MACRO開發應用手冊

匯出日期：2025-07-07
修改日期：2025-02-02

1 前言

為增加控制器應用彈性，新代控制器提供MACRO程式編輯功能。

當加工程式被宣告成MACRO格式後，該檔案將如同一般程式語言，可使用特定數學函數，如此一來，除了原本即有的移動和補償指令功能，更擁有邏輯判斷及數學演算功能。

2 檔案格式

程式內容第一行需使用"%"宣告為標題行，並加入關鍵字"@MACRO"，才會被視為 MACRO 格式檔處理。否則該檔案將被視為ISO格式檔處理。

ISO 格式檔與 MACRO 格式檔的程式內容解讀差異如下

<table>
<tr><th>ISO格式檔</th><th>MACRO格式檔</th></tr>
<tr><td>不支援 MACRO 語法使用
(有部分 MACRO 語法無法被正確解讀，可能會跳出"COM-003 控制器解譯巨集時發現程式句法有誤"的警報)</td><td>可以使用 MACRO 語法的完整功能。</td></tr>
<tr><td>每一行結尾加或不加分號";"皆可。</td><td>1. 除了特定語法外(請參考語法說明內容)，每一行結尾都要加上分號";"。
2. 該行沒有加上分號時，會和下一行合並處理。若沒有檢查出警報，仍會正常執行；若有檢查出來，會跳出 COM-008 子句中沒有結束的符號';'。
範例:
<table>
<tr><th>原加工程式</th><th>等效程式</th><th>說明</th></tr>
<tr><td>%@MACRO
#1= SIN(100) // 漏加";"
G01 Y100.;
M99;</td><td>%@MACRO
#1= SIN(100)G01 Y100.;
M99;</td><td>由於合並後指令不合法
會跳出 COM-008</td></tr>
<tr><td>%@MACRO
G01 X100. // 漏加";"
G01 Y100.;
M99;</td><td>%@MACRO
G01 X100.G01 Y100.;
M99;</td><td>由於合並後指令可直接執行
所以不會跳出警報
會直接走到 X100. Y100.</td></tr>
</table></td></tr>
<tr><td>使用到"(...)"，刮號內的內容...，會視為注解省略。</td><td>使用到"(* ... *)"，刮號內的內容...，會視為注解省略。</td></tr>
<tr><td>依照 PR3201 車床程式所設定的操作習慣書寫。</td><td>只能以車床習慣C-type書寫。</td></tr>
</table>

備註:

1. 被當作副程式（子程式）或巨集呼叫的加工檔，建議不要使用多軸群檔案（含軸群辨識符號$1、$2）。若有此應用情境，多軸群副程式檔案必須小於60KB(60000bytes)，否則控制器將報警COR-203 加工程式格式不符；單軸群副程式則不在此限。

2. 不支援MACRO格式的多軸群（含軸群辨識符號$1、$2）的副程式。
3. 檔案若超過60KB(60000bytes)
    a. 10.120.32及其以後的軟體版本：支援IF、CASE、REPEAT、FOR、WHILE等等有範圍區間的語法。
        i. 執行加工前建議執行圖形模擬進行語法檢查。
        ii. 語法範圍區間過大時，在加工時有可能會使軸向降速。可查看診斷變數D320「解譯串列單節數」及D376「運動規劃減速次數」來判斷是否為語法範圍區間過大造成軸向降速。

代碼區塊

```
%@MACRO

G01 X30. Y30.;

#1 := 1;
IF #1 = 2 THEN

// ====================
// IF條件不成立
// 此處加工程式不執行
// ====================

END_IF;

G01 X100. Y100.;     // 執行此行前，軸向可能產生降速
```

    b. 10.120.32之前的軟體版本：無法支援IF、CASE、REPEAT、FOR、WHILE等等有範圍區間的語法，否則會報警COR-204 檔案太大。

4. 除了程式注解、使用字串做為引數的Macro語法以外，僅允許以ASCII撰寫加工程式。若使用者在加工程式中撰寫了ASCII以外的字元，將會跳警報COM-027 無效的字元。
5. 支援 CR ("\r")、LF ("\n")、CR LF ("\r\n") 三種換行方式。

3 指令格式Block Format

單行動作控制指令的撰寫格式敘述如下:

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

/	單節選擇性跳躍功能，需配合PLC的C41；不支援函數語法及變數運算
N	單節次序碼，必須撰寫在該單節的第一碼位置，且不可在同一行後面撰寫MACRO指令或變數指定
G	功能指定碼，需撰寫在N碼之後
X	X軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
Y	Y軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
Z	Z軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
A	A軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
B	B軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
C	C軸的移動命令，或是擴充G碼的引數，需撰寫在G碼後
I	X方向的的半徑命令，或是擴充G碼的引數，需撰寫在G碼後
J	Y方向的的半徑命令，或是擴充G碼的引數，需撰寫在G碼後
K	Z方向的的半徑命令，或是擴充G碼的引數，需撰寫在G碼後
F	單節進給速度，或是擴充G碼的引數
S	主軸旋轉速度，或是擴充G碼的引數
T	刀具選擇功能，或是擴充G碼的引數
D	刀具補償功能，或是擴充G碼的引數
M	輔助功能，或是擴充G碼的引數

核心解譯處理順序（1.最先 ~ 10.最後）：

1. 模態G碼（G15、G17、G70等）、擴充G碼巨集（G73、G84等）
2. M碼巨集、T碼巨集
3. S碼
4. F碼
5. H碼
6. D碼
7. T碼
8. M碼
9. B碼
10. 插值G碼（G0、G1等）、功能G碼（G4、G51、G68等）

備註:

1. 其餘未說明之指令格式由相關G碼以引數形式帶入
2. 一般在副程式中會使用"GETARG"函數來讀取引數內容，而在主程式（父程式）中可使用之引數形式規則如下:
   a. 使用引數D、E、H、I、J、K、L、M、P、Q、R、T，僅能使用單一符號傳入引數，例如「G101 X30. Y40. D50. ;」，若在其後附帶數字將引發警報，例如「G101 X30. Y40. D1=50. ;」
   b. 使用引數A、B、C、F、S、U、V、W、X、Y、Z，除了使用單一符號傳入引數，亦可在其後附帶數字，例如「G101 X30. Y40. Z1=50. ;」
   c. 使用引數時，若在同一單節下復數個相同的引數，解譯時會使用最後一個引數值來執行。
   例如:

```
// 範例一，引數直接為軸向情形
G00 X10. X100.;  // 此時實際執行G00 X100.
G01 X20. X-100.; // 此時實際執行G01 X-100.


// 範例二，引數會被帶入G碼或Macro
G02 X60. C50. I-30 I20 J60 // 此時實際執行G02 X60. C50 I20 J60
G100 A10. B20. B50. C20.;  // 此時實際執行G100 A10. B50. C20.

// 範例三，功能G碼
G10 L1000 L1500 P1 P2 Q1; // 此時實際執行G10 L1500 P2 Q1
```

   d. 使用引數亦可在其後進行數值或存值為數值之變數運算，運算式的括號可省略。
   例如:

```
#1 := 3.0;

G01 X(1.0+3.0) // 數值運算，執行的結果為G01 X4.0
G01 X(4.0*#1)  // 存值為數值之變數運算，執行的結果為G01 X12.0
G01 X30.0/2.0  // 運算式的括號可省略，執行的結果為G01 X15.0
```

   e. 以上所有動作指令之後，僅可使用數值或存值為數值之變數，否則可能因程式解譯之編碼限制而造成系統錯誤，此誤用情況不在控制器的保護範圍內。

4 運算子（Operator）

運算子	符號	執行順序
括號	( ) [ ]	1
函數賦值	Identifier (引數)	2
負號	-	3
補數	NOT	3
乘號	*	4
除號	/	4
模數（餘數）	MOD	4
加號	+	5
減號	-	5
比較	<,>,<=,>=	6
等於	=	7
不等於	<>	8
布林運算"且"	&AND,	9
布林運算"互斥"	XOR	10
布林運算"或"	OR	11

注1:

使用『/』元件（除號）時，需注意若分子與分母都是整數，所得結果仍為整數。整數及非整數之差別在有無加入小數點。

範例:

- 分子為非整數：1. / 2 = 0.5
- 分母為非整數：1/ 2.0 = 0.5
- 分子、分母均為整數：1/2 = 0

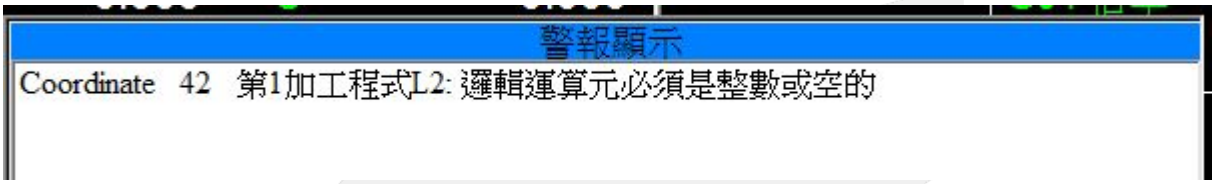
- 刮號內均為整數：(1/2)*1.0 = 0

注2:

MOD運算子（模數）僅適用於數值型態為Long，若數值型態為Double，則會出現下述警報訊息。

範例:

```
%@MACRO
@1:= 4. MOD 3;
M99;
```



5 語法說明

5.1 變數指定

變數指定	
語法	**<變數>** := **<敘述>**;
說明	指定變數內容
範例一 **直接定值**	

```
@1 := 123;
#1 := 456;

AR1 := 789;      // App 區域變數
MAR1 := 789;     // App 永久儲存變數

#10 := "12";     // 區域變數#10內容為12
@10 := "12";     // 公用變數@10內容為12849
```

範例二 **間接定值**	

```
#1:= 123;

@[#1] := 567;        // @123=567
@[#1+7]:=890;        // @130=890

AR[#1] := 789;       // AR123=789
MAR[#1] := 789;      // MAR123=789
```

備註	1. 範例一中的"12"為字串寫法，表示要將字串存入變數中，只有存入公用變數時，控制器會先進行ASCII轉碼，區域變數則不會。 2. 承上，欲正確讀取公用變數所儲存的字串內容，請使用**SCANTEXT函數**。 3. 範例二中請留意用的是"中刮號" 4. App變數(AR、MAR)使用限制: a. 支援版本：10.118.39及之後版本。 b. 只有APP專用的Macro才支援存取AR、MAR。若在其他加工程式呼叫，則跳警報COR-016。 c. 若欲存取的變數超出範圍，例如只有512個MAR卻下MAR512 := 1;，則跳警報COR-016。(AR、MAR變數號碼為0-Based) d. 直接存取負的變數位址，例如MAR-1 := 1;，則跳警報COM-003。 e. 直接存取小數的變數位址，例如MAR1.1 := 1;，則跳警報COM-003。 f. 間接存取負的變數位址，例如MAR[-1] := 1;，則跳警報COR-016。 g. 間接存取小數的變數位址，例如MAR[1.1] := 1;，則跳警報COR-016。 h. AR、MAR皆不支援存字串。 i. 僅支援Macro格式，不支援ISO格式。

5.2 GOTO

GOTO	
語法	**GOTO n;**
說明	此語法需配合單節次序碼使用，可跳到指定的N行號執行其內容，假設程式中同時存在兩個N行號，則以該程式中第一個N行號為準。

範例	%@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;

備註	使用REPEAT/WHILE/FOR/GOTO等回圈功能時，應謹慎注意無窮回圈問題，當此問題發生時，會造成人機畫面鎖死或加工程式卡死。 建議在回圈中適時加入**SLEEP()函數**，若不小心進入無窮回圈，仍有機會操作人機畫面以中止程式執行。

5.3 CASE

CASE	
語法	**CASE** <條件變數> **OF** **<變數>:** <陳述列表> **<變數>,<變數>:** <陳述列表> **<變數>,<變數>,<變數>:** <陳述列表> **ELSE** <陳述列表> **END_CASE;**
說明	多條件判斷，根據條件變數內容，分別執行不同程式區塊。請注意"**變數**"內容需為大於等於0之**整數**型態。
範例	%@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;

備註

- 當未加 END_CASE; 時，將報 COM-023【**CASE巨集子句找不到' ELSE '或' END_CASE '**】或 COM-024【**CASE巨集子句找不到' END_CASE '**】
    - COM-023

```
%@MACRO
#1 := 1;
G01 G90 X0. Y0. F1000;
CASE #1 OF
1:
    X(1.0*#1) Y(1.0*#1);

// 未加 END_CASE

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
// 搜尋到程式結尾仍未找到 END_CASE，於此行跳 COM-023
```

    - COM-024

```
%@MACRO
#1 := 1;
G01 G90 X0. Y0. F1000;
CASE #1 OF
1:
     X(1.0*#1) Y(1.0*#1);
ELSE
     X(4.0*#1) Y(4.0*#1);

// 未加 END_CASE

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
// 搜尋到程式結尾仍未找到 END_CASE，於此行跳 COM-024
```

- 10.120.32 及其以後的軟體版本，當遺漏"CASE....OF"的語句時，將報 COM-022【**CASE巨集子句找不到' CASE '或' OF '**】

代碼區塊

```
%@MACRO
#1 := 1;
G01 G90 X0. Y0. F1000;

// 未加CASE...OF語句
1:                              // 至此處發現缺少CASE...OF，於此行跳 COM-022
     X(1.0*#1) Y(1.0*#1);
ELSE
     X(4.0*#1) Y(4.0*#1);
END_CASE;

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
```

- 10.120.32 及其以後的軟體版本，當遺漏包含": "的條件語句，將報 COM-025 **【CASE巨集子句找不到':'或','】**或 COM-003 **【句法錯誤】**
  - COM-025

    代碼區塊

    ```
    %@MACRO
    #1 := 1;
    G01 G90 X0. Y0. F1000;
    CASE #1 OF
    // 未加": "的條件語句
    END_CASE;        // 至此處發現缺少": "的條件語句，於此行跳 COM-025

    G01 Z10;
    G01 X10. Y10.;
    X0. Y0.;
    M30;
    ```

  - COM-003

    代碼

    ```
    %@MACRO
    ```

```
#1 := 1;
G01 G90 X0. Y0. F1000;
CASE #1 OF
// 未加": "的條件語句
ELSE                          // 至此處發現缺少": "的條件語句，於此行跳
 COM-003
    X(4.0*#1) Y(4.0*#1);
END_CASE;

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
```

- 10.120.32 及其以後的軟體版本，一個加工檔最多只能包含 256 組CASE巨集，超過時將報 COM-029**【CASE巨集超過支援上限】**

代碼

```
%@MACRO
CASE #1 OF          // 第1組CASE巨集
1:
    G91 G01 X10.;
END_CASE;
CASE #2 OF          // 第2組CASE巨集
1:
    G91 G01 X20.;
END_CASE;

...

CASE #256 OF        // 第256組CASE巨集
1:
    G91 G01 X20.;
END_CASE;
CASE #256 OF        // 第257組CASE巨集，於此行跳 COM-029
1:
    G91 G01 X20.;
END_CASE;

M30;
```

- 10.120.32 及其以後的軟體版本，一組CASE巨集內最多包含16個": "的條件語句，超過時將報 COM-005**【數學式太復雜】**

代碼區塊

```
%@MACRO
#1 := 1;

CASE #1 OF
1:              // 第1個": "條件語句
    G01 X10.;
2:              // 第2個": "條件語句
    G01 X20.;
3:              // 第3個": "條件語句
    G01 X30.;

...

16:             // 第16個": "條件語句
    G01 X160.;
17:             // 第17個": "條件語句, 於此行跳 COM-005
    G01 X170.;
ELSE
    G01 X0.;
END_CASE;

M30;
```

5.4 IF

IF	

語法	**IF** <條件> **THEN**
	<陳述列表>
	ELSEIF <條件> **THEN**
	<陳述列表>
	ELSE
	<陳述列表>
	END_IF;

說明	IF條件判斷

範例	%@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;

備註

- 當未加 END_IF; 時，將報 COM-020 **【IF巨集子句找不到' ELSE '或' END_IF '】**或 COM-021 **【IF巨集子句找不到' END_IF '】**
  - COM-020

```
%@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);

// 未加 END_IF

M30;
// 搜尋到程式結尾仍未找到 END_IF，於此行跳 COM-020
```

  - COM-021

```
%@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);

// 未加 END_IF

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
// 搜尋到程式結尾仍未找到 END_IF，於此行跳 COM-021
```

- 10.120.32 及其以後的軟體版本，當遺漏"IF....THEN"的語句時，將報 COM-019 **【IF巨集子句找不到' IF '或' THEN '】**或 COM-003 **【句法錯誤】**
  - COM-019

代碼區塊

```
%@MACRO
#1 := 3.0;
G01 G90 X0. Y0. F1000;

// 遺漏"IF...THEN"語句
    X(1.0*#1) Y(1.0*#1);
ELSEIF #1 = 2 THEN          // 至此處發現缺少"IF...THEN"的條件語句，於此
行跳 COM-019
    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;

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
```

- COM-003

代碼區塊

```
%@MACRO
#1 := 3.0;
G01 G90 X0. Y0. F1000;

// 遺漏"IF...THEN"語句
    X(1.0*#1) Y(1.0*#1);
ELSE                        // 至此處發現缺少"IF...THEN"的條件語句，於此
行跳 COM-003
    X(4.0*#1) Y(4.0*#1);
END_IF;

G01 Z10;
G01 X10. Y10.;
X0. Y0.;
M30;
```

- 10.120.32 及其以後的軟體版本，一個加工檔最多只能包含 256 組IF巨集，超過時將報 COM-030 **【IF巨集超過支援上限】**

代碼區塊

```
%@MACRO
IF #1 = 1 THEN       // 第1組IF巨集
    G01 X10.;
END_IF;
IF #1 = 2 THEN       // 第2組IF巨集
    G01 X10.;
END_IF;

...

IF #1 = 256 THEN     // 第256組IF巨集
    G01 X10.;
END_IF;
IF #1 = 257 THEN     // 第257組IF巨集，於此行跳 COM-030
    G01 X10.;
END_IF;

M30;
```

- 10.120.32 及其以後的軟體版本，一組IF巨集內最多包含16個"ELSEIF...THEN"的語句，超過時將報 COM-005 **【數學式太復雜】**

