附錄的目錄

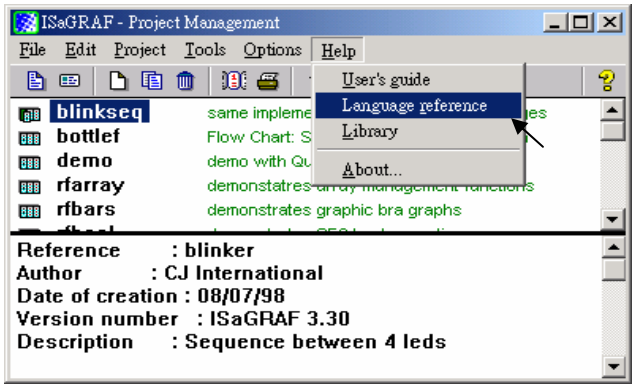
附錄 A ： ISaGRAF 提供的函式及函式方塊

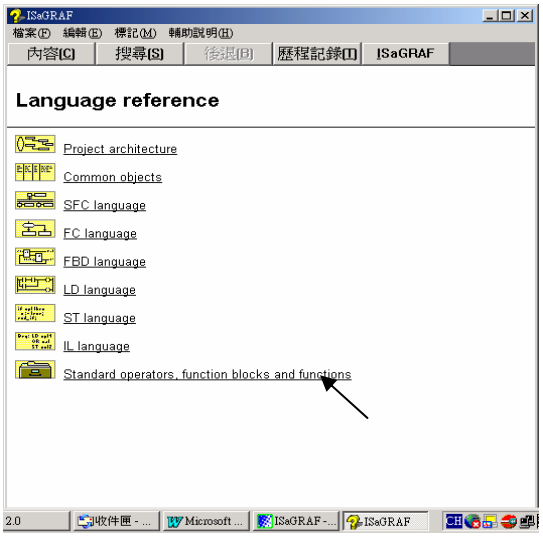
附錄 A.1: 標準 ISaGRAF 函式方塊

下面介紹支持 I-8xx7, I-7188EG/XG & Wincon-8xx7/8 xx6 控制器的 ISaGRAF 標準功能方塊，若有 “*” 或 ”#” 的 表示不被 I-8xx7 & I-7188EG/XG 控制器支持, 但 Wincon-8xx7/8xx6 只有 “#” 不支持.

-	ARWRITE	*F_ROPEN	MSG	SHR
& (AND)	ASCII	F_TRIG	MUX4	SIG_GEN
*	ASIN	*F_WOPEN	MUX8	SIN
/	ATAN	*FA_READ	Neg	SQRT
+	AVERAGE	*FA_WRITE	NOT_MASK	SR
<	BLINK	FIND	ODD	STACKINT
<=	BOO	*FM_READ	#OPERATE	#SYSTEM
<>	CAT	*FM_WRITE	OR_MASK	TAN
=	CHAR	HYSTER	POW	TMR
=1 (XOR)	CMP	INSERT	R_TRIG	TOF
>	COS	INTEGRAL	RAND	TON
>=	CTD	LEFT	REAL	TP
>=1 (OR)	CTU	LIM_ALRM	REPLACE	TRUNC
1 gain	CTUD	LIMIT	RIGHT	XOR_MASK
ABS	#DAY_TIME	LOG	ROL	
ACOS	DELETE	MAX	ROR	
ANA	DERIVATE	MID	RS	
AND_MASK	EXPT	MIN	SEL	
ARCREATE	*F_CLOSE	MLEN	SEMA	
ARREAD	*F_EOF	MOD	SHL	

