diag.] →
     [Display Global] and [Display Coord.]
18   WAIT();
19   G# 100 ;                // Restore interpolation mode
20   M99;
```

14_M code macro

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.
- If M code is registered as an M code macro, it is executed only when the M code and all the other axis arguments are not occupied.
- While M code macro is executed, all the arguments are occupied except T.
- M argument (#13) is updated whether the M code is registered as M code macro or not.
- The corresponding R value of M code is updated only if M code is regarded as auxiliary code. R value is not updated if M code is regarded as macro or argument.

Main	
1	```// Main```
2	```G01 X0. Y0. Z0. F100.;```
3	```M123 X10. Y20. Z30. F300.; // No M argument and no axis argument is occupied, so```
4	``` // M code macro M0123 is executed.```
5	``` // While M code macro is executed, aside from the arguments read by the M code macro,```
6	``` // all other arguments are occupied as well except T argument.```
7	``` // Though the F argument is not read by M0123, it is occupied, then the feedrate is not updated.```
8	``` // G01, executed after completing M123,```
9	``` // can't read any argument since the arguments are all occupied.```
10	
11	```G01 X0. Y0. Z0.; // The F in this line is F100, since the argument in previous line is occupied by M code macro```
12	```M30;```

M0123	
1	```// M0123```
2	```% @MACRO```
3	```# 100 := # 1000 ; // Backup interpolation mode```
4	```# 101 := # 24 ; // Read X argument```
5	```# 102 := # 25 ; // Read Y argument```
6	```# 103 := # 26 ; // Read Z argument```
7	```G00 X# 101 Y# 102 Z# 103 ; // G00 moving along X, Y, Z axis```
8	```M00;```
9	```WAIT(); // Switch the screen to [Diag.] → [Display Global] and [Display Coord.]```
10	``` // user should found #13=123```
11	``` // Switch the screen to [PLC Status] → [PLC Register]```
12	``` // User should found R1=0```
13	```G# 100 ; // Restore interpolation mode```
14	```M99;```

15_G code macro and M code macro in the same line

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.

- When G code macro and M code macro are written in the same line, G code macro is interpreted before M code macro.
- If M code is occupied, the M code macro is not executed.
- On the opposite, if M code is not occupied, the M code macro is executed after G code macro completed.
- All the arguments except T will be occupied after executing the M code macro.

Main

```
 1   // Main
 2   G01 X0. Y0. Z0.;
 3   G1301 M1301 X10. Y20. Z30. P2 F1000.;      // G code macro is
     interpreted first.
 4                                              // The M argument is
     read by G code macro (G1301)
 5                                              // The M argument is
     regarded occupied, M code macro is executed.
 6                                              // Since the X, Y, Z
     arguments is not occupied by G code macro (G1301).
 7                                              // G01 is executed
     after completing the G code macro (G1301).
 8                                              // G01 reads and
     occupies X10. Y20. Z30. P2 F1000., and
 9                                              // then executes the
     corresponding instructions.
10
11   G1302 M1301 X10. Y20. Z30. F1000.;         // G code macro is
     interpreted first
12                                              // Only P argument
     is read by G code macro (G1302)
13                                              // The M argument is
     regarded unoccupied, and
14                                              // no other axis
     arguments are occupied
15                                              // The M code macro
     is executed after completing the G code macro (G1302)
16                                              // The M code macro
     (M1301) do not read any argument,
17                                              // but all arguments
     except T are regarded occupied after the M code macro is
     executed
18                                              // G01, executed
     after completing the M code macro,
19                                              // can't read any
     argument since the arguments are all occupied
20   M30;
```

G1301

```
 1   // G1301
```

```
2    % @MACRO
3    # 101  := # 13 ;            // Only reads M argument
4    M00;
5    WAIT();
6    M99;
```

M1301_01

```
1    // The program won't be executed
2    // M1301
3    % @MACRO
4    M00:
5    WAIT();
6    M99;
```

G1302

```
1    // G1302
2    % @MACRO
3    # 101  := # 16 ;            // Only reads P argument
4    M00:
5    WAIT();
6    M99;
```

M1301_02