代碼區塊

```
%@MACRO
#1 := 1;
IF #1 = 0 THEN
    G01 X0.;
ELSEIF #1 = 1 THEN       // 第1個"ELSEIF...THEN"語句
    G01 X10.;
ELSEIF #1 = 2 THEN       // 第2個"ELSEIF...THEN"語句
    G01 X20.;

...

ELSEIF #1 = 16 THEN      // 第16個"ELSEIF...THEN"語句
    G01 X160.;
ELSEIF #1 = 17 THEN      // 第17個"ELSEIF...THEN"語句，於此行跳 COM-005
    G01 X170.;
ELSE
    G01 X180.;
END_IF;
```

```
M30;
```

5.5 REPEAT

REPEAT	
語法	**REPEAT** <陳述列表> **UNTIL** <條件> **END_REPEAT**;
說明	REPEAT回圈控制
範例	%@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;

備註

- 使用REPEAT/WHILE/FOR/GOTO等回圈功能時，應謹慎注意無窮回圈問題，當此問題發生時，會造成人機畫面鎖死或加工程式卡死。
  建議在回圈中適時加入**SLEEP()函數**，若不小心進入無窮回圈，仍有機會操作人機畫面以中止程式執行。

- 當未加 END_REPEAT 時，將報 COM-016【**REPEAT巨集子句找不到' END_REPEAT '**】

```
%@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) // 未加 END_REPEAT;


G01 Z10; // 搜尋下個單節仍未找到 END_REPEAT，於此行跳 COM-016
G01 X10. Y10.;
X0. Y0.;
M30;
```

- 10.120.32 及其以後的軟體版本，當遺漏"REPEAT"時，將報 COM-034【**REPEAT巨集子句找不到'REPEAT'**】

代碼區塊

```
%@MACRO

#1 := 10;

// 未加"REPEAT"
G01 X0. Y0.;
G01 X10. Y10.;
G01 X10. Y-10.;
G01 X-10. Y-10.;
G01 X-10. Y10.;
```

```
#1 := #1 - 1;
UNTIL #1 = 0 END_REPEAT;     // 至此處發現缺少"REPEAT"，於此行跳 COM-034

M30;
```

- 10.120.32 及其以後的軟體版本，當遺漏"UNTIL...."語句時，將報 COM-015**【REPEAT巨集子句找不到' UNTIL '】**

代碼區塊

```
%@MACRO

#1 := 10;
REPEAT
G01 X0. Y0.;
G01 X10. Y10.;
G01 X10. Y-10.;
G01 X-10. Y-10.;
G01 X-10. Y10.;
#1 := #1 - 1;
// 遺漏"UNTIL..."語句
END_REPEAT;     // 至此處發現缺少"UNTIL..."語句，於此行跳 COM-015

M30;
```

- 10.120.32 及其以後的軟體版本，一個加工檔最多只能包含 256 組REPEAT巨集，超過時將報 COM-031**【REPEAT巨集超過支援上限】**

代碼區塊

```
%@MACRO

#1 := 10;
REPEAT                        // 第1組REPEAT巨集
G91 G01 X10.;
UNTIL #1 = 0 END_REPEAT;
#1 := 10;
REPEAT                        // 第2組REPEAT巨集
G91 G01 X10.;
UNTIL #1 = 0 END_REPEAT;

...

#1 := 10;
```

```
REPEAT                          // 第256組REPEAT巨集
G91 G01 X10.;
UNTIL #1 = 0 END_REPEAT;
#1 := 10;
REPEAT                          // 第257組REPEAT巨集，於此行跳 COM-031
G91 G01 X10.;
UNTIL #1 = 0 END_REPEAT;

M30;
```

5.6 WHILE

WHILE	
語法	**WHILE** <條件> **DO** <陳述列表> **END_WHILE**;
說明	WHILE回圈控制
範例	%@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;

備註

- 使用REPEAT/WHILE/FOR/GOTO等回圈功能時，應謹慎注意無窮回圈問題，當此問題發生時，會造成人機畫面鎖死或加工程式卡死。
  建議在回圈中適時加入**SLEEP()函數**，若不小心進入無窮回圈，仍有機會操作人機畫面以中止程式執行。

- 當未加 END_WHILE 時，將報 COM-018 **【WHILE巨集子句找不到' END_WHILE '】**

```
%@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;

// 未加 END_WHILE;

G01 X10. Y10.;
X0. Y0.;
M30;
// 搜尋到程式結尾仍未找到 END_WHILE，於此行跳 COM-018
```

- 10.120.32 及其以後的軟體版本，當遺漏"WHILE...DO"語句時，將報 COM-017 **【WHILE巨集子句找不到' WHILE '或' DO '】**

代碼區塊

```
%@MACRO
#1 := 10;

// 未加"WHILE...DO"語句
G01 X0. Y0.;
G01 X10. Y10.;
G01 X10. Y-10.;
G01 X-10. Y-10.;
G01 X-10. Y10.;
#1 := #1 - 1;
```

```
END_WHILE;          // 至此處發現缺少"WHILE...DO"語句，於此行跳 COM-017

M30;
```

- 10.120.32 及其以後的軟體版本，一個加工檔最多只能包含 256 組WHILE巨集，超過時將報 COM-032 **【WHILE巨集超過支援上限】**

代碼區塊

```
%@MACRO
#1 := 10;
WHILE #1 > 0 DO     // 第1組WHILE巨集
G91 G01 X10.;
#1 := #1 - 1;
END_WHILE;
WHILE #1 > 0 DO     // 第2組WHILE巨集
G91 G01 X10.;
#1 := #1 - 1;
END_WHILE;

...

WHILE #1 > 0 DO     // 第256組WHILE巨集
G91 G01 X10.;
#1 := #1 - 1;
END_WHILE;
WHILE #1 > 0 DO     // 第257組WHILE巨集，於此行跳 COM-032
G91 G01 X10.;
#1 := #1 - 1;
END_WHILE;

M30;
```

5.7 FOR

FOR

語法	**FOR** <變數1> := 敘述1> **TO** <敘述2> **BY** <敘述3> **DO** <陳述列表> **END_FOR;** 變數1：控制回圈次數的變數 敘述1：回圈計數的起始次數，可為數值或運算式 敘述2：回圈計數的終止次數，可為數值或運算式 敘述3：回圈計數每次的累加次數，可為數值或運算式 陳述列表：每次回圈之執行內容
說明	FOR回圈控制
範例	%@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;

備註

- 使用REPEAT/WHILE/FOR/GOTO等回圈功能時，應謹慎注意無窮回圈問題，當此問題發生時，會造成人機畫面鎖死或加工程式卡死。
  建議在回圈中適時加入**SLEEP()函數**，若不小心進入無窮回圈，仍有機會操作人機畫面以中止程式執行。

- 請勿在 FOR 回圈內使用會跳轉進出回圈的命令，例如復式切削循環(G72-G78)、使用GOTO 跳轉出回圈再跳轉回來、M98 H_，此會造成回圈的累加次數(<敘述3>)數值異常
  a. 導致執行異常範例

```
// FOR 執行時會變為每次累加 5
%@MACRO
FOR #10:=1 TO 100 BY 1 DO
    GOTO 12;
    N13;
END_FOR;
M30;

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

- 當未加 END_FOR 時，將報 COM-014 **【FOR巨集子句找不到' END_FOR '】**

```
%@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;

// 未加 END_FOR

G01 Z10;
G01 X10. Y10.;
X0. Y0.;M30;
```

```
// 搜尋到程式結尾仍未找到 END_FOR，於此行跳 COM-014
```

- 10.120.32 及其以後的軟體版本，當遺漏"FOR...DO"語句時，將報 COM-013 **【FOR巨集子句找不到' FOR '或' DO '】**

代碼區塊

```
%@MACRO
// 未加"FOR...DO"語句
G01 X0. Y0.;
G01 X10. Y10.;
G01 X10. Y-10.;
G01 X-10. Y-10.;
G01 X-10. Y10.;
END_FOR;              // 至此處發現缺少"FOR...DO"語句，於此行跳 COM-013

M30;
```

- 10.120.32 及其以後的軟體版本，一個加工檔最多只能包含 256 組FOR巨集，超過時將報 COM-033 **【FOR巨集超過支援上限】**

代碼區塊

```
%@MACRO
FOR #1 = 1 TO 10 DO          // 第1組FOR巨集
G91 G01 X10.;
END_FOR;
FOR #2 = 1 TO 10 DO          // 第2組WHILE巨集
G91 G01 X10.;
END_FOR;

...

FOR #256 = 1 TO 10 DO        // 第256組WHILE巨集
G91 G01 X10.;
END_FOR;
FOR #257 = 1 TO 10 DO        // 第257組WHILE巨集，於此行跳 COM-033
G91 G01 X10.;
END_FOR;

M30;
```

5.8 EXIT

EXIT	
語法	**EXIT;**
說明	中斷回圈，跳離回圈控制

範例	%@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 程式注解

程式注解	
語法	**(*** <陳述列表> ***)** **//** <陳述列表>
說明	程式注解（Comment）

範例一
單行注解
	%@MACRO
	G00 G90 X0. Y0.; // 移動回原點
	M30;

範例二 **區塊注解**	%@MACRO (* 此區塊為注解區 不管內容為何均不影響程式執行 *) G00 G90 X0. Y0.; G00 G90 X10. Y0.; G00 G90 X10. Y10.; G00 G90 X0. Y10.; G00 G90 X0. Y0.; M30;
備註	若於符合注解語法以外的陳述列表區域添加屬於注解之文字，可能會因解譯編碼之限製造成系統錯誤，此誤用情況不在控制器的保護範圍內。

5.10 軸群辨識符號

程式執行範圍	
語法	$1 軸群一加工程式... $2 軸群二加工程式....
說明	1. 僅適用於Pr732 設定為0或1，非軸群獨立下，同份加工檔中用來區分不同軸群的執行程式內容。 2. 軸群辨識符號$1、$2、$3、$4，不可置於兩個%符號(程式執行範圍)之外，否則將報警COR-210。
範例一(合法)	% $1 軸群一加工程式... $2 軸群二加工程式.... %

範例二 (不合法)	$1 % 軸群一加工程式... % $2 % 軸群二加工程式.... %
範例三 (不合法)	$1 % 軸群一加工程式... % $2 軸群二加工程式....

5.11 程式執行範圍

程式執行範圍	
語法	% 加工程式 %
說明	1. 程式內容中使用到"% ... %"，兩個%符號內的加工程式，會被系統執行；兩個%符號外(包含第一個%符號前面和第二個%符號後面)的加工程式，系統會予以忽略。 2. 程式內容中僅使用了一個%符號，則從第一個%符號開始執行到檔案結束。 3. 程式內容中僅有前2個%符號有效，若第2個%符號後還存在%字元，視同一般字元忽略。 4. 軸群辨識符號$1、$2、$3、$4，不可置於兩個%符號(程式執行範圍)之外，否則將報警COR-210。 5. 此功能不支援超過60KB(60000bytes)以上的檔案且不支援於 DNC 傳檔時使用。
範例一	G91 G00 X10. % G91 G00 Y10. % G91 G00 Z10. M30 //執行後可以發現只有Y軸會移動10.，其他軸向不動。

範例二	G91 G00 X10. % G91 G00 Y10. G91 G00 Z10. M30 //執行後可以發現Y、Z軸都會移動10.，而X軸保持不動。

範例三	// 無效
	%
	// 有效
	%
	// 無效
	% // 無效
	// 無效

5.12 巢狀語法

巢狀語法	
語法	**<語法1開頭>** **<語法2開頭>** **<語法2結尾>** **<語法1結尾>**
說明	CASE/IF/REPEAT/WHILE/FOR語法內可以再加入另一層語法，以處理多層流程控制，上限為 10 層 如超過10層將跳出COM-007**【重覆迴圈太深】**

範例	%@MACRO
	FOR #1 := 0 TO 3 BY 1.0 DO
	  FOR #2 := 0 TO 3 BY 1.0 DO
	  G01 X(#1) Y(#2);
	  END_FOR;
	END_FOR;
	M30;

備註	建議撰寫時，使用空格或 tab 做縮排，增加程式可讀性

6 路徑擴充引數語法

控制器目前支援",C_"、",R_"、",A_"等圓角轉角擴充語法，詳見轉角倒角，轉角圓角，直線角度（C、R、A）(C-Type)。

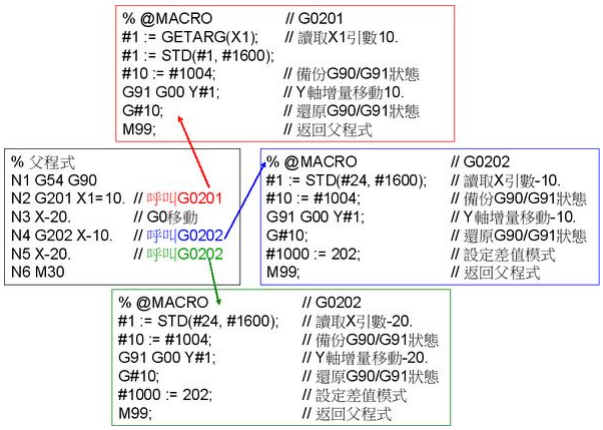
若使用不支援的擴充引數，例如",Z_"，則會觸發 COR-034 **【路徑擴充引數不存在】**警報。以下程式將觸發 COR-034 警報。

以下程式將觸發COR-034警報