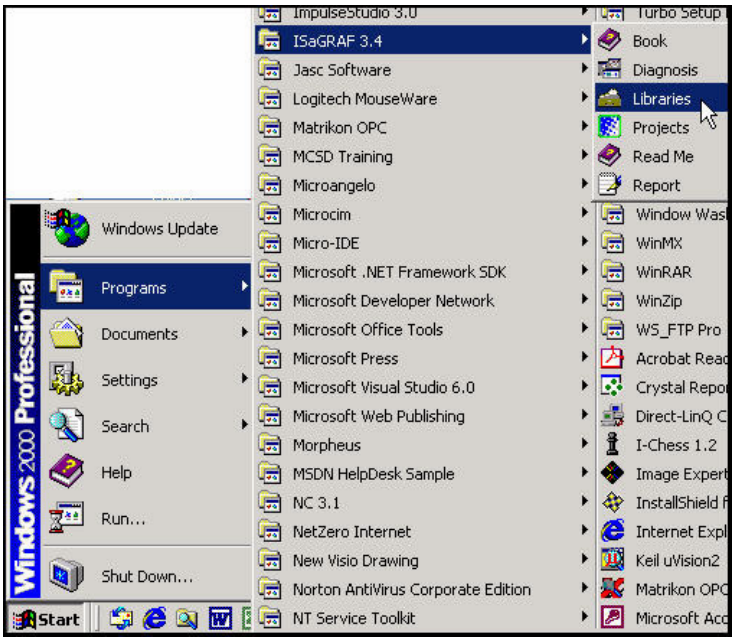
請參考 ISaGRAF 內的線上說明



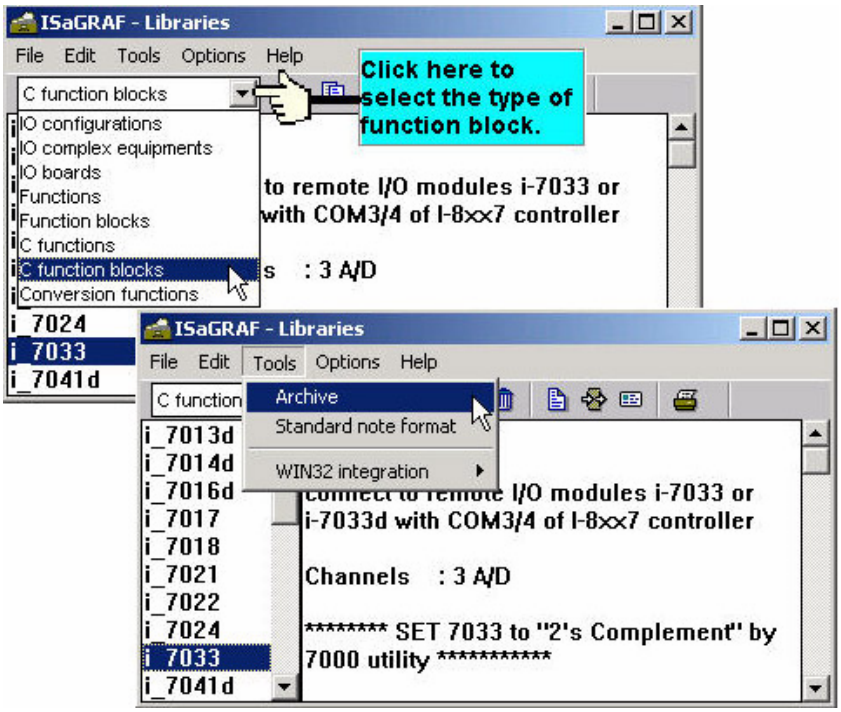


附錄 A.2: 增加新的功能方塊到 ISaGRAF 的 I/O 函式庫

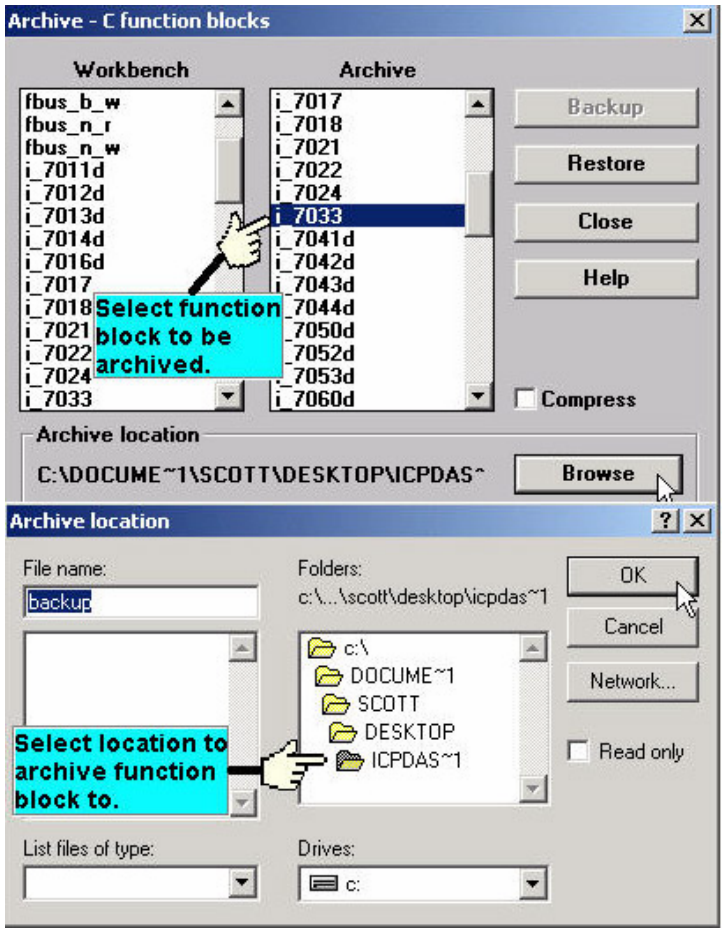
按[Start]→[Program]→[ISaGRAF3.4]的 “Libraries”來開始安裝或更新 ISaGRAF 函式或功能方塊。