```
1    // M1301
2    % @MACRO
3    M00:
4    WAIT();                  // The program does not read any
     argument
5                            // but occupies all the arguments except
     T
6    M99;
```

16_G code macro and M code macro are in the same line, and G code macro reads and occupies axis argument.

- If Pr3601~3610 M code Macro call registry is set to 123, M0123 is registered as M code macro.
- When G code macro and M code macro are written in the same line, G code macro is interpreted before M code macro.
- If axis arguments are occupied by G code macro, M code macro is not executed.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 M123 X10. Y20. Z30.;      // G code macro is interpreted
    first
4                                  // Because X argument is read by
    G code macro (G0200),
5                                  // all axis arguments are
    occupied by G200.
6                                  // M code macro therefore is
    executed since axis arguments are occupied.
7                                  // G01, executed after
    completing G code macro (G0200),
8                                  // can't read any argument since
    the arguments in the line are all occupied
9   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 24 ;          // Only reads X argument
4   M00;
5   WAIT();
6   M99;
```

M0123

```
1   //This program won't be executed
2   // M0123
3   % @MACRO
4   # 101  := # 24 ;
5   M00;
6   WAIT();
7   M99;
```

17_Multiple M code macros in a line

- Pr3601 is set to 123, and M0123 is registered as M code macro.
- Pr3602 is set to 124, and M0124 is registered as M code macro.
- When multiple M codes macro are written in the same line, only the first M code macro is interpreted.

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   M123 M124 X10. Y20. Z30.;     // M123 is interpreted first,
    and only M123 is executed.
4                                 // X10. Y20. Z30. are read by M
    code macro (M0123)
5                                 // All arguments except T are
    occupied after executing the M code macro.
6                                 // G01, executed after
    completing M code macro.
7                                 // can't read any argument since
    the arguments are all occupied.
8   M30;
```

M0123

```
1   // M0123
2   % @MACRO
3   # 100  := # 1000 ;            // Backup interpolation mode
4   # 101  := # 24 ;              // Read X argument
5   # 102  := # 25 ;              // Read Y argument
6   # 103  := # 26 ;              // Read Z argument
7   G00 X# 101  Y# 102  Z# 103 ;     // G00 moving along X, Y,
    Z axis
8   M00;
9   WAIT();                       // Switch the screen to [Diag.]
    → [Display Global] and [Display Coord.]
10                                // User should found #13=124
11                                // Since M124 is regarded as an
    argument, which overwrote #13
12  G# 100 ;                      // Restore interpolation mode
13  M99;
```

M0124

```
1   // This program won't be executed
2   // M0124
3   % @MACRO
4   # 101  := # 24 ;
5   M00;
6   WAIT();
7   M99;
```

18_T code macro and M code macro in the same line

- Pr3215 is set to 2, T0000 is the T code macro
- Pr3601 *M code Macro call registry is set to 123/124/125, M0123 / M0124 / M0125 are registered as M coed macros
- If T code macro and M code macro are written in the same line, the macro on the left is interpreted first.
- Whether the macro on the right is executed or not is decided by the rules in previous section [Macro Characteristics]

Main

```
 1    // Main
 2    G01 X0. Y0. Z0.;
 3     T01 M1601 X10. Y20. Z30.;     // According to the interpreting
      order, T01 is interpreted first, then M1601.
 4                                  // No argument is occupied by T
      code macro, so M code macro (M1601) executes X=10.
 5                                  // Although M code macro (M1601)
      does not read any argument,
 6                                  // all the arguments except T
      are occupied
 7                                  // G01, executed after
      completing M code macro (M1601),
 8                                  // can't read any argument since
      the arguments are all occupied.
 9
10     M1602 T02 X10. Y20. Z30.;     // According to the interpreting
      order, M1602 is interpreted first.
11                                  // M code macro (M1602) does not
      reading any argument.
12                                  // Because M code macro (M1602)
      occupies all arguments except T,
13                                  // T code macro (T0000_02) is
      able to be executed since T argument is not occupied by M
      code macro.
14                                  // G01, executed after
      completing T code macro,
15                                  // can't read any argument since
      the arguments in the line are all occupied
16
17     M1603 T03 X10. Y20. Z30.;     // According to the interpreting
      order, M1603 is interpreted first.
18                                  // M code macro (M1603) only
      reads the T argument
19                                  // M code macro (M1603) will
      occupy all the arguments after the execution, including T
      argument.
20                                  // Then, T code macro (T0000_03)
      is executed since T code here is regarded as an argument.
21                                  // G01 can't read any argument
      since the arguments in the line are all occupied.
```