```
1  G91 G01 X100. Y100, Z100;
```

7 MACRO讀取/處理流程

圖示	說明

	以下依序解釋父程式（主程式）每一行的動作：（左方框框內的程式）
	• N1：設定座標系為G54，並采絕對模式G90移動
	• N2：呼叫G0201巨集，並透過GETARG函數將X1引數帶入
	  • 進入G0201後，將X1引數儲存於區域變數#1
	  • 以#10備份G90/G91狀態
	  • Y軸以G00增量移動10mm
	  • 還原G90/G91狀態
	  • 返回父程式
	• N3：由於離開G0201前最後一個插值模式為G00，故此單節X軸仍以G00移動
	• N4：呼叫G0202巨集，並透過#24將X引數帶入
	  • 進入G0202後，將X引數儲存於區域變數#1
	  • 以#10備份G90/G91狀態
	  • Y軸以G00增量移動-10mm
	  • 還原G90/G91狀態
	  • 設定插值模式為202
	  • 返回父程式
	• N5：由於離開G0202前將插值模式儲存為202，故執行此單節時，系統將再次呼叫G0202
	• N6：程式結束

8 MACRO撰寫注意事項

- MACRO內建議多使用區域變數（Local Variables，#1 ~ #400），除非有跨MACRO的需求才使用公用變數（Global Variables，@1 ~ @165535）。
- 執行MACRO時，使用者的資料是透過引數（A_、B_、…,、Z_、X1=、*Y1*=、…）傳入，引數與區域變數相通，下表為引數與區域變數之關系。
- 針對擴充之引數位址，例如X1引數，請使用GETARG函數來讀取其數值。

引數	對應變數	引數	對應變數	引數	對應變數
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)

- 模式變數（Modal Variables，#2001 ~ #2100、#3001 ~ #3100）在系統重置時，會回復成VACANT狀態，因此可應用於多個MACRO間進行資料交換之時機，以節省變數資源之使用。
- MACRO若需要內定初始值，可多加利用使用者參數（Customer Parameter，#4001 ~ #4100、#5001 ~ #5100）。
- 執行MACRO副程式（子程式）時，若模式G碼會被改變（G91/G90、G40/G41/G42,…等），請在一進入MACRO時，即備份當前的狀態，待離開MACRO前再回復原有狀態。
- 若想在離開MACRO後，繼續保留此MACRO之插值模式（#1000），建議在離開MACRO前，將#1000指定為該MACRO之號碼。爾後只要是軸向位移之指令單節，系統將自動呼叫此MACRO，而不用再次指定。
    - 插值模式在遇到G00/G01/G02/G03/G31/G33或#1000內容變動時，將被自動改寫。
- 對於長度或角度的引數，在運算前請使用STD函數將單位標準化，以符合工具機使用習慣。
- 不可改變座標系統設定，像是G92/G54/G52等與座標系統相關之指令，否則圖形模擬功能將失去參考意義。
- 執行加工時，核心會預解MACRO內容，因此MACRO執行速度會超前G/M碼指令，若變數指定或資料讀取需與G/M碼動作同步，請於變數指定或資料讀取前加入WAIT函數，以確保動作正確。
  注意：WAIT 函數目的是為了確保在運動指令執行完畢前，就因預解而提早讀寫相關數值所造成的錯誤，因此，其只確保 WAIT 前的 G/M 碼執行完畢前不會繼續往後預解 (其中 M98、M99、M198 三者為例外)，對其他指令則不會有作用。
- MACRO程式最後需加"M99;"，才能返回主程式（父程式）。
- 請養成良好習慣，在程式中多加入注解，增加程式可讀性，幫助後續維護及問題排除。

8.1 登錄G碼巨集

- 開發人員可以根據機台需求來新增標準G碼以外的G碼巨集，也可以客製標準G碼的內容。
- 透過【Pr3701~3710 登錄G碼呼叫巨集】設定，登錄想要客製的標準G碼，當程式中執行到對應的G碼時，將不再執行標準G碼之動作，而是執行客製G碼巨集之內容。
- 以下是【Pr3701~3710 登錄G碼呼叫巨集】設定值與目前開放客製的標準G碼對照表

Pr3701	標準G碼	G碼巨集檔案名稱
0	無	無
-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

- 以下是登錄G碼巨集之運作規格
    - 巨集特性視同G碼巨集特性
    - 登錄G碼巨集程式內的所有G碼，均是標準G碼
    - 標準G碼中為插值模式的G碼（例如G00、G01、G02、G03）在登錄客製G碼後，仍然具有繼承功能
        - 例如執行
          G00 X100.

Y100.
其中Y100.也會執行G00巨集，並且會讀取占用Y引數

- 若登錄巨集中，有改變插值模式(#1000)，則務必在離開登錄巨集前，將插值模式還原
  - 例如登錄G00為登錄G碼巨集
    在G0000巨集中，如果將插值狀態變更為G01
    則在離開G0000前，需將插值模式變更回G00
    避免離開登錄巨集之後的狀態錯亂
- 登錄G碼巨集遇到以下指令時，將無法運作
  - 車床 G7.1
  - 車床 G12.1
  - 車床 ,A
  - 車床 ,R
  - 車床 ,C
  - 車床 所有加工循環指令
  - 銑床 所有加工循環指令
  - T碼巨集

- 相容性異動

版本	異動
~Before	登錄G碼與"G碼巨集"的解譯順序相同。
10.114.50	• 登錄G碼與"標準G碼"的解譯順序相同。 • 不再支援原先在G碼巨集中將插值模式改為900000（#1000 := 900000）的特殊用法
10.116.16A、10.116.17	提供G53以自訂MACRO（G0053）取代
10.118.22F、10.118.26	提供G40、G41、G42以自訂MACRO（G0040、G0041、G0042）取代
10.118.45	提供G43、G44、G49以自訂MACRO（G0043、G0044、G0049）取代
10.118.52D, 10.118.54	提供G04以自訂MACRO（G0004）取代

9 函數表（Function List）

功能	說明

ABS	計算某數值的絕對值
	EX:
	#10 := -1.1;
	#1 := ABS(#10); // #1 = 1.1
	#2 := ABS(-1.2); // #2 = 1.2

ACOS	計算某數值的反餘弦值
	EX:
	#10 := 1;
	#1 := ACOS(#10); // #1 = 0
	#2 := ACOS(-1); // #2 = 180

ALARM	觸發巨集警報
	EX:
	ALARM(300); // 觸發巨集第300號警報
	ALARM(301, "ALARM 301 Content");
	備註:
	1. 會伴隨觸發警報COR-027【巨集發出警報】。
	2. 警報內容有字串長度限制，中文 : 19個字，英文 : 39個字。
	3. 警報ID只允許使用0~65535之間的整數，否則跳警報COR-023【語意錯誤】。

ASIN	計算某數值的反正弦值
	EX:
	#10 := 1;
	#1 := ASIN(#10); // #1 = 90
	#2= ASIN(-1); // #2 = -90

ATAN	計算某數值的反正切值。計算角度範圍介於±90°
	EX:
	#10 := 1;
	#1 := ATAN(#10); // #1 = 45
	#2 := ATAN(-1); // #2 = -45

功能	說明

ATAN2(Y, X)	計算Y/X的反正切值，並根據輸入引數(X, Y)所處的象限位置決定輸出結果。計算角度範圍介於±180°
	EX:
	#10 := 1;
	#20 := -1
	#1 := ATAN2(#10, #20); // #1 = 135
	#2 := ATAN2(#20, #10); // #2 = -45
	#3 := ATAN2(1, 0); // #3 = 90
	注意事項:
	1. 有效版本：10.118.29W, 10.118.40C, 10.118.42
	2. 引數X和引數Y必須為數字，否則會跳警報COR-023 【語意錯誤】
	3. 引數X和引數Y不可同時為0，否則會跳警報COR-004 【運算域錯誤】
	錯誤範例:
	@1 := ATAN2("1", 1); // 引數非數字，跳警報COR-023
	@2 := ATAN2(0, 0); // 引數皆為0，跳警報COR-004

AXID	查詢軸名稱所對應的軸編號，當該軸名稱不存在時，回傳值為空白(VACANT, #0)
	EX:
	假設第六軸名稱為Y2（Pr326=202），第二軸名稱為Y（Pr322=200）
	#1 := AXID(Y); // #1 = 2
	#2 := AXID(Y2); // #2 = 6

CEIL	回傳大於或等於某數值的最小整數
	EX:
	#10 := 1.4;
	#1 := CEIL(#10); // #1 = 2
	#2 := CEIL(1.5); // #2 = 2

COS	計算某數值的餘弦值
	EX:
	#10 := 180;
	#1 := COS(#10); // #1 = -1
	#2 := COS(-180); // #2 = -1

CLOSE	關閉由OPEN函數開啟的檔案，程式結束後該檔案亦會自動關閉。在檔案關閉後，PRINT函數會失效。 EX: CLOSE(); // 關閉檔案

功能	說明

INT DBOPEN("File name")	功能:
	• 開啟既有的 Cycle 檔案
	影響:
	• 執行成功後，會指定此檔案為後續操作的目標檔案
	引數 - File name:
	• 欲開啟檔案的檔名
	回傳值:
	• 1-N：開檔成功，檔案中的 Cycle 資料個數
	• 0：開檔失敗
	  • 請確認指定路徑當中有使用的檔名
	備註:
	1. 引數型態錯誤會觸發警報 COR-023 語義錯誤
	2. 此 Macro 語法不支援圖形模擬，會固定回傳 0
	3. 指定路徑可能被客製 Action（CUSTOMFILE_CYCLE1~5）影響，請參考 CE人機客製應用文件-開放使用的Action列表 。
	4. 同時間僅能開啟一個 Cycle 檔案，而 DBOPEN 以及 DBNEW 皆會開啟 Cycle 檔案，若重復呼叫為後令蓋前令。
	Ex: 先使用 DBOPEN("FileA")，再使用 DBNEW("FileB")，此時使用其餘DB語法皆會操作到 FileB
	5. 若有重新載入檔案的需求，需先重置系統（在背景運算元件使用時請執行Bgnd Stop）後重新執行DBOPEN
	EX:
	`// 1.`
	`DBOPEN("Test.cyc");`
	`// 載入GNCFILES\\Test.cyc資料檔`
	
	`// 2.`
	`#1 := 51;`
	`DBOPEN("FLAT\\\\AB#1[3]ZZ.cyc" );`
	`// 載入FLAT\\\\AB051ZZ.cyc 資料檔，[3]表示檔名為三位有效數字`

功能	說明

BOOL DBLOAD(Index)	功能:
	• 讀取第 "**Index**" 筆的 Cycle 資料
	影響:
	• 執行成功後，會以目標位置資料的 Cycle name 作為後續操作用的資料格式
	引數 - Index:
	• 指定要讀取第幾筆資料
	• 限制: 最大值為"目前開啟的 Cycle 檔的最後位置"，ex:目前 Cycle 檔於 0~2 有 Cycle 資料，最多只能 DBLOAD(2)
	回傳值:
	• 1：讀取成功
	• 0：讀取失敗
	  • 請確認有先成功執行 DBOPEN 或 DBNEW
	  • 請確認 Index 為非負整數且未超過範圍
	備註:
	1. 引數型態錯誤會觸發警報 COR-023 語義錯誤
	2. 此 Macro 語法不支援圖形模擬，會固定回傳 0
	3. 此 Macro 語法會操作目前開啟 Cycle 檔。Cycle 檔的開啟可透過 DBOPEN 以及 DBNEW
	4. 同時間僅能指定一個 Cycle name，而 DBLOAD 以及 DBINSERT 都會指定 Cycle name，若重復呼叫為後令蓋前令
	Ex:先使用 DBLOAD(0)，Cycle name 會指定為位置"0"資料的 Cycle name，此時再呼叫 DBINSERT，Cycle name 會改為指定 DBINSERT 的引數 Cycle name
	EX:
	`DBOPEN("FLAT\\TAB01");`
	`// 載入FLAT\\TAB01資料檔`
	`DBLOAD( 0 );`
	`// 讀取第0筆資料`
	`DBLOAD( 1 );`
	`// 讀取第1筆資料`

功能	說明

BOOL DBSAVE(Index)	功能:
	• 根據目前指定的 Cycle name，覆蓋第 "**Index**" 筆的 Cycle 資料
	引數 - **Index**:
	• 指定要覆寫第幾筆資料
	• 限制: 最大值為"目前開啟的 Cycle 檔的最後位置"，ex:目前 Cycle 檔於 0~2 有 Cycle 資料，最多只能 DBSAVE(2)
	回傳值:
	• 1：覆寫成功
	• 0：覆寫失敗
	  • 請確認有先成功執行 DBOPEN 或 DBNEW
	  • 請確認有先成功執行 DBLOAD 或 DBINSERT
	  • 請確認 Index 為非負整數且未超過範圍
	備註:
	1. 引數型態錯誤會觸發警報 COR-023 語義錯誤
	2. 此 Macro 語法不支援圖形模擬，會固定回傳 0
	3. 此 Macro 語法會操作目前開啟 Cycle 檔。Cycle 檔的開啟可透過 DBOPEN 以及 DBNEW
	4. 此 Macro 語法會使用目前指定的 Cycle name 覆寫目標位置的資料。指定 Cycle name 可透過 DBLOAD 以及 DBINSERT
	5. 版本：支援10.118.39及之後版本
	EX:
	`DBOPEN( "GrinderToolTable.cyc" );  // 載入 GrinderToolTable.cyc資料檔`
	`DBLOAD( 0 );  // 讀取第0筆資料`
	`DBSAVE( 0 );  // 儲存第0筆資料`

功能	說明

INT DBNEW("File name")	功能:
	• 新增並開啟全新的 Cycle 檔案
	影響:
	• 執行成功後，會指定此檔案為後續操作的目標檔案
	引數 - **File name**:
	• 欲新增檔案的檔名
	• 限制: 字串長度最大為30個字元
	回傳值:
	• 1：新增檔案成功
	• 0：新增檔案失敗，非預期錯誤
	  • 請聯絡新代原廠
	• -1：新增檔案失敗，File name 過長
	  • 請確認 File name 長度沒有超過於 30 字元
	• -2：新增檔案失敗，指定路徑下已存在該檔名
	  • 請確認指定路徑下沒有相同的檔名
	備註:
	1. 引數型態錯誤會觸發警報 COR-023 語義錯誤
	2. 此 Macro 語法不支援圖形模擬，會固定回傳 0
	3. 指定路徑可能被客製 Action（CUSTOMFILE_CYCLE1~5）影響，請參考 CE人機客製應用文件-開放使用的Action列表
	4. 此 MACRO 語法僅會開新檔案，檔案內並無 Cycle 資料，無法直接接續使用 DBLOAD 或 DBSAVE
	5. 同時間僅能開啟一個 Cycle 檔案，而 DBOPEN 以及 DBNEW 皆會開啟 Cycle 檔案，若重復呼叫為後令蓋前令
	Ex: 先使用 DBOPEN("FileA")，再使用 DBNEW("FileB")，此時使用其餘DB語法皆會操作到 FileB
	6. 版本: 支援 10.118.56L, 10.118.60F, 10.118.66 以及以後的版本
	Ex:
	`// 執行暖機健檢`
	
	`#30 := DBNEW("WarmTest_20220621");  // 新增 20220621 的暖機健檢檔案`
	`IF( #30 = 1 )`
	`THEN  // 新增檔案成功`
	`  DBINSERT(0,"WarmTest");  // 插入 Cycle 資料`
	`ELSE`
	`  // 錯誤處理`
	`END_IF;`

功能	說明

INT DBINSERT(Index,"Cycle Name")	功能:
	• 蒐集 **Cycle name** 指定的資料，新增 Cycle 資料於第 "**Index**" 筆
	影響:
	• 執行成功後，指定位置以及後方的 Cycle 資料會往後移一筆
	• 執行成功後，會以此 Cycle name 作為後續操作用的資料格式
	引數 - **Index**:
	• 指定 Cycle 資料要插在第幾筆
	• 限制: 最大值為"目前開啟的 Cycle 檔的最後資料位置 + 1"。ex:目前 Cycle 檔於 0~2 有 Cycle 資料，最多只能 DBINSERT(3)
	引數 - **Cycle name**:
	• 指定要記錄的資料格式
	• 限制: 須為 Schema 中有定義的
	回傳值:
	• 1：插入 Cycle 資料成功
	• 0：插入 Cycle 資料失敗，非預期錯誤
	  • 請聯絡新代原廠
	• -1：插入 Cycle 資料失敗，Index 超過範圍
	  • 請確認 Index 是否非負整數且未超過範圍
	• -2：插入 Cycle 資料失敗，Cycle 檔未開啟
	  • 請確認有先成功執行 DBOPEN 或 DBNEW
	• -3：插入 Cycle 資料失敗，使用未定義的 Cycle name
	  • 請確認 Cycle name 是否正確
	  • 請確認 Schema 檔案內有定義該 Cycle name
	備註:
	1. 引數型態錯誤會觸發警報 COR-023 語義錯誤
	2. 此 Macro 語法不支援圖形模擬，會固定回傳 0
	3. 此 Macro 語法會操作目前開啟 Cycle 檔。Cycle 檔的開啟可透過 DBOPEN 以及 DBNEW
	4. 同時間僅能指定一個 Cycle name，而 DBLOAD 以及 DBINSERT 都會指定 Cycle name，若重復呼叫為後令蓋前令
	Ex:先使用 DBLOAD(0)，Cycle name 會指定為 Cycle 檔中位置"0"的 Cycle 資料的 Cycle name，此時再呼叫 DBINSERT，Cycle name 會改為指定 DBINSERT 引數所帶入的 Cycle name
	5. 版本: 支援 10.118.56L, 10.118.60F, 10.118.66 以及以後的版本
	Ex:
	`// DBOPEN 搭配 DBINSERT 把資料插入在最後一筆`
	`#30 := DBOPEN("File name");  // 回傳值為目前檔案內的 Cycle 資料個數`
	`DBINSERT(#30,"Cycle name");  // 把資料插入在最後一筆`

<table>
<tr><th>功能</th><th>說明</th></tr>
<tr><td>INT DBDELETE(Index)</td><td>功能:
刪除第 "Index" 筆的 Cycle 資料
影響:
執行成功後，所有於指定位置後方的 Cycle 資料會往前移一筆
引數 - Index:
指定要刪除第幾筆 Cycle 資料
限制: 最大值為"目前開啟的 cycle 檔的最後位置"，ex:目前 Cycle 檔於 0~2 有 Cycle 資料，最多只能 DBDELETE(2)
回傳值:
1：刪除 Cycle 資料成功
0：刪除 Cycle 資料失敗，非預期錯誤
請聯絡新代原廠
-1：刪除 Cycle 資料失敗，Index 超過範圍
請確認 Index 是否非負整數且未超過範圍
-2：刪除 Cycle 資料失敗，Cycle 檔未開啟
請確認有先成功執行 DBOPEN 或 DBNEW
備註:
引數型態錯誤會觸發警報 COR-023 語義錯誤
此 Macro 語法不支援圖形模擬，會固定回傳 0
此 Macro 語法會操作目前開啟 Cycle 檔。Cycle 檔的開啟可透過 DBOPEN 以及 DBNEW
版本: 支援 10.118.56L, 10.118.60F, 10.118.66 以及以後的版本
Ex:<pre>1 // DBOPEN 搭配 DBDELETE 刪除最後一筆資料
2 #30 := DBOPEN("File name"); // 回傳值為目前檔案內的 Cycle 資料個數
3 DBDELETE(#30 - 1); // 刪除最後一筆資料</pre></td></tr>
<tr><td>DRAWHOLE</td><td>依據刀具半徑及SETDRAW函數所定義之顏色，在目前位置畫一個圓（只在圖形模擬內有效，而不是在實際軌跡加走一個圓）</td></tr>
<tr><td>EXP</td><td>計算以自然數當作基底的指數值
EX:
#1:=EXP(1); // e^1 = 2.71828
有效版本：10.116.16</td></tr>
</table>

功能	說明

FLOOR	回傳小於或等於某數值的最大整數
	EX:
	#10 := 1.4;
	#1 := FLOOR(#10); // #1 = 1
	#2 := FLOOR(1.5); // #2 = 1

GETARG	讀取呼叫者傳遞的引數
	EX:
	假設O0001主程式內容為
	G101 X30. Y40. Z1=40. Z2=50.;
	G0101擴充巨集程式使用GETARG讀取引數內容
	#1 := GETARG(X); // 將X引數內容30.存到#1
	#2 := GETARG(Z1); // 將Z1引數內容40.存到#2
	#3 := GETARG(W);
	// 因W不存在，所以#3內容為(VACANT, #0)

GETTRAPARG	讀取G66/G66.1在Trap單節內的引數內容
	EX:
	假設O0001主程式內容為
	G66 P100 X100. Y100.
	G01 X20.
	O0100副程式使用GETTRAPARG讀取引數內容
	#1 := GETARG(X);
	// 將X引數內容100.存到#1
	#2 := GETTRAPARG(X);
	// 將Trap單節X引數內容20.存到#2

LN	計算以自然數當作基底的對數值
	EX:
	#2:=LN(100); // ln100 = 4.60517
	注意事項：後方引數需為正數，否則會跳警報
	有效版本：10.116.16

MAX	決定兩輸入值的最大值
	EX:
	#10 := 1.2;
	#20 := 4.5;
	#1 := MAX(#10, #20); // #1 = 4.5
	#2 := MAX(-1.2, -4.5); // #2 = -1.2

功能	說明

MIN	決定兩輸入值的最小值
	EX:
	#10 := 1.2;
	#20 := 4.5;
	#1 := MIN(#10, #20); // #1 = 1.2
	#2 := MIN(-1.2, -4.5); // #2 = -4.5

MSG	自訂提示，詳情請參閱『MACRO自訂提示』
	EX:
	MSG(100); // 提示ID
	MSG("鑽頭移失"); // 提示顯示內容
	MSG(100, "鑽頭遺失"); // 提示ID + 顯示內容
	備註:
	1. 提示內容有字串長度限制，中文 : 19個字，英文 : 39個字。
	2. 提示ID只允許使用0~65535之間的整數，否則跳警報COR-023【語意錯誤】。

功能	說明

OPEN("檔名")
or
OPEN("檔名", "寫檔方式")
	在 NcFiles 或 Macro 資料夾(資料夾位置請參考 Pr3219)開啟一文字檔，其檔名為所指定之名稱。需在檔案開啟後，PRINT函數才有效。
	開檔規則:
	檔案已存在: 開啟原本的檔案，路徑可以在NcFiles或Macro資料夾。(注意事項: 如果Macro資料夾已存在該檔案，就不會在NcFile資料夾再生成檔案)
	檔案不存在: 生成一份新的在NcFiles資料夾。
	若檔案名稱為"COM"時，表示打開RS232/RS485傳輸埠，其設定由Pr3905決定。
	EX:
	OPEN("COM"); // 打開傳輸埠
	PRINT("\\p"); // 輸出'%'字元
	FOR #1 = 1 TO 5000 DO
	#30 := #1 * 10.;
	PRINT("G01 X#30"); // 輸出G01 X10.0
	END_FOR;
	PRINT("\\p"); // 輸出'%'字元
	CLOSE(); // 關閉傳輸埠
	"寫檔方式"決定OPEN時，保留或清空原先的檔案內容。（有效版本：10.116.36I）
	(i) 指定為"a"：保留原先的檔案內容，並將新的資料接續輸出在該文字檔。
	EX:
	OPEN("PROBE.NC", "a");
	// 開啟並保留PROBE.NC檔案內容，準備作資料輸出
	(ii) 不指定或指定為"w"：清空原先的內容，並將新的資料重新輸出在該文字檔。
	EX:
	OPEN("PROBE.NC");
	// 開啟並清空PROBE.NC檔案內容，準備作資料輸出
	OPEN("PROBE.NC", "w");
	// 開啟並清空PROBE.NC檔案內容，準備作資料輸出
	(iii) 與上述兩點不同之寫檔方式：寫檔方式指定錯誤，系統會發出COR-301警報
	EX:
	OPEN("PROBE.NC", "abc");
	// 寫檔方式指定錯誤，發出COR-301警報，無法開啟PROBE.NC作資料輸出
	(iv) 使用#或@變數轉成字串，小數點輸出位數會與Pr17連動（有效版本：10.118.12C）
	(v) 使用#或@變數轉成字串，若在後方加上[*]，小數點輸出位數會由此數值決定

PARAM	讀取系統參數的內容 EX: #1 := PARAM(3204); // 讀取Pr3204(PLC掃瞄時間)之內容

功能	說明

POP	將堆疊（STACK）里面的資料取出，依序由最上層一路取到最底層，使用者須注意堆疊里面共有幾筆資料，有5筆資料就只能使用5次POP。
	EX:
	PUSH(5); // 將數字5塞入堆疊中
	#1 := POP(); // 取出堆疊中最上層數值(#1 = 5)

POW	計算以某數值當作基底的指定乘冪次方值 EX: #3:=POW(16,0.5); // 16^0.5 = 4 注意事項：基底不可為負值，否則會跳警報COR-122 有效版本：10.116.16

PRINT	此函數用來輸出字串，輸出字串中的變數名稱會被取代成該變數之內容。
	字元"\"為逃脫字元，相關特殊字元定義如下:
	\\：表示\字元
	\@：表示@字元
	\#：表示#字元
	\p：表示%字元
	\"：表示"字元
	使用#或@變數轉成字串，小數點輸出位數會與Pr17連動（有效版本：10.118.12C）
	使用#或@變數轉成字串，若在後方加上[*]，小數點輸出位數會由此數值決定
	EX:
	假設Pr17=2，且為公制
	@53 = 20 ;
	#3 = 23.1234 ;
	PRINT("G01 X#3 Y@53 Z20.0") ;
	輸出結果為G01 X23.123 Y20 Z20.0 ; // 顯示到小數點後3位
	EX:
	@53 = 20 ;
	#3 = 23.1234 ;
	PRINT("G01 X#3[2] Y@53 Z20.0") ; // #3[2]代表只要小數點後2位
	輸出結果為G01 X23.12 Y20 Z20.0 ; // 顯示到小數點後2位

PUSH	將資料塞進堆疊（STACK），最先PUSH進入控制器的數值會堆疊在最底層，並依序往上疊加。 EX: PUSH(#1); // 將變數#1塞入堆疊中

功能	說明
RANDOM	產生0~32767間的一個隨機數 EX: #1 := RANDOM();

READDI (I點編號)
READDO(O點編號)
	以READDI、READDO指令後方小括弧內的數字，決定所讀取的I/O點編號。
	EX:
	@52 := READDI(31); // 把I31的值讀出來填到@52
	#88 := READDO(11); // 把O11的值讀出來填到#88
	G90 G10 L1000 P4000 R READDI(15); // 把I15的值讀出來填到R4000
	注意事項:
	1. 有效版本：10.116.23
	2. I/O點的讀取是在預解階段，但需在加工程式實際執行到READDI、READDO指令才處理，因此為了避免預解造成I/O點讀取錯誤，系統內部會自行等待，在使用READDI、READDO指令時，會有減速到零的情形。
	3. 讀取的I/O點編號範圍僅限於0~511間，若範圍錯誤，系統將發出COR-138 I/O/A點讀寫指令格式錯誤警報。

READABIT(A點編號)	以READABIT指令後方小括弧內的數字，決定所讀取的A點編號。
	EX:
	@52 := READABIT(31); // 把A31的值讀出來填到@52
	#88 := READABIT(11); // 把A11的值讀出來填到#88
	注意事項:
	1. 有效版本：10.116.44
	2. A點的讀取是在預解階段，但需在加工程式實際執行到READABIT指令才處理，因此為了避免預解造成A點讀取錯誤，系統內部會自行等待，在使用READABIT指令時，會有減速到零的情形。
	3. 讀取的A點編號範圍僅限於0~511間，若範圍錯誤，系統將發出COR-138 I/O/A點讀寫指令格式錯誤警報。

功能	說明

READRREGBIT(R值編號,指定Bit)	以READRREGBIT指令後方小括弧內的2個數字，決定所讀取的R值編號及指定的Bit。
	EX:
	@52 := READRREGBIT(31,3); // 把R31的第三個Bit的值讀出來填到@52
	注意事項:
	1. 有效版本：10.116.39
	2. R值的讀取是在預解階段，但需在加工程式實際執行到READRREGBIT指令才處理，因此為了避免預解造成R值讀取錯誤，系統內部會自行等待，在使用READRREGBIT指令時，會有減速到零的情形。
	3. R值編號若小於0或大於65535，系統將發出COR-135 R值讀寫指令格式錯誤。
	4. R值編號若為不正確字元，系統將發出COR-5 程式載入失敗、COM-8 子句中沒有結束的符號';'。
	5. 指定Bit若小於0或大於31，系統將發出COR-135 R值讀寫指令格式錯誤。
	6. 指定Bit若為不正確字元，或R值編號及指定Bit皆為不正確字元，系統將發出COR-5 程式載入失敗、COM-9 錯誤的給值符號':='。

ROUND	回傳某數值完成四舍五入進位後的值
	EX:
	#10 := 1.4;
	#1 := ROUND(#10); // #1 = 1
	#2 := ROUND(1.5); // #2 = 2

SCANTEXT	此函數用來讀取公用變數所儲存的字串內容 將字串存入公用變數時，控制器會先進行ASCII轉碼，因此若直接輸出將會得到錯誤的結果，故需使用此函數以取得正確字串內容 EX: %@MACRO @1:="12"; // 16進位HEX=3231，10進位DEC=12849 #1:=SCANTEXT(1); OPEN("NC"); PRINT("@1"); PRINT("#1"); CLOSE(); M30; 所得結果為 @1 = 12849 #1 = 12

功能	說明

SETDO(O點編號, O點開或關)	以SETDO指令後方小括弧內的2個數字，決定所設定的O點編號及該O點開或關（1：開，0：關）
	EX:
	SETDO(3, 1); // 設定O3 on
	SETDO(8, 0); // 設定O8 off
	注意事項:
	1. 有效版本：10.116.23
	2. O點的寫入是在插值階段，因此在執行時可不必減速到0，但在預解階段的MACRO處理，開發人員應視需求決定是否以WAIT做保護，WAIT則會造成減速至0。
	3. PLC與SETDO應避免混用，例如MACRO中以SETDO讓O1 on，卻在PLC中讓O1 off。目前規格是依順序後令蓋前令，但在應用上不好掌握，故建議不要混用。
	4. 設定的O點編號範圍僅限於0~511間，若範圍錯誤，系統將發出COR-138 I/O/A點讀寫指令格式錯誤警報。

SETABIT(A點編號, A點開或關)	以SETABIT指令後方小括弧內的2個數字，決定所設定的A點編號及該A點開或關（1：開，0：關）
	EX:
	SETABIT(3, 1); // 設定A3 on
	SETABIT(8, 0); // 設定A8 off
	注意事項:
	1. 有效版本：10.116.44
	2. A點的寫入是在插值階段，因此在執行時可不必減速到0，但在預解階段的MACRO處理，開發人員應視需求決定是否以WAIT做保護，WAIT則會造成減速至0。
	3. PLC與SETABIT應避免混用，例如MACRO中以SETABIT讓A1 on，卻在PLC中讓A1 off。目前規格是依順序後令蓋前令，但在應用上不好掌握，故建議不要混用。
	4. 設定的A點編號範圍僅限於0~511間，若範圍錯誤，系統將發出COR-138 I/O/A點讀寫指令格式錯誤警報。

功能	說明

SETRREGBIT(R值編號, 指定Bit, 開或關)	以SETRREGBIT指令後方小括弧內的3個數字，決定所設定的R值編號、指定的Bit及該Bit的開或關（1：開，0：關）
	EX:
	SETRREGBIT(50,3,1); // 設定R50的第3個Bit on
	SETRREGBIT(50,4,0); // 設定R50的第4個Bit off
	注意事項:
	1. 有效版本：10.116.39
	2. R值的寫入是在插值階段，因此在執行時可不必減速到0，但在預解階段的MACRO處理，開發人員應視需求決定是否以WAIT做保護，WAIT則會造成減速至0。
	3. PLC與SETRREGBIT應避免混用，例如MACRO中以SETRREGBIT讓R50的第一個Bit on，卻在PLC中讓R50的第一個Bit off。目前規格是依順序後令蓋前令，但在應用上不好掌握，故建議不要混用。
	4. R值編號若小於0或大於65535，系統將發出COR-135 R值讀寫指令格式錯誤。
	5. 指定Bit若小於0或大於31，系統將發出COR-135 R值讀寫指令格式錯誤。
	6. 第三個引數若不是0（off）或1（on），系統將發出COR-135 R值讀寫指令格式錯誤。
	7. 任意引數輸入不正確字元，系統將發出COR-5 程式載入失敗、COM-3 句法錯誤。

功能	說明

SETDRAW(路徑顏色) or SETDRAW(路徑顏色,填充顏色,刀具半徑)	定義圖形模擬的畫圖樣式:
	1. 路徑顏色：設定輪廓線的顏色，可使用BGR碼，或參考模擬參數設定里面的顏色代碼來進行設定。
	2. 填充顏色：設定DRAWHOLE函數圓內填充的顏色，可使用BGR碼，或參考模擬參數設定里面的顏色代碼來進行設定。
	3. 刀具半徑：設定DRAWHOLE函數圓半徑大小，影響包含可使用刀徑補償功能的G碼，如G01
	常用BGR碼如下所示:
	 
	顏色代碼/BGR碼:
	• 0：0
	• 1：8388608
	• 2：32768
	• 3：8421376
	• 4：128
	• 5：8388736
	• 6：32896
	• 7：12632256
	• 8：8421504
	• 9：16711680
	• 10：65280
	• 11：16776960
	• 12：255
	• 13：16711935
	• 14：65535
	• 15：16777215
	注意事項:
	• BGR碼與RGB碼，只差在B(Blue)與R(Red)的擺放數值順序不同，使用者可以自行換算。
	e.g.
	RGB碼紅色為0xFF0000，轉到BGR碼則為0x0000FF = 255 (上述顏色代碼12)
	• SETDRAW會同時設定到路徑及畫圓的顏色，若想讓路徑及畫圓的顏色有所區別，要記得在執行完DRAWHOLE之後，再下一次SETDRAW，將路徑顏色改回來。
	e.g.
	%@MACRO
	#3:=SETDRAW(#1,#2,#18);
	// 使用#3記錄原本路徑顏色，#2定義填充顏色，#18定義畫圓半徑
	DRAWHOLE();
	SETDRAW(#3); // 畫圓後將路徑顏色改回
	M99;

功能	說明

SIGN	回傳某數值的正負號，負數為-1，正數為1，0為0
	EX:
	#10 := 4;
	#1 := SIGN(#10); // #1 = 1
	#2 := SIGN(-4); // #2 = -1
	#3 := SIGN(0); // #3 = 0

SIN	計算某數值的正弦值
	EX:
	#10 := 90;
	#1 := SIN(#10); // #1 = 1
	#2 := SIN(-90); // #2 = -1

SLEEP	暫時放棄此次巨集循環的執行權，一般配合回圈使用（FOR、WHILE...等），以避免進入無窮回圈而造成人機卡死 EX: SLEEP();

SQRT	計算某數值的平方根值
	EX:
	#10 := 4;
	#1 := SQRT(#10); // #1 = 2
	#2 := SQRT(9); // #2 = 3

功能	說明

STD(引數1,引數2)	根據Pr17將數值轉換成當時系統設定的輸入單位（Input Unit, IU）
	1. 引數1為欲改變單位之數值
	2. 引數2為標準化單位，一般使用#1600，而#1600設定值來自Pr17
	公制:
	EX1:
	當Pr17=2，#1600對應LIU = 0.001mm
	#9 := 100;
	#10 :=STD(#9,#1600); // #9為100個BLU，故#10為0.1mm(100*0.001)
	Ex2:
	當Pr17=3，#1600對應LIU = 0.0001mm
	#9 := 100;
	#10 :=STD(#9,#1600); // #9為100個BLU，故#10為0.01mm(100*0.0001)
	英制:
	Ex3:
	當Pr17=2，#1600對應LIU = 0.0001inch
	#9 := 100;
	#10 :=STD(#9,#1600); // #9為100個BLU，故#10為0.01inch(100*0.0001)

STDAX(引數1,引數2)	將數值轉成對應軸向的標準單位 引數1為變數，引數2為對應軸向的名稱 EX: #24 := STDAX(#24,X); #3 := STDAX(#3,A);

STKTOP	將堆疊（STACK）里面的資料複製出來
	EX:
	PUSH(5); // 將數字5塞入堆疊中
	PUSH(6); // 將數字6塞入堆疊中
	PUSH(7); // 將數字7塞入堆疊中
	#1 := STKTOP[0]; // #1 = 7
	#2 := STKTOP[1]; // #2 = 6
	#3 := STKTOP[2]; // #3 = 5

功能	說明
SYSVAR(軸群識別碼,系統變數碼)	讀取特定軸群中的的系統變數 軸群識別碼：1為第一軸群組，2為第二軸群組，以此類推。 系統變數碼：欲讀取的系統變數編號 EX: #1 := SYSVAR(1, 1000); // 讀取第一軸群的系統變數#1000（插值模式）

TAN	計算某數值的正切值
	EX:
	#10 := 45;
	#1 := TAN(#10); // #1 = 1
	#2 := TAN(-45); // #2 = -1

WAIT	系統停止預解，直到WAIT之前的指令執行完畢
	EX:
	%@MACRO
	@50:=1; // @50內容填成1
	G90 G01 X100. F1000; // 假設此時Reset系統
	WAIT();
	@50:=0; // @50內容填成0
	M30;
	假設在G01尚未執行完畢時重置系統，由於下一個單節為WAIT，因此系統停止預解，故重置後@50內容仍然是1。

CHKMN("機械廠代碼")	檢查機械廠代碼，1：一致，0：不符
	EX:
	%@MACRO
	#51 := CHKMN("5566"); //#51之值為檢查結果
	IF #51=0 THEN //若機械廠代碼不符則發出警報
	ALARM(501, "The manufacturer code is invalid.");
	END_IF;
	目標版本：10.116.6A

功能	說明

CHKSN("序號")	檢查控制器序號，1：一致，0：不符
	EX:
	%@MACRO
	#52 := CHKSN("M9A0001"); //#52之值為檢查結果
	IF #52=0 THEN //若序號不符則發出警報
	ALARM(502, "The serial number is invalid.");
	END_IF;
	目標版本：10.116.6A

CHKMT("機床屬性")	檢查機床屬性，1：一致，0：不符
	EX:
	%@MACRO
	#53 := CHKMT("MILL"); //#53之值為檢查結果
	IF #53=0 THEN //若機床屬性不符則發出警報
	ALARM(503, "The machine type is invalid.");
	END_IF;
	目標版本：10.116.6A

CHKMI("機型")	檢查控制器機型，1：一致，0：不符
	除了SUPER系列需輸入S，其他機型皆按照實際型號輸入，如10B則輸入10B，11A則輸入11A。
	EX:
	%@MACRO
	#54 := CHKMI("S"); //#54之值為檢查結果
	IF #54=0 THEN //若機型不符則發出警報
	ALARM(504, "The hardware type is invalid.");
	END_IF;
	目標版本：10.116.6A

功能	說明

CHKINF(類別編號, "代碼")	檢查代碼與類別編號對應的內容是否一致，一致：1 不一致：0
	類別編號1~5分別為：
	1. 機械廠代碼
	2. 序號
	3. 機床屬性
	4. 機型
	5. 專機代碼
	機型代碼除了Super系列簡寫成s之外，其餘系列皆維持相同ex. 10B->10B、EHMC→EHMC
	EX：
	%@MACRO
	#51 := CHKINF(1, "5566"); //#51之值為檢查結果
	IF #51=0 THEN //若機械廠代碼不符則發出警報
	ALARM(501, "The manufacturer code is invalid.");
	END_IF;
	#52 := CHKINF(2, "M9A0001"); //#52之值為檢查結果
	IF #52=0 THEN //若序號不符則發出警報
	ALARM(502, "The serial number is invalid.");
	END_IF;
	#53 := CHKINF(3, "MILL"); //#53之值為檢查結果
	IF #53=0 THEN //若機床屬性不符則發出警報
	ALARM(503, "The machine type is invalid.");
	END_IF;
	#54 := CHKINF(4, "S"); //#54之值為檢查結果
	IF #54=0 THEN //若機型不符則發出警報
	ALARM(504, "The hardware type is invalid.");
	END_IF;
	#55 := CHKINF(5, "10"); //#55之值為檢查結果
	IF #55=0 THEN //若專機代碼不符則發出警報
	ALARM(505, "The industrial machine ID is invalid.");
	END_IF;
	參數型態需正確，且類別編號不可超出範圍，否則會跳警報COR-353 **【CHKINF 參數輸入錯誤】**
	EX：
	%@MACRO
	#51 := CHKINF(60, "Mill"); //檢查編號超出範圍
	#51 := CHKINF("2", "Mill"); //參數一型態不正確
	#53 := CHKINF(5, 12345); //參數二型態不正確
	目標版本：10.118.22M、10.118.28B、10.118.30

功能	說明

STR2INT("字串")	轉換數字字串為整數型態
	EX1:
	%@MACRO
	@1:="5555";
	#1:= SCANTEXT(1); // #1 = 字串5555
	#2:= STR2INT("#1"); // #2 = 5555
	EX2:
	%@MACRO
	#1:=STR2INT("100"); // #1 = 100
	EX3:
	%@MACRO
	#1:=STR2INT("123.456"); // #1 = 123
	注意事項：只要有文字的部分，即為不合法之輸入

SYSDATA(系統診斷變數號碼)	讀取系統診斷變數
	範例:
	// 假設要取得系統診斷變數 D336、D77
	WAIT(); // 擋預解，否則無法取得當前最新的數值
	#1 := SYSDATA(336); // 軸卡交換時間(D336)
	#2 := SYSDATA(77); // 硬體剩餘記憶體(D77)
	OPEN("DbgData.txt", "a"); // 開檔，檔名為"DbgData.txt"
	PRINT("#1 #2"); // 將變數值輸出到檔案內
	CLOSE(); // 關檔
	注意事項:
	1. 有效版本：10.118.23U, 10.118.28H、10.118.33
	2. 執行函數前建議下 WAIT() 函數擋預解，以取得執行當前的最新數值
	3. 引數格式須為整數，否則會導致函數無法正常執行
	4. 變數號碼須在診斷變數號碼範圍內，否則發 COR-016 **【不合法的變數存取】**
	錯誤範例:
	SYSDATA("77"); // 引數須為整數，發 COR-023 警報
	SYSDATA(77.0); // 引數須為整數，發 COR-023 警報
	SYSDATA(D77); // 引數須為整數，此輸入導致語法錯誤，發 COM-008 警報
	SYSDATA(10000); // 變數號碼超出診斷變數數值範圍，發 COR-016 警報

功能	說明

DRVDATA(站號, 10進位格式變數號碼)
DRVDATA(站號, "16進位格式變數號碼")
	讀取驅動器狀態變數
	變數號碼有10進位與16進位兩種輸入方式
	10進位：將變數號碼轉成10進位
	16進位：格式為 "xxxh", x=0~F，在字串內輸入16進位的編號(3位數)，以h做結尾，字母大小寫不拘
	範例:
	// 假設要取得第一軸(站號1000)速度命令(Pn-D26)與第一主軸(站號1003)編碼器內部溫度(Pn-D61)
	WAIT(); // 擋預解，否則無法取得當前最新的數值
	#1 := DRVDATA(1000, 3366); // 第一軸速度命令(Pn-D26，以10進位整數表示D26為3366)
	#2 := DRVDATA(1003, "D61h"); // 第一主軸編碼器內部溫度(Pn-D61，以16進位字串表示D61為 "D61h")
	OPEN("DbgData.txt", "a"); // 開檔，檔名為"DbgData.txt"
	PRINT("Pn-D26: #1, Pn-D61: #2"); // 將變數值輸出到檔案內
	CLOSE(); // 關檔
	DbgData.txt 可能的輸出為
	Pn-D26: 150, Pn-D61: 430
	注意事項:
	1. 有效版本：10.118.23U, 10.118.28I、10.118.34
	2. 執行函數前建議下 WAIT() 函數擋預解，以取得執行當前的最新數值
	3. 由於通訊限制，驅動器診斷變數並非及時從驅動器讀取上來，每個函數的執行時間大約為 0.1~0.2s
	4. 第一引數格式須為整數，否則發 COR-023 **【語義錯誤】**
	5. 第二引數如為字串需為16進位格式("xxxh", x=0~F)，否則發 COR-023 **【語義錯誤】**
	6. 第二引數需以10進位整數/16進位字串表示，否則發 COM-003 **【句法錯誤】**
	7. 變數號碼須在驅動器狀態變數號碼範圍內，且控制器支援讀取此狀態變數，否則發 COR-016 **【不合法的變數存取】**，可藉由查看控制器的"驅動器軸向資訊"頁面查詢控制器是否有支援讀取此狀態變數
	8. 如使用的驅動器不支援該狀態變數號碼，但控制器支援讀取，函數將回傳 0
	9. 站號無對應的驅動器，或使用非新代 M3 驅動器，函數回傳 VACANT
	10. 由於狀態變數在開機時需要一段時間進行讀取，建議於開機一分鐘後再呼叫此函數，否則會因為變數未讀取完成而發COR-016 **【不合法的變數存取】**
	錯誤範例:
	// 發 COM-3 警報
	DRVDATA(1003, D61h); // 第二引數需以10進位整數/16進位字串表示，錯誤時發 COM-3 警報

功能	說明

	// 發 COR-023 警報
	DRVDATA("1003", 3425); // 站號數值必須為整數，發 COR-023 警報
	DRVDATA(1003, "G21h"); // 第二引數如為字串需為16進位格式("xxxh", x=0~F)，錯誤時發 COR-023 警報
	DRVDATA(1003, "3425"); // 第二引數如為字串需為16進位格式("xxxh", x=0~F)，錯誤時發 COR-023 警報
	DRVDATA(1003, "0D61H"); // 第二引數如為字串需為16進位格式("xxxh", x=0~F)，錯誤時發 COR-023 警報
	
	// 發 COR-016 警報
	DRVDATA(1003, "DFFh"); // 驅動器不支援此變數號碼，發 COR-016 警報
	
	// 回傳 VACANT
	DRVDATA(9999, "D61h"); // 站號無對應的驅動器，回傳 VACANT
	DRVDATA(9999, "DFFh"); // 站號無對應的驅動器，不檢查變數號碼，回傳 VACANT

10 副程式呼叫

10.1 呼叫方式

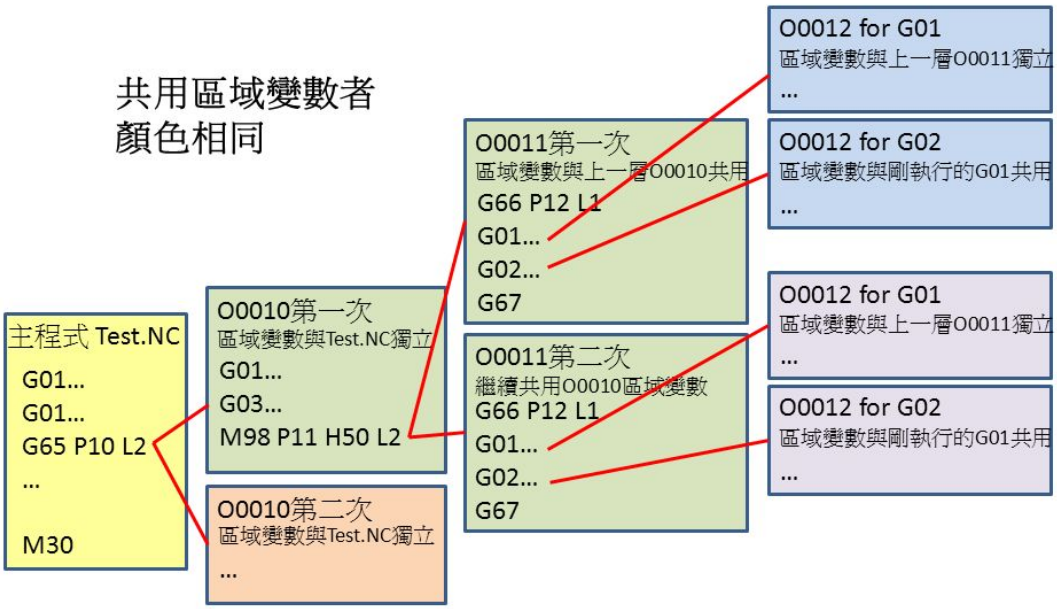
語法	說明	呼叫類型	區域變數	範例
M98 P_ H_ L_	呼叫副程式 P_ 副程式名稱 H_ 起始序號 L_ 重覆次數	子程式	繼承主程式之區域變數#1~#400	M98 P10 L2; 說明: 呼叫O0010兩次
M198 P_ H_ L_ （當M198未被Pr3601~登錄時）	呼叫副程式 P_ 副程式名稱 H_ 起始序號 L_ 重覆次數	子程式	繼承主程式之區域變數#1~#400	M198 P10 L2; 說明: 呼叫O0010兩次
G65 P_ L_	呼叫單一巨集程式 P_ 副程式名稱 L_ 重覆次數	巨集程式	使用獨立於主程式之區域變數#1~#400，並於#1~#26記載呼叫時所帶之引數	G65 P10 L2 X10.0 Y10.0; 說明: 呼叫O0010兩次，並輸入引數。
G66 P_ L_	使用移動指令來呼叫模式巨集程式 P_ 副程式名稱 L_ 重覆次數	模式巨集程式	每次呼叫G66時會創建一塊獨立的#1~#400，這塊區域變數會被共用直到執行到G67，執行完G67後，此塊區域變數會被回收清除。 但請注意，這塊區域變數只在G66 P引數指定到的副程式內共用，並不與呼叫G66、G66.1的該層程式的區域變數共用。	G66 P10 X10.0 Y10.0; **X20.** **Y20.** 說明: X20.與Y20.移動指令會呼叫O0010，並輸入引數X10.0、Y10.0。
G66.1 P_ L_	每個單節皆會呼叫模式巨集程式 P_ 副程式名稱 L_ 重覆次數	模式巨集程式	與G66相同	G66.1 P10 X10.0 **X20.** **G04 X2.;** **M30;** 說明: 每一單節指令都會呼叫O0010，並輸入引數X10.0。

語法	說明	呼叫類型	區域變數	範例
G_ L_	呼叫擴充G碼 L_ 重覆次數	巨集程式	每次呼叫都會創建一塊新的區域變數 #1~#400，於該Macro結束時恢復上一層的區域變數。	G128 L3 X1.0; 說明: 呼叫G0128三次
G_	呼叫客製G碼（G00、G01、G02、G03、G53、G40、G41、G42） 使用前必須在Pr3701~登錄。	巨集程式	每次呼叫都會創建一塊新的區域變數 #1~#400，於該Macro結束時恢復上一層的區域變數。	G01 A_ B_ C_; 說明: 呼叫客製G01
T_	呼叫副程式T0000來換刀，在副程式內的T碼為普通T碼，不會再呼叫T0000。	若Pr3215 = 1，則為子程式	繼承主程式之區域變數#1~#400	T3; 說明: 呼叫T0000
		若Pr3215 = 2，則為巨集程式	每次呼叫都會創建一塊新的區域變數 #1~#400，於該Macro結束時恢復上一層的區域變數。	
M_	呼叫M碼巨集程式， 在巨集內的M碼為普通M碼，不會再呼叫M碼巨集 使用前必須在Pr3601~登錄。 注一現況支援範圍為0 ~ 9999	巨集程式	每次呼叫都會創建一塊新的區域變數 #1~#400，於該Macro結束時恢復上一層的區域變數。	M13 A_ B_ C_; 說明: 呼叫M0013之巨集程式。

備註:

- 上表內之L引數未指定時，系統內定預設值為1。
- 上表內呼叫的副程式之區域變數（#1~#400），其生命周期請參考Macro變數規格。

變數生命周期之範例:



10.2 返回方式

語法	說明	範例
M99	返回主程式	M99;
M99 P_	返回主程式特定序號，P_：欲到達的**序號**編號	M99 P100; 回到主程式序號N100位置
M99 Q_	返回主程式特定行號，Q_：欲到達的**行號**編號	M99 Q100; 回到主程式行號100行位置
G67	取消G66	G67;

11 變數規格

- 有關#及@的變數規格說明，請參考Macro變數規格
- 將不同型態的變數(Long和Double)一起做四則運算，會強制轉型為Double再做計算，例如