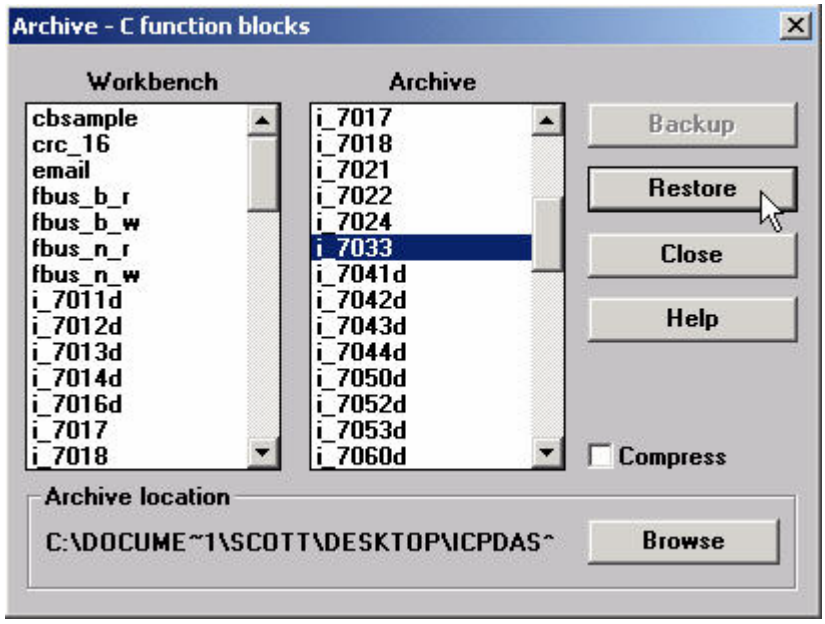
當您按下 “Libraries”， “ISaGRAF Libraries”視窗將會開啓。選擇 “Tools”裡的 “Archive”選項來增加新的 C 函式方塊 (C function Block)。



按下你所要 “Archive” 的檔名，然後按 “Browse”去選擇您所要增加的 C 功能方塊，檔案放置於 CD_ROM 內的 \Napdos\ISaGRAF\ARK\ 內

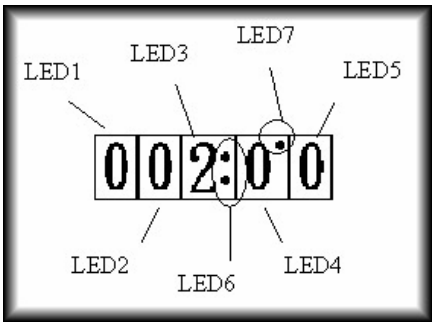


在 “Archive”視窗中，選擇您所要增加的 C 功能方塊，按下 “Restore”鍵，當您按下 “Restore”鍵後方塊即已加入到 ISaGRAF workbench 中了。



附錄 A.3: 七段式顯示 LED 的定義表格

在下面的表格中，我們提供了 I-8xx7 & I-7188EGD & I-7188XGD 控制器中七段式顯示 LED 的定義表格。



LED 6: 若設爲 TURE 則顯示":" (冒號):
LED 7: 若設爲 TURE 則顯示"." (點)

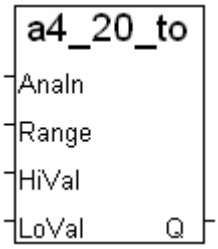
顯示表: LED 1 至 LED 5

顯示的字元	設的值	顯示的字元	設的值	顯示的字元	設的值
0	0	4.	20	r	40
1	1	5.	21	L	41
2	2	6.	22	n	42
3	3	7.	23	y	43
4	4	8.	24	U	44
5	5	9.	25	P	45
6	6	A.	26	o	46
7	7	b.	27	r.	47
8	8	C.	28	n.	48
9	9	d.	29	y.	49
A	10	E.	30	h.	50
b	11	F.	31	L.	51
C	12		32	U.	52
d	13		33	P.	53
E	14	‾	34	o.	54
F	15		35	.	55
0.	16	_	36	‾.	56
1.	17	H	37	_.	57
2.	18	h	38	r	其他
3.	19	H.	39		

附錄 A.4: 給 ICP DAS 的 ISaGRAF 控制器使用的功能方塊

下面的函式方塊是專門爲 I-8xx7, I-7188EG/XG, µPAC-7186EG & Wincon-8xx7/8xx6 控制器設計的. 型態若爲 Standard_Function 或 Standard_Function Block 表示爲 ISaGRAF 提供的標準 Function 與 Function Block. 若爲 C_Function 或 C_Function Block 則爲 ICP DAS Controller 提供的 Function 與 Function Block.