```
22      M30;
```

T0000_01

```
1    // T0000
2    % @MACRO
3    # 100  := # 1000 ;        // Backup interpolation mode
4    # 101  := # 1004 ;        // Backup absolute/increment command
     mode
5
6    G91 G00 X10.;             // X axis moving in increment of 10.
7
8    M00;                      // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
9    WAIT();
10   G# 100 ;                  // Restore the interpolation mode
11   G# 101 ;                  // Restore the absolute/increment
     command mode
12   M99;
```

M1601

```
1    // M1601
2    % @MACRO
3    # 100  := # 1000 ;        // Backup interpolation mode
4    # 101  := # 1004 ;        // Backup absolute/increment command
     mode
5
6    G91 G00 Y10.;             // Y axis moving in increment of 10.
7
8    M00;                      // Switch the screen to [Diag.] →
      [Display Global] and [Display Coord.]
9    WAIT();
10   G# 100 ;                  // Restore the interpolation mode
11   G# 101 ;                  // Restore the absolute/increment
     command mode
12   M99;
```

M1602

```
1    // M1602
2    % @MACRO
3    # 100  := # 1000 ;        // Backup interpolation mode
4    # 101  := # 1004 ;        // Backup absolute/increment command
     mode
```

```
 5
 6    G91 G00 Y10.;           // Y axis moving in increment of 10.
 7
 8    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
 9    WAIT();
10    G# 100 ;                // Restore the interpolation mode
11    G# 101 ;                // Restore the absolute/increment
      command mode
12    M99;
```

T0000_02

```
 1    // T0000
 2    % @MACRO
 3    # 100  := # 1000 ;      // Backup interpolation mode
 4    # 101  := # 1004 ;      // Backup absolute/increment command
      mode
 5
 6    G91 G00 X10.;           // X axis moving in increment of 10.
 7
 8    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
 9    WAIT();
10    G# 100 ;                // Restore the interpolation mode
11    G# 101 ;                // Restore the absolute/increment
      command mode
12    M99;
```

M1603

```
 1    // M1603
 2    % @MACRO
 3    # 100  := # 1000 ;      // Backup interpolation mode
 4    # 101  := # 1004 ;      // Backup absolute/increment command
      mode
 5    # 102  := # 20 ;        // Read data of #20, the line will
      occupy the T argument
 6
 7    G91 G00 Y10.;           // Y axis moving in increment of 10.
 8
 9    M00;                    // Switch the screen to [Diag.] →
      「Display Global] and [Display Coord.]
10    WAIT();
11    G# 100 ;                // Restore the interpolation mode
12    G# 101 ;                // Restore the absolute/increment
      command mode
```

```
13    M99;
```

T0000_03

```
1     // This program won't be executed
2     // T0000
3     % @MACRO
4     # 100  := # 1000 ;        // Backup interpolation mode
5     # 101  := # 1004 ;        // Backup absolute/increment command
      mode
6
7     G91 G00 X10.;             // X axis moving in increment of 10.
8
9     M00;                      // Switch the screen to 〔Diag.〕→
      〔Display Global〕 and 〔Display Coord.〕
10    WAIT();
11    G# 100 ;                  // Restore the interpolation mode
12    G# 101 ;                  // Restore the absolute/increment
      command mode
13    M99;
```

19_One-shot macro calling (G65)

- G65 executes the macro file assigned by the P argument

Main

```
1     // Main
2     G01 X0. Y0. Z0.;
3     G65 P1 X10. Y20. Z30.;          // Execute O0001
4     M30;
```

O0001