```
@1 := 1;         // 以Long型態指派變數 @1
@2 := 2.0;       // 以Double型態指派變數 @2
@3 := @1 + @2;   // 得到結果為"@3 := 3.0"型態為Double

@4 := 2;         // 以Long型態指派變數 @4
@5 := 100;       // 以Long型態指派變數 @5
@6 := @4 * @5;   // 得到結果為"@6 := 200"型態仍為Long
```

12 MACRO自訂警報

12.1 MACRO警報觸發語法

```
%@MACRO
ALARM(xxx);// xxx為警報編號
M30;
```

12.2 DOS系統警報內容編輯說明

- 檔案路徑:
  -> 繁簡中文: C:\\CNC\\EXE\\APPDATA.RES\\CNCCHI.STR
  -> 英文版: C:\\CNC\\EXE\\APPDATA.RES\\CNCENG.STR
  -> 其他: C:\\CNC\\EXE\\APPDATA.RES\\CNCLOC.STR
- 內容格式: **24xxx="1;MSG=警報內容"**，其中xxx為警報編號，請選擇未使用過的號碼作為自訂警報編號，並請注意識別代號為**24**。
- 範例:
  -> CNCCHI.STR:
  24003="1;MSG=**最大圓弧弦長小於或等於0**"

  -> CNCENGSTR:
  24003="1;MSG= **max arc length can not be negative**"

12.3 WinCE系統警報內容編輯說明

- 檔案路徑:
  -> 中文版: DiskC\\OCRes\\CHT\\String\\AlarmMacro_CHT.Xml
  -> 英文版: DiskC\\OCRes\\Common\\String\\AlarmMacro_Com.Xml
  -> 通則: DiskC\\OCRes{color:#0000ff}**L**\\String\\AlarmMacro_**L**.Xml，**其中L為各語系之名稱**
- 檔案格式: <Message ID="AlarmMsg::**Macro**::ID=xxx" Content="**警報內容**" />，其中xxx為警報編號，請選擇未使用過的號碼作為自訂警報編號，並請注意識別字母為**Macro**。警報內容的字串長度可容納48個英文字，或是31個中文字，多餘的字串會超出警報視窗。
- 範例:
  -> CusMacroAlarmMsg_CHT.Xml:
  <Message ID="AlarmMsg::**Macro**::ID=3" Content="**最大圓弧弦長小於或等於0**" />

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

12.4 透過SI（SyntecIDE）進行警報字串編輯

MACRO警報字串編輯功能已整合到SI內，相關操作說明請參考Macro警報字串編輯器

13 MACRO自訂提示（MSG）

13.1 MSG規格說明

- MACRO警報發生時，必須重置系統才可消除警報；而MSG自訂提示僅需按下"ESC"即可消除，可用於單純提示狀態，但須注意當程式結束時，提示也會自動消失。
- 提示內容有字串長度限制，中文 : 19個字，英文 : 39個字。
- 提示ID只允許使用0~65535之間的整數，否則跳警報COR-023【語意錯誤】。

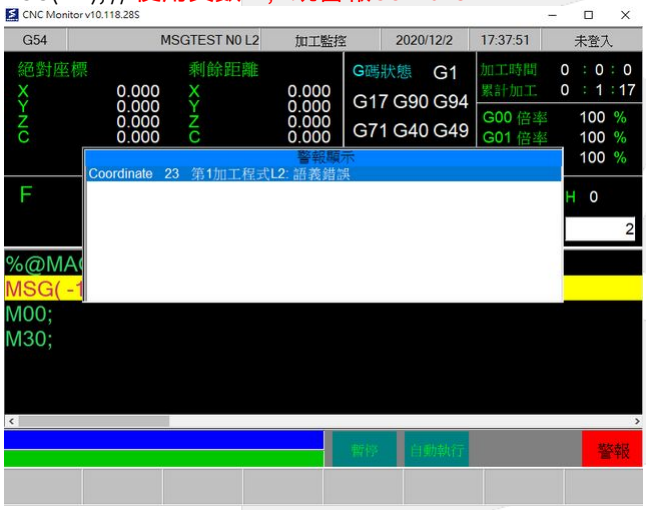
13.2 MSG觸發語法

- MSG(100); //警報ID

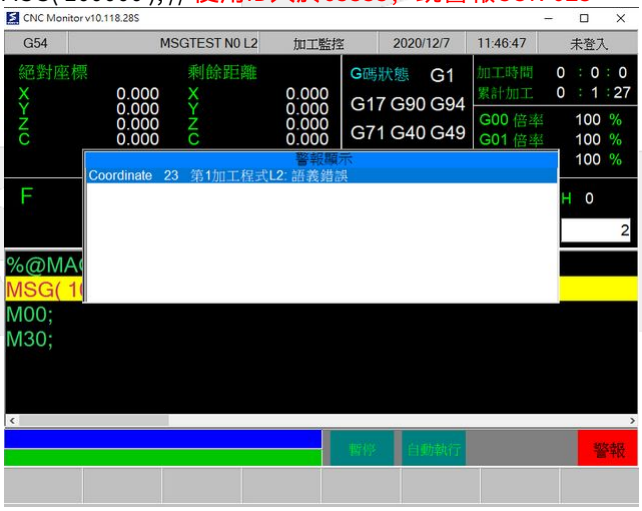
- MSG("鑽頭移失");//警報預設ID+顯示內容
  10.118.48U, 10.118.52O, 10.118.56I, 10.118.60C, 10.118.62及之後版本警報預設ID為65535 (之前版本預設ID為-1)

- MSG(100, "鑽頭遺失");//警報ID+顯示內容

- MSG(-1); // 使用負數ID，跳警報COR-023



- MSG(100000); // 使用ID大於65535，跳警報COR-023



14 附錄

14.1 巨集使用說明

14.1.1 引言

- 控制器內建的基本G、T、M碼可能無法滿足各產業應用之需求，對此，新代科技提供客制巨集的功能，讓開發人員可以根據其機臺之特性及需求，開發對應的巨集程序，提供機臺專用之特殊動作，提升機臺價值。
- 在開始介紹前，先針對主程序中，呼叫其他程序的方式進行簡單基本定義
    - 呼叫副程序：執行呼叫之程序，不可讀取占用引數。
    - 呼叫巨集：執行呼叫之程序，可以讀取占用引數。
    - 引數的定義請參閱 引數說明
- 後續章節將介紹巨集程序的相關規格，呼叫副程序的規格將不贅述。

14.1.2 巨集分類

- 所有巨集均需要符合以下條件
    - 撰寫語法須符合巨集語法
        - 程式內容以%@MACRO開頭
        - 每行以分號"；"結尾
    - 檔名結尾不得有副檔名
    - 程式內容結尾必須是M99，回到呼叫的主程式
- 巨集基本上可以分以下幾類
    - G碼巨集
    - 非模態調用巨集G碼（G65）
    - 模態調用巨集G碼（G66/G66.1）
    - T碼巨集
    - M碼巨集

類型	特性	檔名規範	檔名範圍
G碼巨集	開發人員自行撰寫的G碼巨集，稱之為擴充G碼巨集，以便與標準G碼對比。	-檔名開頭必定是字母G -不可有副檔名 -擴充G碼巨集的檔名分成4碼及6碼數字兩種 **4碼** • G碼指令沒有小數位，檔名 = G + 四碼數字 • 左手邊數過來遇到非零數字前的零都可以省略 • 範例：檔名若為G0200，加工檔指令寫G200或G0200均可 **6碼** • G碼指令帶有小數位，檔名 = G + 六碼數字 • 前三碼對應G碼指令整數位 • 後三碼對應G碼指令小數位 • 左手邊數過來遇到非零數字前的零都可以省略 • 範例：檔名若為G200001，加工檔指令寫G200.1或G200.001均可 • 範例：檔名若為G200010，加工檔指令寫G200.10或G200 .010均可 • 範例：檔名若為G200100，加工檔指令則為G200.100	-G200~G999 -超出範圍不保證可正常使用 -可能與系統標準G碼沖突
非模態調用巨集G碼 （G65）	-透過巨集呼叫指定的程式 -必須是該行最後一個G碼	• 檔名開頭必定是字母O • 不可有副檔名 • 檔名除了O以外，其他字元必須是數字 • 呼叫時可以省略開頭的字元O • 例如檔案名稱為O0123，則指令可寫成G65 P123;	-O0000~O9999 -超出範圍不保證可正常使用
模態調用巨集G碼 （G66/G66.1）	-G66：任何移動單節執行完畢，透過巨集呼叫指定的程式 -G66.1：任何單節執行完畢，透過巨集呼叫指定的程式 -必須是該行最後一個G碼	• 檔名開頭必定是字母O • 不可有副檔名 • 檔名除了O以外，其他字元必須是數字 • 呼叫時可以省略開頭的字元O • 例如檔案名稱為O0123，則指令可寫成G66 P123;	-O0000~O9999 -超出範圍不保證可正常使用

類型	特性	檔名規範	檔名範圍
T碼巨集	-透過巨集呼叫T0000這支檔案 -Pr3215需設定為2，才算是T碼巨集 **Pr3215簡述** 設定為0: • T碼輔助碼 • 不呼叫T0000 設定為1: • 透過副程式呼叫T0000 • 不讀取、占用任何引數 設定為2: • 透過巨集呼叫T0000 • 可讀取、占用引數	• 檔名只能是T0000，不接受其他檔名 • 不可有副檔名	-T0000
M碼巨集	-與M碼輔助碼不同，會透過巨集呼叫對應的M碼巨集檔案 -登錄在 Pr3601~3610 的M碼才算是M碼巨集 -以下M碼乃是系統標準M碼，無法被登錄為M碼巨集 • M00 • M01 • M02 • M30 • M96 • M97 • M98 • M99	• 檔名開頭必為字母M • 不可有副檔名 • M碼檔名必為M後面帶四碼數字 • 範例：Pr3601=123，檔名為M0123，加工檔指令寫M123或M0123均可	-M0000~M9999 -超出範圍不保證可正常使用

14.1.3 巨集運作流程說明

- 以下主要以G碼巨集作為范例說明，其他巨集若有規格不同之處，會特別說明。
- 當加工程序執行到G碼時，會執行G碼巨集的內容。
- G碼巨集內最後一行撰寫M99，當G碼巨集執行到M99之後，會回到主程序繼續往下加工。
- G碼巨集內的動作可能會改變系統狀態，例如在G碼巨集內下G90/G91，會使命令模式改變。
  - 為了避免影響到主程序的執行，或者其他巨集/副程序的運作，通常希望主程序的狀態，在執行G碼巨集的前後狀態一致。
  - 所以建議G碼巨集內容的一開始要備份系統狀態，G碼巨集的結尾要將還原系統狀態。
- 范例
  - 主程序中會下G200這個指令，進而執行G0200這個G碼巨集
  - G0200中會先備份系統狀態
  - 接著使X軸增量移動10.
  - 在離開前將狀態還原

范例_Main