A4_20_TO



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 Analog Input 值 從 4 – 20 mA 變爲 User 自定的工程數值

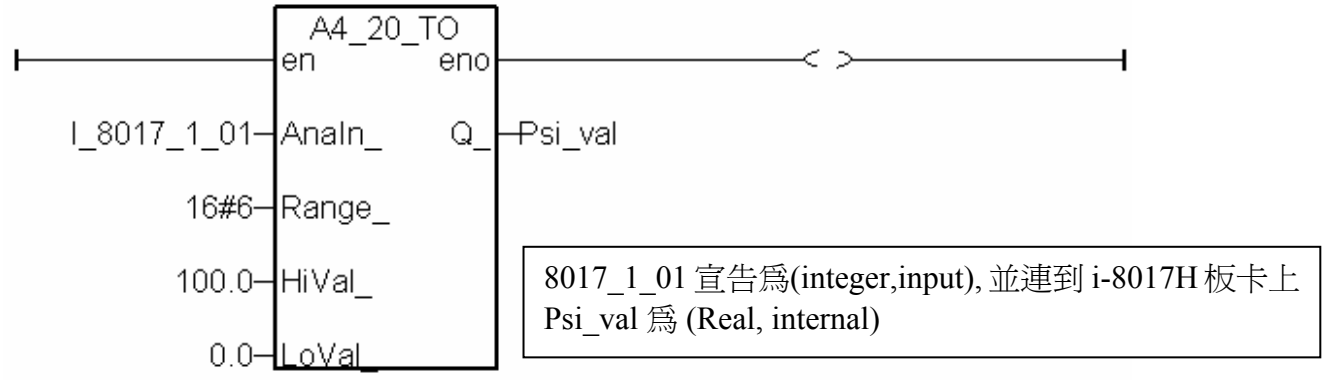
輸入參數 :

AnaIn_	Integer	對應於 Analog Input 的整數變數. 此變數之值通常爲-32768 ~ +32767. 與 IO 板卡的設定有關.
Range_	Integer	Analog Input 板卡的 Range 設定值 16#6 : -20 to +20 mA, 16#7 : 4 to +20 mA 16#D : -20 to +20 mA, 16#1A : 0 to +20 mA,
HiVal_	Real	User 自定的工程數值 上限值 (Analog input 爲 20 mA 時的值)
LoVal_	Real	User 自定的工程數值 下限值 (Analog input 爲 4 mA 時的值) 例如,轉換 I-8017H 的值 由 4 ~ 20 mA 變爲 0 ~ 100 psi. 請設 HiVal_ 爲 100.0 , LoVal_爲 0.0 , Range_ 爲 16#6 (參照板卡的 range 設定)

傳回值 :

Q_	Real	轉換後的 User 工程數值. 若發生錯誤, 如 Range_設錯, 會回傳 1.23E-20

範例 : 轉換 I-8017H 的電流輸入, Range=6: (-20 to +20 mA), 由 (4mA, +20mA)轉到 (0, 100 psi).
4 mA 代表 0 psi , 20 mA 代表 100 psi

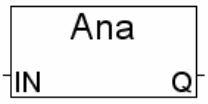


注意: 1. 請參考類似的 function: to_A4_20 , to_V0_10 , A4_20_to , V0_10_to.

2. 使用 A4_20_to, To_A4_20, To_V0_10, V0_10_to 等 function, 需將 driver 更新爲 i-7188EG: 2.16 版, i-7188XG:2.14 版, i-8xx7:3.18 版, 或更高的版本, 程式才不會有問題(較舊的 driver 會發生程式 run 一段時間後會停止的現象).

ANA

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

轉換任何資料型態的數值 成爲 Integer

輸入參數 :

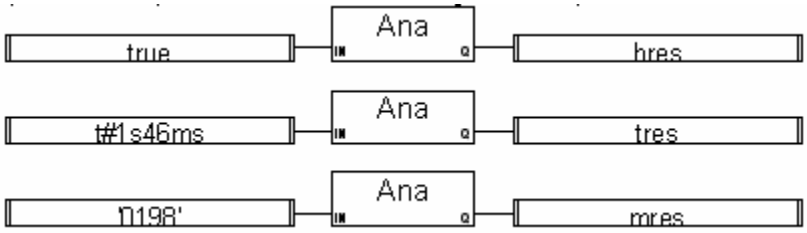
IN	Any	任何非整數的數值

傳回值 :

Q	Integer	若 IN 爲 FALSE 則傳回 0 / 若 IN 爲 TRUE 則傳回 1 Timer 則傳回毫秒數值 Real 值則傳回整數 String 則傳回對應的十進位數值

範例 :

(* FBD 範例*)



(* ST 相等式 : *)

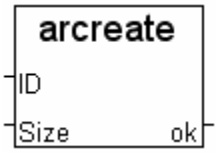
```
bres := ANA (true);              (* bres = 1 *)
tres := ANA (t#1s46ms);          (* tres = 1046 *)
mres := ANA ('0198');            (* mres = 198 *)
r1 := ANA (3.27);                (* r1 = 3 *)
```