```
1     // O0001
2     % @MACRO
3     # 101  := # 24 ;
4     # 102  := # 25 ;
5     # 103  := # 26 ;
6     M00;
7     WAIT();
8     M99;
```

20_Modal macro calling (G66)

- G66 is executed every time a movement block is completed.

Main

```
1  // Main
2  G01 X- 5 . Y- 5 . Z- 5 .;
3  G200 G66 P1 X10. Y20. Z30. ;        // G200 in this line can't
   read X argument
4  G200 X10.;                          // G200 in this line can
   read X argument
5                                      // There are 2 lines of G01
   movement blocks in G200
6                                      // G66 P1 X10. Y20. Z30. is
   executed every time G01 movement block is finished.
7  G67;
8  M30;
```

G0200_01

```
1   // G0200_01
2   % @MACRO
3   # 100  := # 1000 ;       // Backup interpolation mode
4   # 101  := # 24 ;         // Executing G200 for the first
    time, G200 can't read X argument, so
5   G01 X# 101 ;             // this line is not executed.
6   G01 X0.;
7   M00;
8   WAIT();
9   G# 100 ;                 // Restore the interpolation mode
10  M99;
```

G0200_02

```
1   // G0200_02
2   % @MACRO
3   # 100  := # 1000 ;       // Backup interpolation mode
4   # 101  := # 24 ;         // Executing G200 for the second
    time, G200 is able to read X argument, so
5   G01 X# 101 ;             // this line is executed
6                           // This line is a movement block, after
    finished, G66 P1 X10. Y20. Z30. is executed.
7   G01 X0.;                // This line is a movement block, after
    finished, G66 P1 X10. Y20. Z30. is executed.
8   M00;
9   WAIT();
```

```
10    G# 100 ;                    // Restore the interpolation mode
11    M99;
```

O0001

```
1    // O0001
2    % @MACRO
3    # 100   := # 1000 ;          // Backup interpolation mode
4    # 101   := # 24 ;
5    # 102   := # 25 ;
6    # 103   := # 26 ;
7    G91 G01 X# 101  Y# 102  Z# 103 ;
8    M00;
9    WAIT();
10   G# 100 ;                     // Restore the interpolation mode
11   M99;
```

21_Non modal call macro (G66.1)

- G66.1 is executed every time after a block is completed.

Main

```
1    // Main
2    G01 X- 5 . Y- 5 . Z- 5 .;
3    G66. 1  P1 X10. Y20. Z30. ;
4    G200 X10.;                    // G200 in this line can read
     the X argument
5                                  // G66.1 P1 X10. Y20. Z30. is
     executed every time the block in G200 is executed.
6    G67;                          // G66 P1 X10. Y20. Z30. is
     executed after this block is executed
7    M30;
```

G0200

```
1    // G0200
2    % @MACRO
3    # 100   := # 1000 ;          // Backup interpolation mode
4    # 101   := # 24 ;            // Executing G200 for the second
     time, G200 is able to read X argument
5    G01 X# 101 ;                 // This line is executed
6                                 // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
```

```
 7   G01 X0.;              // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
 8   M00;                  // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
 9   WAIT();               // This line is not a block
10   G# 100 ;               // Restore the interpolation mode
11                         // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
12   M99;                  // G66 P1 X10. Y20. Z30. is executed
     after this block is executed
```

O0001

```
 1   // O0001
 2   % @MACRO
 3   # 100  := # 1000 ;        // Backup interpolation mode
 4   # 101  := # 1004 ;        // Backup absolute/increment command
     mode
 5   # 111  := # 24 ;
 6   # 112  := # 25 ;
 7   # 113  := # 26 ;
 8   G91 G01 X# 111  Y# 112  Z# 113 ;
 9   M00;
10   WAIT();
11   G# 100 ;                  // Restore the interpolation mode
12   G# 101 ;                  // Restore the absolute/increment
     command mode
13   M99;
```

22_G65 and G66/G66.1 must be the last G code in the line

- One-shot G code macro calling (G65) and modal G code macro calling (G66/G66.1) must be the last G code in the line.

Main