```
// Main
G90;
G01 X10. F100.;          // X軸透過G01移動    X=10.
G201;                    // X軸透過G201移動   X=20.
X-20.;                   // X軸透過G01移動    X=-20.
M30;
```

范例_G0200

```
// G0200
%@MACRO
#101 := #1000;                  // 備份#1000狀態，G00/G01/G02/G03/G33/G34/
G35
#102 := #1004;                  // 備份#1004狀態，G90/G91
G91 G00 X10.;                   // X軸透過G00增量移動10.   X=20.;
G#101;                          // 還原#1000狀態，G00/G01/G02/G03/G33/G34/
G35
G#102;                          // 還原#1004狀態，G90/G91
M99;                            // 回到主程序
```

14.1.4 引數用途簡介

- 巨集可以執行開發人員設計的動作，從簡單的狀態轉換，到復雜的多工序動作。
- 但如果一個巨集只能執行一個單一的流程，那開發人員為了各種狀況就需要撰寫數不清的巨集。
- 比如某一個巨集可以切削出10公分的正方形，但如果想要切削20公分的正方形時，卻需要再撰寫一個新的巨集。
- 這樣的巨集太過呆板，此時就可以透過引數，對巨集的執行內容進行變數調整，例如正方形的邊長，讓該巨集具有更高的彈性。
- 參考下面的范例
- 范例
  - 主程序中為了切削出10公分的正方形，所以下了G200
  - 主程序中為了切削出20公分的正方形，所以下了G201
  - 主程序為了切削出任意大小的正方形，所以下了G202，大小邊長由G202的引數X來決定。

- **范例_Main**

```
// Example002_Main
G90 G00 X0. Y0.;
G200;
G201;
G202 X30.;
M30;
```

范例_G0200

```
// G0200
%@MACRO
#101:=#1000;
#102:=#1004;
G91 G01 X10.;
G91 G01 Y10.;
G91 G01 X-10.;
G91 G01 Y-10.;
G#101;
G#102;
M99;
```

范例_G0201

```
// G0201
%@MACRO
#101:=#1000;
#102:=#1004;
G91 G01 X20.;
G91 G01 Y20.;
G91 G01 X-20.;
G91 G01 Y-20.;
G#101;
G#102;
M99;
```

范例_G0202

```
// G0202
%@MACRO
#101:=#1000;
#102:=#1004;
#103:=#24;
G91 G01 X#103;
G91 G01 Y#103;
G91 G01 X-#103;
G91 G01 Y-#103;
G#101;
G#102;
M99;
```

14.1.5 引數說明

- 引數乃是採用英文26個字母，除了G/N/O以外，每一個字母都有一個對應的區域變數（#變數），如下表所示。

引數	#		引數	#		引數	#
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			

說明				
分類	軸向引數	條件引數	特殊引數	例外引數

- 引數會被巨集讀取占用。
- 「讀取」表示該巨集內有撰寫到該引數對應的#變數，故該巨集讀取該引數進行運算。
- 「占用」表示該引數被讀取後，無法再被其他巨集讀取，稱之為引數該巨集占用。
- 引數被讀取之後不代表就會被占用，要看讀取的巨集特性。請參閱後續章節【巨集解譯處理順序】【巨集特性】。
- 引數沒有被讀取不代表不會被占用，要看引數的特性。請參閱此章節後續說明。
- 同一行內有重復的引數，並不會發生警報，但只有最後一個引數的數值會被讀取。
- 可以理解為重復的引數均是對同一個#變數填值，後面的引數數值蓋掉前面的引數數值。
- 引數基本上可以分為以下幾類

<table>
<tr><th>類型</th><th>特點</th><th>其他說明</th></tr>
<tr><td>軸向引數</td><td>軸向引數包含：X / Y / Z / A / B / C / I / J / K / U / V / W
只要其中一個引數被某個巨集占用，其他軸向引數均會一起被該巨集占用</td><td>-B碼受到【Pr3806 啟動第二輔助碼B碼】影響
<table><tr><th>Pr3806</th><th>類型</th><th>說明</th></tr><tr><td>0</td><td>軸向引數</td><td></td></tr><tr><td>1</td><td>B碼輔助碼</td><td>會將B碼數值填到R5</td></tr></table></td></tr>
</table>

<table>
<tr><th>類型</th><th>特點</th><th>其他說明</th></tr>
<tr><td>條件引數</td><td>條件引數包含：F / S / T / D / E / H / P / Q / R / M / B
相較於軸向引數，條件引數彼此都是獨立的。
不會因為任一引數被占用，就一起被該巨集占用。</td><td>-T碼受到【Pr3215 選刀時呼叫模式】影響
<table>
<tr><th>Pr3215</th><th>類型</th><th>說明</th></tr>
<tr><td>0</td><td>T碼輔助碼，不呼叫T000</td><td>僅將T碼數值填到#1036及對應的R值
（各軸群對應的R值不同）</td></tr>
<tr><td>1</td><td>透過副程式呼叫T0000</td><td>不接受任何引數，不算是巨集。</td></tr>
<tr><td>2</td><td>透過巨集呼叫T0000</td><td>可接受其他引數</td></tr>
</table>
-M碼受到【Pr3601~3610 登錄M碼呼叫巨集】影響
<table>
<tr><th>Pr3601~3610</th><th>類型</th><th>說明</th></tr>
<tr><td>沒有登錄的M碼</td><td>M碼輔助碼</td><td>僅將M碼數值填到#1038及R值
（各軸群對應的R值不同）</td></tr>
<tr><td>有登錄的M碼</td><td>M碼巨集</td><td>可接受其他引數</td></tr>
</table>
-M碼受到【Pr3598 巨集調用M碼作為引數使用時的模式選擇】影響
<table>
<tr><th>Pr3598</th><th>說明</th></tr>
<tr><td>0</td><td>巨集調用 #13 時，讀取該單節的最後一個 M 碼並占用該單節的所有 M 碼</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>類型</th><th>特點</th><th>其他說明</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Pr3598</th><th>說明</th></tr>
<tr><td>1</td><td>• M 碼被登錄為巨集時，不會被當作其他巨集的引數
• 巨集調用 #13 時，只會讀取並占用單節內的第一個非巨集 M 碼
• 同一單節內有多個巨集調用 #13 時，會報警 COR-371</td></tr>
</table>
-B碼受到【Pr3806 啟動第二輔助碼B碼】影響
<table>
<tr><th>Pr3806</th><th>類型</th><th>說明</th></tr>
<tr><td>0</td><td>軸向引數</td><td></td></tr>
<tr><td>1</td><td>B碼輔助碼</td><td>會將B碼數值填到R5</td></tr>
</table>
</td></tr>
<tr><td>特殊引數</td><td>L引數乃是設定該巨集的連續執行次數
例如：G200 L10，表示G200會連續執行10次。
就算呼叫巨集時沒有提供L引數，系統也會自動補上L=1，讓該巨集執行一次。</td><td>T碼巨集不會讀取L碼，故不論L碼數值是多少，T碼巨集永遠只會執行一次。</td></tr>
<tr><td>例外引數</td><td>例外引數包含：G / N / O
這三個字母乃是控制器語法中的關鍵字，無法被當做引數使用。
所以也沒有對應的#變數。</td><td>
<table>
<tr><th>類型</th><th>說明</th></tr>
<tr><td>G</td><td>乃是用來作為G碼指令或者G碼巨集使用</td></tr>
<tr><td>N</td><td>乃是用來作為程式旗標使用</td></tr>
<tr><td>O</td><td>乃是一般加工程式的檔名開頭</td></tr>
</table>
</td></tr>
</table>

14.1.6 巨集解譯處理順序

Unable to render include or excerpt-include. Could not retrieve page.

14.1.7 巨集特性

- 巨集在讀取占用引數時，除了前述的引數基本運作說明外，還有一些特性在，以下簡短說明。

條件 特性 引數讀取與占用 一行內多個同碼指令 巨集內其他指令

類型	啟用條件
G碼與G碼巨集	無
T碼巨集	• 【Pr3215 選刀時呼叫模式】此參數被設定為2，斷電重啟後，系統才會將T碼認定為T碼巨集。 • T碼被當做T碼巨集時，需要在T碼引數（#20）沒有被占用時，才會執行T碼巨集。
M碼巨集	• 【Pr3601~3610 登錄M碼呼叫巨集】M碼被登錄在此區參數，斷電重啟後，系統才會將設定的M碼認定為M碼巨集。 • 當M碼（#13）引數被占用，M碼巨集不會執行。 • 當有任何一個軸向引數被占用，M碼巨集不執行。

條件 特性 引數讀取與占用 一行內多個同碼指令 巨集內其他指令

類型	特性
G碼與G碼巨集	繼承功能 • 若有更新插值模式（#1000），則G碼具有繼承功能。 • 因為繼承功能，只要下軸向指令，亦會執行G碼。 • 例如：系統開啟X、Z軸 G98 G83 Z-40.0 R-5.0 P0.0 Q10.0 F1.5; X-3.; // 再次執行G83 X-3.

<table>
<tr><th>類型</th><th>特性</th></tr>
<tr><td>T碼巨集</td><td>

T碼的狀態更新會根據T碼當下的類型，更新不同的變數，以下是對應的更新規則。
<table>
<tr><th>T碼狀態更新</th><th>#1036</th><th>#20</th><th>R值</th></tr>
<tr><td>巨集</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>副程式</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>輔助碼</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>引數</td><td>X</td><td>O</td><td>X</td></tr>
</table>

在T碼巨集中，若要擷取T碼的數值，請使用T碼變數（#1036）。
在其他巨集中要擷取T碼，請使用T碼引數（#20）。
只有T碼被當做輔助碼時，T碼對應的R值才會被更新。T碼被當做巨集或引數時，T碼對應的R值均不會更新。

</td></tr>
<tr><td>M碼巨集</td><td>

M碼的狀態更新會根據M碼當下的類型，更新不同的變數，以下是對應的更新規則。
<table>
<tr><th>M碼狀態更新</th><th>#1038</th><th>#13</th><th>R值</th></tr>
<tr><td>巨集</td><td>O</td><td>Pr3598 =
0：O
1：X</td><td>X</td></tr>
<tr><td>輔助碼</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>引數</td><td>X</td><td>O</td><td>X</td></tr>
</table>

#13 作為引數時會被更新，作為巨集時受 Pr3598 控制。
只有M碼被當做輔助碼時，M碼對應的R值才會被更新。M碼被當做巨集或引數時，M碼對應的R值均不會更新。

</td></tr>
</table>

條件　特性　引數讀取與占用　一行內多個同碼指令　巨集內其他指令

類型	引數讀取與占用
G碼與G碼巨集	• G碼巨集執行時，會占用G碼巨集有讀取的引數。 • 如果有讀取到軸向引數，所有軸向引數都會被同時占用。

類型	引數讀取與占用
• 模態G碼 • 非模態調用G碼	• 會占用除了T碼以外的所有引數
T碼巨集	• T碼巨集執行時，會讀取所有需要被讀取的引數，不管該引數是否已經被占用。 • T碼巨集執行後，會占用T碼巨集有讀取的引數。如果有讀取到軸向引數，所有軸向引數都會被同時占用。 • T碼巨集不會受到L碼引數影響，永遠都只會執行一次。 • T碼巨集內如果撰寫L碼引數（#12），並不會去讀取L碼引數的數值，而是會被系統填為1。

M碼巨集	• M碼巨集執行後，會占用所有被M碼巨集讀取的引數。接著會再占用除了T以外的所有引數。
	  • 假設M碼巨集沒有讀取T引數，則在M碼巨集執行完畢後，T引數不會被占用。
	  • 假設M碼巨集有讀取T引數，則在M碼巨集執行完畢後，T已經被占用。

條件　特性　引數讀取與占用　一行內多個同碼指令　巨集內其他指令

類型	一行內多個同碼指令
G碼與G碼巨集	• 會由左至右依序解譯 • 只有最後一個G碼可以讀取及占用引數，前面的G碼均不會讀取任何引數，也不會占用任何引數。 • 最後一個G碼又分以下狀況

類型	讀取/占用引數	解譯 & 執行
G碼巨集	會	立刻解譯，立刻執行
模態調用G碼：G66	會	立刻解譯，不會立刻執行，任何移動單節執行完畢，會執行此G碼一次
模態調用G碼：G66.1	會	立刻解譯，不會立刻執行，任何單節執行完畢，會執行此G碼一次。
-插值G碼 -功能G碼	不會	立刻解譯，直到同行其他指令均解譯完畢後，倘若尚有對應引數，才執行此G碼

類型	一行內多個同碼指令
T碼巨集	• 多個T碼巨集撰寫在同一行時，會依序對每一個T碼進行解譯處理，且每一個T碼巨集都可以讀取引數。
M碼巨集	• 多個M碼巨集撰寫在同一行時，只會解譯處理第一個M碼巨集。 • 【Pr3810 啟動同一單節M code同時執行功能】此參數乃是針對M碼輔助碼，而非M碼巨集，所以就算此參數開成1，同一行的M碼巨集仍然只會執行第一個。

條件　特性　引數讀取與占用　一行內多個同碼指令　巨集內其他指令

類型	巨集內指令
G碼巨集	• G碼巨集內的G碼可以作為 指令 或 巨集。 • G碼巨集內的M碼可以作為 輔助碼 或 巨集。 • G碼巨集內的T碼可以作為 輔助碼 、 巨集 或 副程式。
T碼巨集	• T碼巨集內的G碼可以作為 指令 或 巨集。 • T碼巨集內的M碼均會被視為 輔助碼，不會被視為 巨集。 • T碼巨集內的T碼均會被視為 輔助碼，不會被視為 巨集 或 副程式。
M碼巨集	• M碼巨集內的G碼可以作為 指令 或 巨集 。 • M碼巨集內的M碼均會被視為 輔助碼，不會被視為 巨集。 • M碼巨集內的T碼均會被視為 輔助碼，不會被視為 巨集 或 副程式。

14.1.8 巨集呼叫範例

Unable to render include or excerpt-include. Could not retrieve page.

01_引數與程式變數（區域變數）

- 除了例外引數以外（G、N、O），所有的引數都對應到一個#變數（屬區域變數）。

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

```
// G0200
%@MACRO

//軸向引數
@101 := #1;                    // A
@102 := #2;                    // B
@103 := #3;                    // C
@104 := #4;                    // I
@105 := #5;                    // J
@106 := #6;                    // K
@121 := #21;                   // U
@122 := #22;                   // V
@123 := #23;                   // W
@124 := #24;                   // X
@125 := #25;                   // Y
@126 := #26;                   // Z