(* IL 相等式 : *)

```
LD      true
ANA
ST      bres
LD      t#1s46ms
ANA
ST      tres
LD      '0198'
ANA
ST      mres
```

ARCREATE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

配置一塊整數(或實數)記憶區, 給 ISaGRAF 程式暫存資料用

輸入參數 :

ID	Integer	WinCon 只支援設爲 1, 不可設成其他數字
SIZE	Integer	可以是 (1 ~ 3,000,000) WinCon 最多可配置 3,000,000 個 32-bit Integer 記憶空間. (換算爲 byte 共 3,000,000 x 4 = 12,000,000 bytes)

傳回值 :

OK	Integer	1: 配置成功; 其他: 失敗.

注意 :

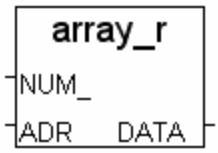
1. W-8xx7/8xx6 只能使用 ARcreate 一次, 如下:
   在一開機後的第一個 PLC Scan 配置此記憶區:

```
IF  INIT  THEN
      INIT  :=  False;
      TMP_v  :=  ARcreate(1, 2000000);
END_IF;
(* INIT 初值爲 True,   TMP_v 型態爲 Internal Integer *)
```

2. 請參考第 11.3.10 節的範例說明
3. W-8XX7 的驅動程式, 需爲 3.36 版起才有支援 ARCREATE, ARREAD 與 ARWRITE.

ARRAY_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

從 byte 陣列內讀出 1 個 byte (unsigned 8-bit)

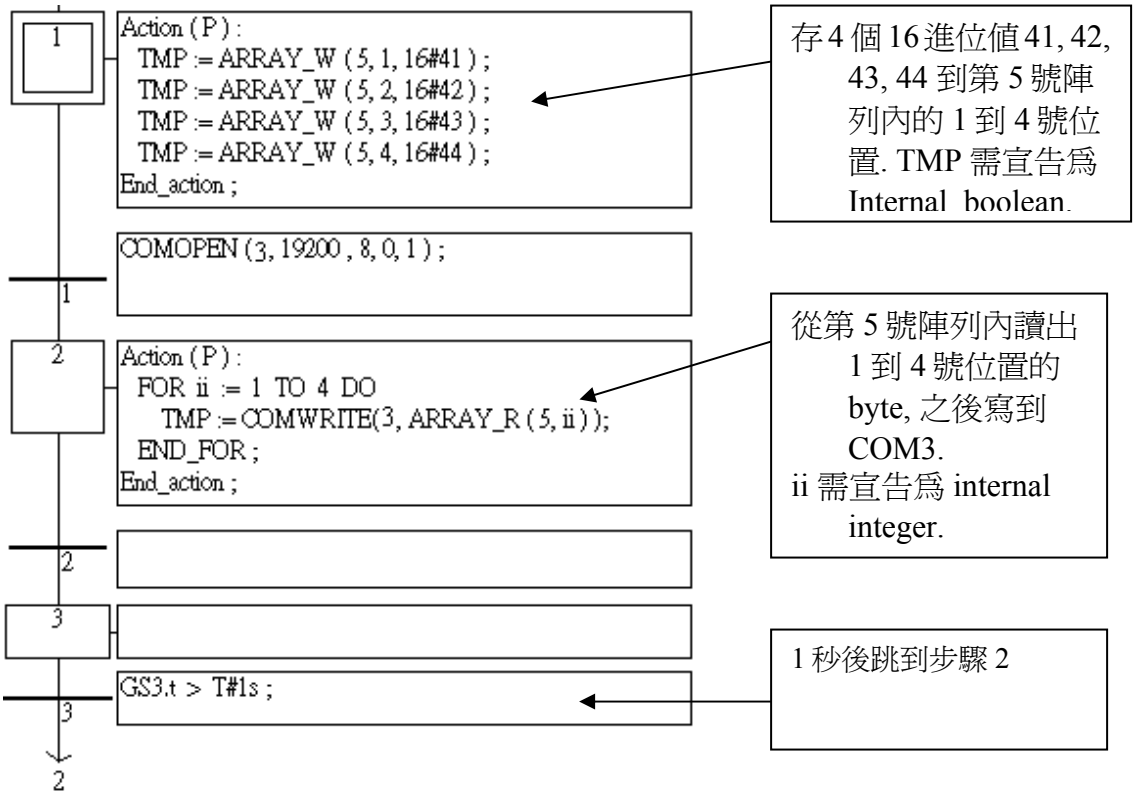
輸入參數 :

NUM_	Integer	所使用的陣列編號. 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 24 , Wincon-8xx7: 1 到 48
ADR_	Integer	所使用的陣列內的位置編號. 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 256 , Wincon-8xx7: 1 到 512

傳回值 :

DATA_	Integer	取得的 byte 值 (0~255)

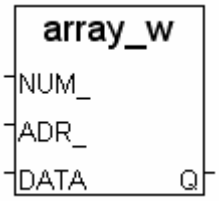
範例 :



注意: 存到陣列內的資料關電後會消失.