```
 1   // Main
 2   G01 X0. Y0. Z0.;
 3   G65 P1 X10. Y20. Z30. G200;  // Alarm COR-013 shows up owing
     to wrong syntax in this line, because
 4                               // G65 and G200 are in the same
     line
 5                               // but G65 is not the last G
     code.
 6   M30;
```

G0200

```
1   // G0200
2   % @MACRO
3   # 101  := # 24 ;
4   M00:
5   WAIT();
6   M99;
```

O0001

```
1   // O0001
2   % @MACRO
3   # 101  := # 24 ;
4   # 102  := # 25 ;
5   # 103  := # 26 ;
6   M00:
7   WAIT();
8   M99;
```

26.2 MACRO XML Data Application

- MACRO is able to read xml files with special functions, which, respectively, are DBLOAD and DBOPEN. DBOPEN is used to load xml file and DBLOAD is used to read the data content.

- Application example：Following is a customized HMI, which automatically produces xml file recording the related machining data. The content of the xml file will be read and taken as reference when planning movements in macro afterwards.

No	X 間隙	Y 送線	Z 右旋	A 上切	B 左旋	C 下切	R	K	T	起始速度
1	0.00	17.63	12.98	267.54						
2	2.26	21.34	13.20	300.87						
3	2.26	91.19	13.20	443.29						
4	0.00	21.55	12.98	116.95						
5	-6.05	21.16	12.98	150.00						

-> The customized HMI first outputs the the user-defined contents to an xml file, and save the xml file to the GNCFILES file path of which is assigned by user (refer to Pr3219).

The syntax format is defined as below,：

```
<?xml version="1.0" encoding="UTF-16"?>
<CycleFile>

      <Cycle Name="Cycle_HerdonProg"> ← The beginning of first data

            <Field Name="Col_Y" Value="17.63"/>
            <Field Name="Col_Z" Value="12.98"/>
            <Field Name="Col_X" Value="0.00"/>
            <Field Name="Col_A" Value="267.54"/>

      </Cycle> ← The end of first data
      <Cycle Name="Cycle_HerdonProg"> ← The beginning of second data

            <Field Name="Col_Y" Value="21.34"/>
            <Field Name="Col_Z" Value="13.20"/>
            <Field Name="Col_X" Value="2.26"/>
            <Field Name="Col_A" Value="300.87"/>

      </Cycle> ← The end of second data
      <Cycle Name="Cycle_HerdonProg"> ← The beginning of third data

            <Field Name="Col_Y" Value="91.19"/>
            <Field Name="Col_Z" Value="13.20"/>
            <Field Name="Col_X" Value="2.26"/>
            <Field Name="Col_A" Value="443.29"/>

      </Cycle> ← The end of third data
      <Cycle Name="Cycle_HerdonProg"> ← The beginning of fourth data

            <Field Name="Col_Y" Value="21.55"/>
            <Field Name="Col_Z" Value="12.98"/>
            <Field Name="Col_X" Value="0.00"/>
            <Field Name="Col_A" Value="116.95"/>

      </Cycle> ← The end of fourth data
      <Cycle Name="Cycle_HerdonProg"> ← The beginning of fifth data

            <Field Name="Col_Y" Value="21.16"/>
            <Field Name="Col_Z" Value="12.98"/>
            <Field Name="Col_X" Value="-6.05"/>
            <Field Name="Col_A" Value="150.00"/>

      </Cycle> ← The end of fifth data

</CycleFile>
```

-> User have to write the configuration file of XML file by their own.

   Configuration file(schema file) defines the data to be read and the variable where data is put while DBLOAD is used.

   The syntax format is defined as below, and the configuration file should be saved in OCRes\\Common\\Schema\\

   If Dipole is enable, the schema file should be in the folder OCRes\\Common\\Schema\\ in controller.