//條件引數
@107 := #7;                    // D
@108 := #8;                    // E
@109 := #9;                    // F
@111 := #11;                   // H
@113 := #13;                   // M
@116 := #16;                   // P
@117 := #17;                   // Q
@118 := #18;                   // R
@119 := #19;                   // S
@120 := #20;                   // T

//特殊引數
@112 := #12;                   // L

M00;                           // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
M99;
```

02_引數重復撰寫

- 重復的引數只有最後一個引數會被讀取
- 可以理解為重復的引數均是對同一個程式變數#值填值，後面的引數數值會蓋掉前面的引數數值。

Main

```
// Main
```

```
G01 X0.;
G200 X10. X-10.;     // X引數寫了兩次，只有X-10.會被G200讀取
                     // 可以視為#24先被X10.填成10.後
                     // 又被X-10.填成-10.
                     // G200執行完畢後，執行G01
                     // 因為此行的引數均已被占用
                     // 所以G01讀取不到任何引數。
M30;
```

G0200

```
// G0200
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #24;         // 讀取到的#24是-10.
G01 X#101;           // 移動到X-10.
M00;
WAIT();
G#100;               // 還原插值模式
M99;
```

03_軸向引數與條件引數

- 軸向引數只要其中一個被占用，則其他軸向引數都會一起被占用。
- 條件引數各自獨立，不會互相影響。

Main

```
// Main
G01 X0. Y0. Z0.;
G200 X10. Y20. Z30.;                // 雖然G200只有讀取X引數
                                    // 但Y/Z引數也一起被占用
                                    // G200執行完畢後，會執行G01
                                    // 因為此行的引數均已被占用
                                    // 所以G01讀取不到任何引數。
G01 X0. Y0. Z0. F100.;
G201 P20. X10. Y20. Z30. F200.;     // G201只有讀取P引數
                                    // 所以XYZF引數都還沒有被占用
                                    // G201執行完畢後，執行G01
                                    // G01會讀取到X10. Y20. Z30. F200.
                                    // 并執行對應動作
M30;
```

G0200

```
// G0200
%@MACRO
@1:=#24;              // 只有讀取X引數。
M00;
WAIT();
M99;
```

G0201

```
// G0201
%@MACRO
@1:=#16;              // 只有讀取P引數。
M00;
WAIT();
M99;
```

04_H碼作為條件引數

- H碼作為條件引數。

Main

```
// Main
G01 X0. Y0. Z0.;
G200 X10. Y20. Z30. H40.;            // G200只有讀取H引數
                                     // H引數為條件引數
                                     // G200執行完畢後，執行G01
                                     // G01會讀取占用X10. Y20. Z30.
                                     // 并執行對應動作
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #11;      // 只有讀取H引數。
M00;
WAIT();
M99;
```

05_H碼作為軸向引數

- 【Pr3809 *UVWH為XYZC軸增量指令】設定為1，H碼除了是條件引數外，**同時**也被視為軸向引數。
- 但在MACRO中讀取H引數，僅視為條件引數被占用；因此在G碼巨集執行完畢后，H引數會再次作為軸向引數被占用，并執行對應動作。

Main

```
// Main
G01 X0. Y0. Z0.;
G200 X10. Y20. Z30. H40.;            // G200只有讀取H引數
                                     // 因為Pr3809=1，所以H引數同時被視為軸向引數及條件引數
                                     // 但在MACRO中其僅視為條件引數被占用
                                     // G200執行完畢後，執行G01
                                     // G01會讀取占用X10. Y20. Z30. H40.
                                     // 并執行對應動作
                                     // H40.是因被視為軸向引數的部分還未被占用，而再次被G01作為軸向引數占用
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #11;      // 只有讀取H引數。
M00;
WAIT();
M99;
```

- 在MACRO中讀取其他軸向引數，軸向引數只要其中一個被占用，則其他軸向引數都會一起被占用。
- 所以，H引數作為軸向引數的部分，同樣也會視為被占用。

Main

```
// Main
G01 X0. Y0. Z0.;
G200 X10. Y20. Z30. H40.;            // G200只有讀取X引數
                                     // 因為Pr3809=1，所以H引數同時被視為軸向引數及條件引數
                                     // 作為軸向引數的H引數也一起被占用
                                     // G200執行完畢後，執行G01
                                     // 因為此行的軸向引數均已被占用
                                     // 所以G01讀取不到任何引數
```

```
9                                                    // 這同樣也包括H引數作為軸向引數的部
分，也不會被讀取到
10  M30;
```

G0200

```
1   // G0200
2   %@MACRO
3   #101 := #24;      // 只有讀取X引數。
4   M00;
5   WAIT();
6   M99;
```

06_B碼作為軸向引數

B碼作為軸向引數

- 【Pr3806 啟動第二輔助碼B碼】設定為0，B碼是軸向引數。

Main

```
1   // Main
2   G01 X0. Y0. Z0.;
3   G200 X10. Y20. Z30. B40.;            // G200只有讀取B引數
4                                        // 因為Pr3806=0，所以B引數被判定為軸
向引數
5                                        // XYZ引數將被G200占用
6                                        // G200執行完畢後，執行G01
7                                        // 因為此行的引數均已被占用
8                                        // 所以G01讀取不到任何引數
9   M30;
```

G0200

```
1   // G0200
2   %@MACRO
3   #101 := #2;      // 只有讀取B引數。
4   M00;
5   WAIT();
6   M99;
```

07_B碼作為條件引數

【Pr3806 啟動第二輔助碼B碼】設定為1，B碼是條件引數。

Main

```
// Main
G01 X0. Y0. Z0.;
G200 X10. Y20. Z30. B40.;           // G200只有讀取B引數
                                    // 因為Pr3806=1，所以B引數被判定為條件引數
                                    // XYZ引數將不會被G200占用
                                    // G200執行完畢後，執行G01
                                    // G01會讀取占用X10. Y20. Z30.
                                    // 并執行對應動作
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #2;     // 只有讀取B引數。
M00;
WAIT();
M99;
```

08_多個G碼巨集在同一行

- 當多個G碼撰寫在同一行時，不管是G碼指令還是G碼巨集，均會依序對每一個G碼巨集進行解譯處理。
- 但只有寫在最後的G碼巨集可以讀取及占用引數，前面的G碼雖然會先解譯處理，但不會讀取任何引數，因此也不會占用任何引數

- **Main**

```
// Main
G01 X0. Y0. Z0.;
G200 G201 X10. P20.;        // 因為兩個G碼巨集寫在同一行
                            // 則只有最後一個G碼巨集可以讀取引數
                            // 即使G201沒有讀取到X引數
                            // G200仍然不能讀取X引數
                            // G200先執行完畢後，執行G201
                            // G201讀取占用P引數後，執行G01
                            // G01會讀取占用X10.
                            // 并執行對應動作
M30;
```

G200

```
// G0200
%@MACRO
#100 := #1000;                  // 備份插值模式
#101:=#24;                      // 讀取X引數，但讀不到。
G01 X#101;                      // 此行不會執行，因為#101沒有數值。
G01 X0.;                            // 此行會執行
M00;
WAIT();
G#100;                          // 還原插值模式
M99;
```

G201

```
// G0201
%@MACRO
#100 := #1000;                  // 備份插值模式
#101:=#16;                      // 讀取P引數，可以讀取得到。
G01 X#101;                      // 此行會執行
G01 X0.;
M00;
WAIT();
G#100;                          // 還原插值模式
M99;
```

09_G碼指令與G碼巨集在同一行

- 當多個G碼撰寫在同一行時，不管是G碼指令還是G碼巨集，均會依序對每一個G碼巨集進行解譯處理。
- 但只有寫在最後的G碼巨集可以讀取及占用引數，前面的G碼雖然會先解譯處理，但不會讀取任何引數，因此也不會占用任何引數

- **Main**

```
// Main
G90;
G00 X0. Y0. Z0.;                // G00 定位到X0 Y0 Z0
G200 G01 X10. F100.;            // 因為兩個G碼寫在同一行
                                // 則只有最後一個G碼可以讀取引數
                                // G200會執行，但讀取不到任何引數。
                                // G200執行完畢後，執行G01 X10. F100.;

G00 X0. Y0. Z0.;                // G00 定位到X0 Y0 Z0
G01 G201 X10. F100.;            // 因為兩個G碼寫在同一行
                                // G01先解譯，所以此時系統狀態變更為G01
```

```
                                    // 接著解譯G201
                                    // G201內部執行G01 X10. F100.
                                    // 因為G201有讀取占用軸向引數，所以所有軸向引數均已被占用
                                    // 接著G01發現沒有任何後續移動指令，所以沒有執行任何動作。

G00 X0. Y0. Z0.;                    // G00 定位到X0 Y0 Z0
G01 G202 X10. F100.;                // 因為兩個G碼寫在同一行
                                    // G01先解譯，所以此時系統狀態變更為G01
                                    // 接著解譯G202
                                    // G202內部沒有執行任何動作，也沒有讀取占用任何引數。
                                    // 接著G01發現軸向引數尚未被占用，所以執行G01 X10. F100.
M30;
```

G200

```
// G0201
%@MACRO
#100 := #1000;                      // 備份插值模式
#101 := #24;                        // 讀取X引數，但讀不到。
G#100 X#101 F#9;                    // 此行不會執行，因為#100沒有數值，但是此時#100=0，插值模式會變成G00
M00;
WAIT();
M99;
```

G201

```
// G0201
%@MACRO
#100 := #1000;                      // 備份插值模式
#101 := #24;                        // 讀取X引數，讀到X=10.。
G#100 X#101 F#9;                    // 此行會執行，且此時#100=1，所以是執行G01
M00;
WAIT();
M99;
```

G202

```
// G0202
%@MACRO
M00;
```

```
WAIT();
M99;
```

10_T碼巨集

- 【Pr3215 選刀時呼叫模式】設定為2，T碼巨集。
- 當T碼引數沒有被占用時，會執行T碼巨集。
- #20不會有值，R3也不會有值，但#1036會顯示T碼的數值。

- **Main**

```
// Main
G01 X0. Y0. Z0.;
T01 X10. Y20. Z30.;      // T碼引數沒有被占用 且 Pr3215=2
                         // 所以會執行T碼巨集
                         // T碼巨集中有讀取X引數
                         // 因為X是軸向引數，所以YZ引數也會一起被T碼巨集占用
                             // T01執行完畢後，執行G01
                         // 因為此行的引數均已被占用
                         // 所以G01讀取不到任何引數
M30;
```

T0000

```
// T0000
%@MACRO
#101 := #24;             // 只有讀取X引數。
M00;                     // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
                         // 觀察#20/#1036/#101
WAIT();
M99;
```

11_G碼巨集與T碼巨集在同一行

- 【Pr3215 選刀時呼叫模式】設定為2，T碼巨集。
- 當G碼巨集與T碼巨集在同一行時，會先解譯處理G碼巨集，再解譯T處理碼巨集。
- 如果G碼巨集占用了T碼，則T碼巨集將不執行。
- 如果G碼巨集不占用T碼，則T碼巨集會在G碼巨集執行完畢之後執行。
- 系統中只會有一個T0000，不過以下范例會以T0000_T01和T0000_T02來表示前後兩次T碼巨集。

- **Main**

```
// Main
G01 X0. Y0. Z0.;
T01 G200 X10. Y20. Z30. F1000;      // 解譯順序是先解譯G碼巨集
```

```
                                        // G碼巨集(G0200)內有讀取T碼引數
                                        // T碼引數視為被占用，所以不會執行T碼巨集
                                        // G碼巨集(G0200)執行完畢後，執行G01
                                        // G01會讀取占用X10. Y20. Z30. F1000
                                        // 并執行對應動作

T02 G201 X10. Y20. Z30. F1000.;         // 解譯順序是先解譯G碼巨集
                                        // G碼巨集(G0201)內沒有讀取T碼引數
                                        // T碼引數視為尚未占用
                                        // 所以G碼巨集(G0201)執行完畢後，會再執行T碼巨集
                                        // G碼巨集(G0201)雖然有占用X引數
                                        // 但是T碼巨集會讀取所有想要讀取的引數，所以還是讀取的到XYZF
                                        // 執行完T碼巨集後，會再執行G01
                                        // 因為此行的引數均已被占用
                                        // 所以G01讀取不到任何引數
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #20;            // 只有讀取T引數。
M00;
WAIT();
M99;
```

T0000_01

```
                        // 不會執行此程序
// T0000
%@MACRO
#201 := #1000;
#202 := #1004;
#101 := #24;
#102 := #25;
#103 := #26;
#104 := #9;
M00;                    // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G91 G01 X#101 Y#102 Z#103 F#104;

G#201;
G#202
```

```
M99;
```

G0201

```
// G0201
%@MACRO
#101 := #24;          // 只有讀取X引數。
M00;                  // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
M99;
```

T0000_02

```
// T0000
%@MACRO
#201 := #1000;
#202 := #1004;
#101 := #24;
#102 := #25;
#103 := #26;
#104 := #9;
M00;                  // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G91 G01 X#101 Y#102 Z#103 F#104;     // T碼巨集仍可以讀取到引數，進行對應動作。

G#201;
G#202
M99;
```

12_T碼巨集不受L引數影響

- 【Pr3215 選刀時呼叫模式】設定為2，T碼巨集。
- T碼巨集不會讀取也不會占用L碼引數

- **Main**

```
// Main
G01 X0. Y0. Z0.;
T01 L2;          // T0000雖然有讀取#12
                 // 但讀取到#12是1，而不是L引數給定的2
                 // T01執行完畢之後，執行G01
                 // G01會讀取占用L2
                 // 并執行對應動作(G01 L2沒有任何意義)
```

```
M30;
```

T0000

```
// T0000
%@MACRO
#101 := #12;          // 只有讀取L引數。
M00;                  // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
M99;
```

13_多個T碼巨集在同一行

- 【Pr3215 選刀時呼叫模式】設定為2，T碼巨集。
- 多個T碼巨集撰寫在同一行時，會依序對每一個T碼進行解譯處理，且每一個T碼巨集都可以讀取引數。
- 系統中只會有一個T0000，不過以下范例會以T0000_T01和T0000_T02來表示前後兩個T碼巨集。

Main

```
// Main
G01 X0. Y0. Z0.;
T01 T02 X10. Y20. Z30.;      // 此行會執行兩次
                             // 會先執行T01再執行T02
                             // 兩次均會讀取XYZ引數且均可順利讀取
                             // 不過後續若有其他巨集要讀取XYZ，則無法讀取，
視為被T碼巨集占用。
                             // T02執行完畢後，執行G01
                             // 因為此行的引數均已被占用
                             // 所以G01讀取不到任何引數
M30;
```

T0000_T01

```
// T0000_T01
%@MACRO
#100 := #1000;        // 備份插值模式
#101 := #24;          // 讀取X引數
#102 := #25;          // 讀取Y引數
#103 := #26;          // 讀取Z引數
IF #1036 = 1 THEN     // T02 → #1036=1
    G01 X#101;        // X軸移動
    G01 X0.;
END_IF;

IF #1036 = 2 THEN     // 此段不會執行
```

```
    G01 Y#102;
    G01 Y0.;
END_IF;

M00;                  // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;                          // 還原插值模式
M99;
```

T0000_T02

```
// T0000_T02
%@MACRO
#100 := #1000;        // 備份插值模式
#101 := #24;          // 讀取X引數
#102 := #25;          // 讀取Y引數
#103 := #26;          // 讀取Z引數
IF #1036 = 1 THEN     // 此段不會執行
    G01 X#101;        //
    G01 X0.;
END_IF;

IF #1036 = 2 THEN     // T02 → #1036=2
    G01 Y#102;        // Y軸移動
    G01 Y0.;
END_IF;

M00;                  // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;                              // 還原插值模式
M99;
```

14_M碼巨集

- 【Pr3601~3610 登錄M碼呼叫巨集】設定為123，登錄M0123為M碼巨集。
- M碼被登錄為M碼巨集時，需要在M碼沒有被占用，同時也沒有任何軸向引數被占用時，才會執行M碼巨集。
- M碼巨集執行時會占用除了T碼以外的所有引數。
- 不管M碼有沒有被當做M碼巨集使用，M碼引數(#13)都會被更新。
- 只有M碼被當做輔助碼時，M碼對應的R值才會被更新。M碼被當做巨集或者引數時，M碼對應的R值均不會更新。

- **Main**

```
// Main
G01 X0. Y0. Z0. F100.;
M123 X10. Y20. Z30. F300.;  // M碼引數沒有被占用
```

```
                                    // 也沒有任何軸向引數被占用
                                    // 所以會執行M碼巨集M0123
                                    // 因為M碼巨集執行後，除了會占用M碼巨集以讀取的引數以外，還會占用除了T碼以外的所有引數。
                                    // F引數雖然沒有被M0123讀取，但會一起被視為占用，所以進給速度不會被更新。
                                    // M123執行完畢後，執行G01
                                    // 因為此行的引數均已被占用
                                    // 所以G01讀取不到任何引數

G01 X0. Y0. Z0.;                    // 此行的F速度是F100. 因為上一行的F引數被M碼巨集占用。
M30;
```

M0123

```
// M0123
%@MACRO
#100 := #1000;                      // 備份插值模式
#101 := #24;                        // 讀取X引數
#102 := #25;                        // 讀取Y引數
#103 := #26;                        // 讀取Z引數
G00 X#101 Y#102 Z#103;              // G00移動XYZ軸向
M00;
WAIT();                             // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
                                    // 可以發現#13=123
                                    // 切換畫面到【PLC狀態】→【R Bit】
                                    // 可以發現R1=0
G#100;                              // 還原插值模式
M99;
```

15_G碼巨集與M碼巨集在同一行

- 【Pr3601~3610 登錄M碼呼叫巨集】設定為123，登錄M0123為M碼巨集。
- 當G碼巨集與M碼巨集寫在同一行時，會先解譯G碼巨集再解譯M碼巨集。
- 如果M碼被占用，則不會執行M碼巨集。
- 反之，如果M碼沒有被占用，則會在執行完G碼巨集後，在執行M碼巨集。
- M碼巨集執行完畢後，會占用除了T碼以外的所有引數。

Main