ARRAY_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

存 1 個 byte(unsigned 8-bit) 到 byte 陣列

輸入參數 :

NUM_	Integer	所使用的陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 24 , Wincon-8xx7: 1 到 48
ADR_	Integer	所使用的陣列內的位置編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 256 , Wincon-8xx7: 1 到 512
DATA_	Integer	要存進去的 byte 值, 範圍爲 0 到 255.

傳回值 :

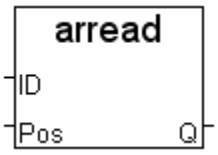
Q_	Boolean	成功 回傳 TRUE, 失敗回傳 FALSE

範例: 請參考 “ARRAY_R” 的範例

注意: 存到陣列內的資料關電後會消失.

ARREAD

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

從 User 配置的整數記憶區讀取一個整數 (32-bit signed)

輸入參數 :

ID	Integer	WinCon 只支援設爲 1, 不可設成其他數字
POS	Integer	要讀取哪個位址的整數, 可以是 (1 ~ 3,000,000) 若 POS 超出 ARcreate 配置的數量, 資料會不正確

傳回值 :

Q	Integer	讀到的整數.

ARWRITE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

寫一個整數到 User 配置的記憶區

輸入參數 :

ID	Integer	WinCon 只支援設爲 1, 不可設成其他數字
POS	Integer	要寫到哪個位址, 可以是 (1 ~ 3,000,000) 若 POS 超出 ARcreate 配置的數量, 資料不會寫入
IN	Integer	要寫入的整數.

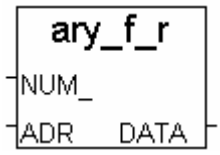
傳回值 :

OK	Integer	1: 成功; 其他值爲失敗.

注意 :

1. 若未使用過 ARcreate 先配置記憶區, ARread 與 ARwrite 的傳回值都是錯的
2. 若要在該記憶區內讀/寫實數, 請配合使用 Real_Int 與 Int_Real 函式 (請參考第 11.3.10 節的範例)
3. W-8XX7 的驅動程式, 需爲 3.36 版起才有支援 ARCREATE, ARREAD 與 ARWRITE.

ARY_F_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 Float 陣列內讀出 1 個實數值 (32-bit float)

輸入參數 :

NUM_	Integer	所使用的陣列編號. W-8xx7/8xx6 範圍爲 1 到 18. 7188EG/XG, I-8xx7 範圍爲 1 到 6.
ADR_	Integer	所使用的陣列內的位置編號. 範圍爲 1 到 256

傳回值 :

DATA_	Real	取得的實數值 (32-bit float)

ARY_F_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

存 1 個實數值(32-bit float) 到 float 陣列

輸入參數 :

NUM_	Integer	所使用的陣列編號. W-8xx7/8xx6 範圍爲 1 到 18. 7188EG/XG, I-8xx7 範圍爲 1 到 6.
ADR_	Integer	所使用的陣列內的位置編號. 範圍爲 1 到 256
DATA_	Real	要存進去的實數值 (32-bit float)

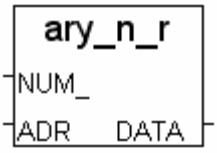
傳回值 :

Q_	Boolean	成功 回傳 TRUE, 失敗回傳 FALSE

注意:

1. 存到陣列內的資料關電後會消失.
2. I-7188EG/XG 與 I-8XX7 的 Float 陣列跟 Integer 陣列使用相同的記憶區, 請小心使用, 當使用 ARY_F_R 讀 REAL 值卻發現該記憶位址內存的是 Integer 資料時, 值會錯誤.
3. 從以下版本起的驅動程式才有支援 ARY_F_R 與 ARY_F_W:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

ARY_N_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 integer 陣列讀出 1 個長整數 (signed 32-bit)

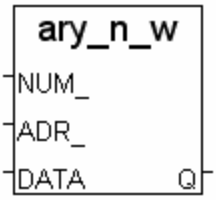
輸入參數 :

NUM_	Integer	所使用的陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 6 , Wincon-8xx7: 1 到 18
ADR_	Integer	所使用的陣列內的位置編號, 範圍爲 1 到 256

傳回值 :

DATA_	Integer	取得的長整數值

ARY_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

存 1 個長整數(signed 32-bit) 到 integer 陣列

輸入參數 :

NUM_	Integer	所使用的陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 6 , Wincon-8xx7: 1 到 18
ADR_	Integer	所使用的陣列內的位置編號, 範圍爲 1 到 256
DATA_	Integer	要存進去的長整數值

傳回值 :

Q_	Boolean	成功 回傳 TRUE, 失敗回傳 FALSE

注意: 1. Integer 陣列和 word 陣列使用同一塊記憶區. 請小心安排使用.