```
<?xml version="1.0" encoding="UTF-16"?>
<Schema>

      <Cycle name="Cycle_HerdonProg">
```

```
        <Field>

                <Name>Col_X</Name>
                <InputStorage>@1200</InputStorage> ← The variable Col_X saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
        <Field>

                <Name>Col_Y</Name>
                <InputStorage>@1201</InputStorage> ← The variable Col_Y saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
        <Field>

                <Name>Col_Z</Name>
                <InputStorage>@1202</InputStorage> ← The variable Col_Z saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
        <Field>

                <Name>Col_A</Name>
                <InputStorage>@1203</InputStorage> ← The variable Col_A saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
        <Field>

                <Name>Col_B</Name>
                <InputStorage>@1204</InputStorage> ← The variable Col_B saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
        <Field>

                <Name>Col_C</Name>
                <InputStorage>@1205</InputStorage> ← The variable Col_C saves in
                <InputFormat>Variant</InputFormat>
                <DefaultValue></DefaultValue>

        </Field>
    </Cycle>
</Schema>
```

-> MACRO example

```
// Load GNCFILES\\Number of test data, total 5 data, therefore @1:=5;
@1:=DBOPEN("Test");
```
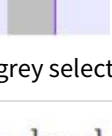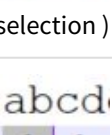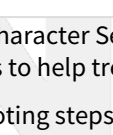
```
// Load the first data, DBLOAD argument is 0
// @1200=0.00 @1201=17.63 @1202=12.98 @1203=267.54
DBLOAD( 0 );


// Load the second data, DBLOAD argument is 1
// @1200=2.26 @1201=21.34 @1202=13.20 @1203=300.87
DBLOAD( 1 );
```

26.3 Recommended editing environment on PC

- notepad++：https://notepad-plus-plus.org/downloads/
  - Open Show All Characters：View→Show Symbol→Show All Characters
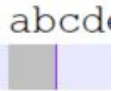   - Can check whether Marco files have unsupported characters.

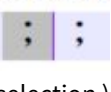**Common characters**

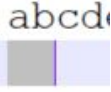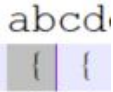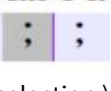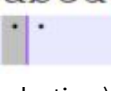| Name	Icon	Instruction	Supported
half-width spaces	abcd（ grey selection ）	Red dots, one dot represents a blank.	Yes
full-width spaces	abcd（ blank, grey selection ）	Blank, difficult to check.	No
half-width semicolons	abcd（ grey selection ）	As shown on the left.	Yes
full-width semicolons	abcd（ grey selection ）	As shown on the left.	No

- Switch encoding：Encoding→Character Set
  - You can switch encodings to help troubleshoot problems.

- notepad++ standard troubleshooting steps
  i. Confirm the encoding used
    - ANSI、UTF-8：skip to the second point.
    - Traditional Chinese encoding (Big5) and Simplified Chinese encoding (GB2312): It is recommended to switch between the two encodings, because if there are full-shaped symbols in the file, the displayed content may change.
    Display differences for encoding switching
    Traditional Chinese→Simplified Chinese

Name	Traditional Chinese encoding	Simplified Chinese encoding	Supported
full-width spaces	(blank, grey selection)	(blank, grey selection)	No
full-width semicolons	(grey selection)	(blank, grey selection)	No

Simplified Chinese→Traditional Chinese

Name	Simplified Chinese encoding	Traditional Chinese encoding	Supported
full-width spaces	(blank, grey selection)	(grey selection)	No
full-width semicolons	(grey selection)	(grey selection)	No

ii. Check for unsupported characters
- Full-width space/semicolon suggested troubleshooting method
  a. Click Search → Find
  b. Enter full-width space/semicolon in Find what
  c. Click Find All in Current Document
  d. If found, it will appear in the search results
  e. Modified to support characters

26.4 MACRO support characters

- Supported characters (only ASCII characters are supported)

Printable characters	Number	1	2	3	4	5	6	7	8	9	0															
	English alphabet 0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
	Other (half-width)	(space)	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	:								
		;	<	>	=	?	@	[	]	\	^	_	`	{	}	|	~									

SYNTEC

Control code chart	Unicode Control	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	DLE	
		DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	DEL		

For details of ASCII characters, please refer to：ASCII