```
// Main
G01 X0. Y0. Z0.;
G1301 M1301 X10. Y20. Z30. P2 F1000.;    // 解譯順序是先解譯G碼巨集
                                         // G碼巨集(G1301)內有讀取M碼引數
```

```
                                                    // M碼引數視為被占用，所以不會執
行M碼巨集
                                                    // 但因為G碼巨集(G1301)內沒有占
用XYZF引數
                                                    // 所以執行完G碼巨集(G1301)後，
執行G01
                                                    // G01會讀取占用X10. Y20. Z30.
P2 F1000.
                                                    // 并執行對應動作

G1302 M1301 X10. Y20. Z30. F1000.;                  // 解譯順序是先解譯G碼巨集
                                                    // G碼巨集(G1302)內僅讀取P引數
                                                    // M碼引數視為尚未占用
                                                    // 也沒有任何軸向引數被占用
                                                    // 所以G碼巨集(G1302)執行完畢
後，會再執行M碼巨集
                                                    // M碼巨集(M1301)沒有讀取任何引
數
                                                    // 但在M碼巨集執行完畢後，除了T以
外的所有引數會被視為已被占用
                                                    // 執行完M碼巨集後，會執行G01
                                                    // 因為此行的引數均已被占用
                                                    // 所以G01讀取不到任何引數

M30;
```

G1301

```
// G1301
%@MACRO
#101 := #13;          // 只有讀取 M 引數。
M00;
WAIT();
M99;
```

M1301_01

```
                      // 不會執行此程序
// M1301
%@MACRO
M00;
WAIT();
M99;
```

G1302

```
// G1302
%@MACRO
#101 := #16;          // 只有讀取 P 引數。
M00;
WAIT();
M99;
```

M1301_02

```
// M1301
%@MACRO
M00;
WAIT();               // 此檔案不讀取任何引數
                      // 但卻會占用除了T碼以外的所有引數
M99;
```

16_G碼巨集與M碼巨集在同一行，G碼巨集讀取占用軸向引數

- 【Pr3601~3610 登錄M碼呼叫巨集】設定為123，登錄M0123為M碼巨集。
- 當G碼巨集與M碼巨集寫在同一行，會先解譯G碼巨集再解譯M碼巨集。
- 如果軸向引數已經被G碼巨集占用，則不會執行M碼巨集。

- **Main**

```
// Main
G01 X0. Y0. Z0.;
G200 M123 X10. Y20. Z30.;    // 解譯順序是先解譯G碼巨集
                             // G碼巨集(G0200)內有讀取X碼引數
                             // 所有的軸向引數均被G200占用
                             // 因為軸向引數已經被占用，所以不會執行M碼巨集。
                             // G碼巨集(G0200)執行完畢後，執行G01
                             // 因為此行的引數均已被占用
                             // 所以G01讀取不到任何引數
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #24;          // 只有讀取X引數。
```

```
M00;
WAIT();
M99;
```

M0123

```
                    //不會執行此程序
// M0123
%@MACRO
#101 := #24;
M00;
WAIT();
M99;
```

17_多個M碼巨集在同一行

- Pr3601設定為123，登錄M0123為M碼巨集。
- Pr3602設定為124，登錄M0124為M碼巨集。
- 多個M碼巨集撰寫在同一行時，只會解譯處理第一個M碼巨集。

- **Main**

```
// Main
G01 X0. Y0. Z0.;
M123 M124 X10. Y20. Z30.;    // 解譯順序會先遇到M123，所以只會執行M123
                             // M碼巨集(M0123)內會讀取X10. Y20. Z30.
                             // M碼巨集執行後，占用除了T碼以外的所有引數
                             // M碼巨集執行後，執行G01
                             // 因為此行的引數均已被占用
                             // 所以G01讀取不到任何引數
M30;
```

M0123

```
// M0123
%@MACRO
#100 := #1000;               // 備份插值模式
#101 := #24;                 // 讀取X引數
#102 := #25;                 // 讀取Y引數
#103 := #26;                 // 讀取Z引數
G00 X#101 Y#102 Z#103;       // G00移動XYZ軸向
M00;
WAIT();                      // 切換畫面到【診斷功能】→【共用變數】及【程式
變數】
```

```
                                    // 可以發現#13=124
                                    // 因為M124被當做引數，覆蓋掉#13
G#100;                              // 還原插值模式
M99;
```

M0124

```
                          // 不會執行此程式
// M0124
%@MACRO
#101 := #24;
M00;
WAIT();
M99;
```

18_T碼巨集與M碼巨集在同一行

- 【Pr3215 選刀時呼叫模式】設定為2，T碼巨集。
- 【Pr3601 登錄M碼巨集】設定為123/124/125，登錄 M0123/M0124/M0125 為M碼巨集。
- 如果T碼巨集與M碼巨集寫在同一行，則寫在前面的巨集會先被解譯。
- 此時後面一個巨集是否執行，則依照前述各項章節【巨集特性】來決定。

- **Main**

```
// Main
G01 X0. Y0. Z0.;
T01 M1601 X10. Y20. Z30.;    // 解譯順序會先遇到T01，再解譯M1601
                             // T碼巨集沒有占用任何引數，因此M碼巨集(M1601)
將被執行
                             // X=10.
                             // M碼巨集(M1601)沒有讀取任何引數
                             // M碼巨集(M1601)執行後，會占用除了T碼引數外的
所有引數。
                             // Y=10.
                             // M碼巨集(M1601)執行後，執行G01
                             // 因為此行的引數均已被占用
                             // 所以G01讀取不到任何引數
M1602 T02 X10. Y20. Z30.;    // 解譯順序會先遇到M1602，再解譯T02
                             // M碼巨集(M1602)沒有讀取任何引數
                             // M碼巨集(M1602)執行後，會占用除了T碼引數外的
所有引數。
                             // Y=20.
                             // M碼巨集(M1602)沒有占用T碼引數，因此T碼巨集
(T0000_02)將被執行
                             // X=20.
                             // T碼巨集沒有讀取任何引數。
```

```
                                // T碼巨集執行完畢後，執行G01
                                // 因為此行的引數均已被占用
                                // 所以G01讀取不到任何引數
M1603 T03 X10. Y20. Z30.;       // 解譯順序會先遇到M1603，再解譯T03
                                // M碼巨集(M1603)只有讀取T引數
                                // M碼巨集(M1603)執行後，會占用所有引數，包含T引數。
                                // Y=30.
                                // M碼巨集(M1603)占用T碼引數，因此T碼巨集(T0000_03)不會被執行。
                                // 因為此行的引數均已被占用
                                // 所以G01讀取不到任何引數
M30;
```

T0000_01

```
// T0000
%@MACRO
#100 := #1000;          // 備份插值模式
#101 := #1004;          // 備份絕對/增量命令模式

G91 G00 X10.;           // X軸增量移動X10.

M00;                    // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;                  // 還原插值模式
G#101;                  // 還原絕對/增量命令模式
M99;
```

M1601

```
// M1601
%@MACRO
#100 := #1000;          // 備份插值模式
#101 := #1004;          // 備份絕對/增量命令模式

G91 G00 Y10.;           // Y軸增量移動Y10.

M00;                    // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;                  // 還原插值模式
G#101;                  // 還原絕對/增量命令模式
M99;
```

M1602

```
// M1602
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #1004;       // 備份絕對/增量命令模式

G91 G00 Y10.;        // Y軸增量移動Y10.

M00;                 // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;               // 還原插值模式
G#101;               // 還原絕對/增量命令模式
M99;
```

T0000_02

```
// T0000
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #1004;       // 備份絕對/增量命令模式

G91 G00 X10.;        // X軸增量移動X10.

M00;                 // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;               // 還原插值模式
G#101;               // 還原絕對/增量命令模式
M99;
```

M1603

```
// M1603
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #1004;       // 備份絕對/增量命令模式
#102 := #20;         // 讀取#20的數據，此行會占用T引數

G91 G00 Y10.;        // Y軸增量移動Y10.

M00;                 // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
WAIT();
G#100;               // 還原插值模式
G#101;               // 還原絕對/增量命令模式
```

```
13  M99;
```

T0000_03

```
1                       // 不會執行此程式
2   // T0000
3   %@MACRO
4   #100 := #1000;      // 備份插值模式
5   #101 := #1004;      // 備份絕對/增量命令模式
6
7   G91 G00 X10.;       // X軸增量移動X10.
8
9   M00;                // 切換畫面到【診斷功能】→【共用變數】及【程式變數】
10  WAIT();
11  G#100;              // 還原插值模式
12  G#101;              // 還原絕對/增量命令模式
13  M99;
```

19_非模態呼叫巨集（G65）

- G65會執行P引數指定之巨集檔案。

- **Main**

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

O0001

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

20_模態呼叫巨集（G66）

- G66會在每一個移動單節結束後執行。

- **Main**

```
// Main
G01 X-5. Y-5. Z-5.;
G200 G66 P1 X10. Y20. Z30. ;     // 這行的G200無法讀取X引數
G200 X10.;                       // 這行的G200可以讀取X引數
                                 // G200內有兩行G01移動單節
                                 // 每執行一行就會執行一次G66 P1 X10.
Y20. Z30.;
G67;
M30;
```

G0200_01

```
// G0200_01
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #24;         //第一次執行G200，讀取不到X引數
G01 X#101;           //所以此行不會動作
G01 X0.;
M00;
WAIT();
G#100;               // 還原插值模式
M99;
```

G0200_02

```
// G0200_02
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #24;         // 第二次執行G200，可以讀取X引數
G01 X#101;               // 所以此行可以動作
                     // 此行是移動單節，執行完畢後會執行一次 G66 P1 X10.
Y20. Z30.
G01 X0.;             // 此行是移動單節，執行完畢後會執行一次 G66 P1 X10.
Y20. Z30.
M00;
WAIT();
G#100;               // 還原插值模式
M99;
```

O0001

```
// O0001
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #24;
#102 := #25;
#103 := #26;
G91 G01 X#101 Y#102 Z#103
M00;
WAIT();
G#100;               // 還原插值模式
M99;
```

21_非模態呼叫巨集（G66.1）

- G66.1會在每一個單節結束後執行。

- **Main**

```
// Main
G01 X-5. Y-5. Z-5.;
G66.1 P1 X10. Y20. Z30. ;
G200 X10.;                    // 這行的G200可以讀取X引數
                              // G200內每一行單節執行完畢後均會執行G66.1 P1
X10. Y20. Z30.
G67;                          // 此行單節，執行完畢後會執行一次 G66 P1 X10.
Y20. Z30.
M30;
```

G0200

```
// G0200
%@MACRO
#100 := #1000;       // 備份插值模式
#101 := #24;         // 第二次執行G200，可以讀取X引數
G01 X#101;           // 所以此行可以動作
                     // 此行單節，執行完畢後會執行一次 G66 P1 X10. Y20.
Z30.
G01 X0.;             // 此行單節，執行完畢後會執行一次 G66 P1 X10. Y20.
Z30.
M00;                     // 此行單節，執行完畢後會執行一次 G66 P1 X10.
Y20. Z30.
WAIT();              // 此行不算單節
G#100;               // 還原插值模式
```

```
                        // 此行單節，執行完畢後會執行一次 G66 P1 X10. Y20.
Z30.
M99;                    // 此行單節，執行完畢後會執行一次 G66 P1 X10. Y20.
Z30.
```

O0001

```
// O0001
%@MACRO
#100 := #1000;          // 備份插值模式
#101 := #1004;          // 備份絕對/增量命令模式
#111 := #24;
#112 := #25;
#113 := #26;
G91 G01 X#111 Y#112 Z#113;
M00;
WAIT();
G#100;                  // 還原插值模式
G#101;                  // 還原絕對/增量命令模式
M99;
```

22_G65及G66/G66.1必須是該行最後一個G碼

- 非模態呼叫巨集G碼（G65）以及模態呼叫巨集G碼（G66/G66.1）必須是該行最後一個G碼。

Main

```
// Main
G01 X0. Y0. Z0.;
G65 P1 X10. Y20. Z30. G200; // 此行會發出警報COR-013
                            // 因為G65和G200在同一行
                            // 但G65不是最後一個G碼
M30;
```

G0200

```
// G0200
%@MACRO
#101 := #24;
M00;
WAIT();
M99;
```

O0001

```
// O0001
%@MACRO
#101 := #24;
#102 := #25;
#103 := #26;
M00;
WAIT();
M99;
```

14.2 MACRO XML資料應用

- MACRO可以使用特殊的函數讀取xml檔案，分別是DBLOAD和DBOPEN，DBOPEN用來載入xml資料檔案，DBLOAD用來讀取資料內容。
- 應用範例：下圖為某產機客製人機畫面，該畫面將自行產生一xml檔案，以記錄相關加工資料，之後在巨集規劃動程時，同步讀取該xml內容，以作為規劃依據。

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						

-> 該客製人機首先將使用者設定內容輸出成xml檔案，其語法格式定義如下，並且將該xml檔案存放於使用者所指定的GNCFILES路徑中（參閱Pr3219說明）：

```
<?xml version="1.0" encoding="UTF-16"?>
<CycleFile>
    <Cycle Name="Cycle_HerdonProg"> ←第一筆資料開頭
        <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> ←第一筆資料結尾
    <Cycle Name="Cycle_HerdonProg"> ←第二筆資料開頭
```

```
        <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> ←第二筆資料結尾
    <Cycle Name="Cycle_HerdonProg"> ←第三筆資料開頭

        <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> ←第三筆資料結尾
    <Cycle Name="Cycle_HerdonProg"> ←第四筆資料開頭

        <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> ←第四筆資料結尾
    <Cycle Name="Cycle_HerdonProg"> ←第五筆資料開頭

        <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> ←第五筆資料結尾

</CycleFile>
```

-> 使用者需自行編撰xml資料結構設定檔，用來宣告當巨集利用DBLOAD函數讀取資料時，要將所讀取的相關資料，存放至哪些對應變數中。
其語法格式定義如下，並且要將此設定檔儲存於OCRes\\Common\\Schema\\中。
若為Dipole前後台連線時，schema檔案放置於**後台控制器**之OCRes\\Common\\Schema\\中。

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

    <Cycle name="Cycle_HerdonProg">

        <Field>

            <Name>Col_X</Name>
            <InputStorage>@1200</InputStorage> ←Col_X存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>
        <Field>

            <Name>Col_Y</Name>
            <InputStorage>@1201</InputStorage> ←Col_Y存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>
        <Field>
```

```
            <Name>Col_Z</Name>
            <InputStorage>@1202</InputStorage> ←Col_Z存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>
        <Field>

            <Name>Col_A</Name>
            <InputStorage>@1203</InputStorage> ←Col_A存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>
        <Field>

            <Name>Col_B</Name>
            <InputStorage>@1204</InputStorage> ←Col_B存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>
        <Field>

            <Name>Col_C</Name>
            <InputStorage>@1205</InputStorage> ←Col_C存入的變數中
            <InputFormat>Variant</InputFormat>
            <DefaultValue></DefaultValue>

        </Field>

    </Cycle>

</Schema>
```

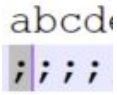
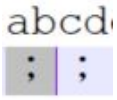
-> MACRO範例

```
// 載入GNCFILES\\Test資料數，總共5筆，因此@1:=5;
@1:=DBOPEN("Test");


// 載入第1筆資料，DBLOAD引數為0
// @1200=0.00 @1201=17.63 @1202=12.98 @1203=267.54
DBLOAD( 0 );


// 載入第2筆資料，DBLOAD引數為1
// @1200=2.26 @1201=21.34 @1202=13.20 @1203=300.87
DBLOAD( 1 );
```

14.3 建議PC端編輯環境

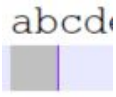
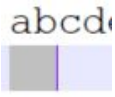
- notepad++：https://notepad-plus-plus.org/downloads/
  - 開啟顯示所有字元：檢視→特殊字元→顯示所有字元
    - 可以檢視 Marco 檔案是否有未支援字元

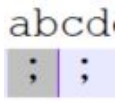
常見字元判斷

名稱	圖式	說明	是否支援
半形空白符號	 (灰色選取部分)	半形空白符號為紅色點點，一點代表一格空白	是
全形空白符號	 (空白，灰色選取部分)	全形空白符號為完全空白，較不易排查	否
半形分號符號	 (灰色選取部分)	半形分號符號如左圖所示	是
全形分號符號	 (灰色選取部分)	全形分號符號如左圖所示	否

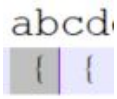
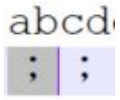
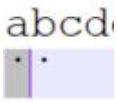
- 切換編碼 (語系)：編碼→字元集
    - 可以切換顯示的語系，協助排查問題
- notepad++ 標準排查步驟
    i. 確認使用的編碼
        1. ANSI、UTF-8：跳至第二點
        2. 繁體中文編碼 (Big5) 與簡體中文編碼 (GB2312)：建議切換兩種編碼檢視，因為如果檔案中有全形符號，顯示內容可能會改變，造成排查問題。
           編碼切換顯示差異

           繁體中文→簡體中文

名稱	繁體中文編碼	簡體中文編碼	是否支援
全形空白符號	 (空白，灰色選取部分)	 (空白，灰色選取部分)	否

名稱	繁體中文編碼	簡體中文編碼	是否支援
全形分號符號	 (灰色選取部分)	 (空白，灰色選取部分)	否

簡體中文→繁體中文

名稱	簡體中文編碼	繁體中文編碼	是否支援
全形空白符號	 (空白，灰色選取部分)	 (灰色選取部分)	否
全形分號符號	 (灰色選取部分)	 (灰色選取部分)	否

ii. 確認是否有未支援字元
- 全形空白/分號建議排查方式
  a. 點選搜尋→搜尋
  b. 在尋找內容輸入全形空白/分號
  c. 點選在目前文件中全部尋找
  d. 如有找到，在搜尋結果中會顯示
  e. 修改為支援字元

14.4 MACRO 支援字元

14.4.1 支援字元

僅支援 ASCII 字元

- 可顯示字元
  - 數字：0-9
  - 英文字母
    - A-Z
    - a-z
  - 其他（半形）

!	"	#	$	%	&	'	|
(	)	*	+	,	-	.	/

:	;	<	>	=	?	@	~
[	]	\	^	_	`	{	}
(space)							

- 控制字元（Unicode 表示法）

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		

ASCII 字元詳細內容請參考：ASCII