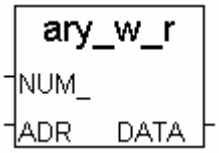
word 陣列(編號,位置)	Integer 陣列(編號,位置)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

2. 存到陣列內的資料關電後會消失.

範例: 請參考 “ARRAY_R” 的範例

ARY_W_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

從 word 陣列讀出 1 個 word (signed 16-bit)

輸入參數 :

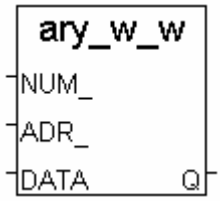
NUM_	Integer	所使用的陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 12 , Wincon-8xx7: 1 到 36
ADR_	Integer	所使用的陣列內的位置編號, 範圍爲 1 到 256

傳回值 :

DATA_	Integer	取得的 word 值, 範圍爲–32768 到 +32767

ARY_W_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

存 1 個 word (signed 16-bit) 到 word 陣列

輸入參數 :

NUM_	Integer	所使用的陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 12 , Wincon-8xx7: 1 到 36
ADR_	Integer	所使用的陣列內的位置編號, 範圍爲 1 到 256
DATA_	Integer	要存進去的 word 值 (-32768 到 +32767)

傳回值 :

Q_	Boolean	成功 回傳 TRUE, 失敗回傳 FALSE

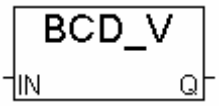
注意: 1. Integer 陣列和 word 陣列使用同一塊記憶區. 請小心安排使用.

word 陣列(編號,位置)	Integer 陣列(編號,位置)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

2. 存到陣列內的資料關電後會消失.

範例: 請參考 “ARRAY_R” 的範例

BCD_V



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 BCD 值到 1 個 整數值

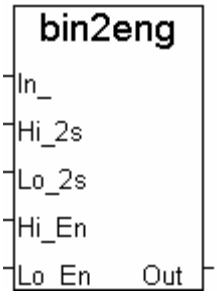
輸入參數 :

IN_	Integer	要被轉換的 BCD 值

傳回值 :

Q_	Integer	轉換後得到的值,例如, 16#12345 → 12345 18 → 12

BIN2ENG



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 IO 板卡的 2 的補數值 到 1 個 整數值

輸入參數 :

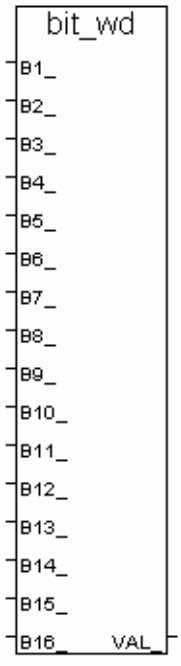
IN_	Integer	要被轉換的 2's complement 值
HI_2S_	Integer	2's complement 的上限值
LO_2S_	Integer	2's complement 的下限值
HI_EN_	Integer	轉換後的值 的上限值
LO_EN_	Integer	轉換後的值 的下限值

傳回值 :

OUT_	Integer	轉換後得到的值,例如, HI_2s_ = 32767 , LO_2s_ = -32768, HI_EN_ = 1000, LO_EN_ = -1000 IN_ = 16383 → OUT_= 500 IN_ = -12345 → OUT_= -377

注意: HI_2S_ 不可跟 LO_2S_.值相同, 且二者之值需在 (-32768 到 +32767)區間內

BIT_WD



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 16 個 Boolean 值到 1 個 word 值 (signed 16-bit)

輸入參數 :

B1_ ~ B16_	Boolean	要被轉換的 16 個 boolean 值

傳回值 :

VAL_	Integer	轉換後得到的 word 值 例如, B1_ 和 B2_爲 TRUE, 而其它皆爲 FALSE, 得到 VAL_ 爲 3. 如果 B4_ 爲 TRUE, 而其它皆爲 FALSE, 得到 VAL_ 爲 8

BLINK

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function Block

產生一個 ON_OFF 方波閃爍的訊號

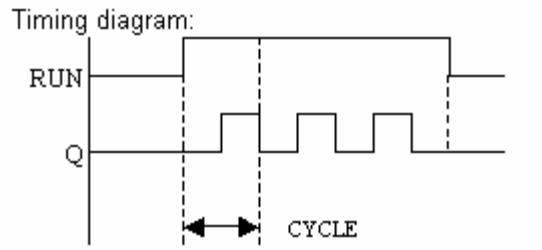
輸入參數 :

RUN	Boolean	模式: TRUE=閃爍; FALSE=不閃爍, 並會將 Q 輸出爲 FALSE
CYCLE	Timer	閃爍週期

傳回值 :

Q	Boolean	輸出的 ON_OFF 方波閃爍訊號

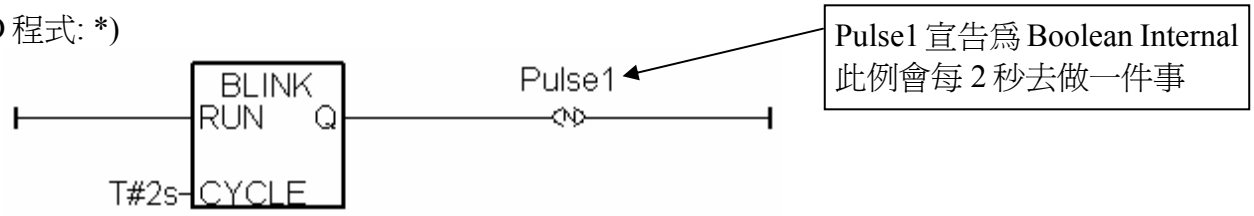
時序圖 :



應用說明:

BLINK 方塊可用來每隔固定一段時間就產生一個 Pulse True, 所以可以使用在每固定一段時間就做一件事的應用上. 如下:

(* LD 程式: *)




```
(* ST 程式: *)
    IF  Pulse1  THEN
        (* 做一件事 *)
        (*  ……      *)
    END_IF;
```

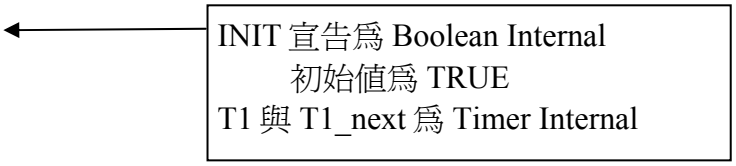
但以上的程式在時間間隔較短, 比如小於 200ms 或 PLC Scan Time 較大時會變得不精確. 例如每 50ms 做一件事, 因爲 50ms 是比較小的間隔, 跟 PLC Scan Time 比較接近, 就會不準確, 所以可以改成以下方法, 就可以提高準確性.

(見下頁)

ST 程式:

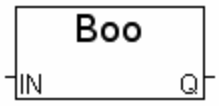
```
IF   INIT  THEN
     INIT  :=  False;
     T1  :=  T#0s;
     T1_next  :=  T1 + T#50ms;
     Tstart (T1);
END_IF;

IF   T1 >= T1_next  THEN
     IF   T1 > T#22h  THEN
          T1 := T#0s;
          T1_next := T#0s;
     END_IF;
     T1_next := T1_next + T#50ms;
    (* 做一件事 *)
    (*  ……       *)
END_IF;
```



BOO

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

轉換任何資料型態的變數爲布林型態

輸入參數 :

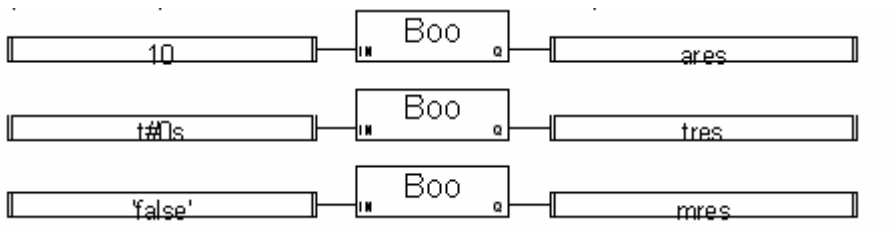
IN	Any	任一非布林的值

傳回值 :

Q	Boolean	非 0 的數值傳回 TRUE, 0 則傳回 FALSE 'TRUE' message 傳回 TRUE, 'FALSE' message 傳回 FALSE

範例 :

(* FBD 範例 "Convert to Boolean" blocks *)



(* ST 相等式: *)

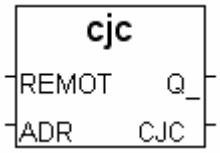
```
ares := BOO (10);                (* ares is TRUE *)
tres := BOO (t#0s);              (* tres is FALSE *)
mres := BOO ('false');           (* mres is FALSE *)
```

(* IL 相等式: *)

```
LD      10
BOO
ST      ares
LD      t#0s
BOO
ST      tres
LD      'false'
BOO
ST      mres
```

CJC

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function Block

讀取 I-7018/7019 和 I-87018/87019 模組的冷端補償值
(Cold-Junction Compensation, 簡稱 CJC)

輸入參數 :

REMOTE_ :	Boolean	必需是常數, 不能爲變數 若 I-7018/7019 和 I-87018/87019 爲遠端 I/O 模組 則 REMOTE_ 爲 TRUE. 若 I-87018/87019 插在主要控制器上 則 REMOTE_ 爲 FALSE.
ADR_ :	Integer	必需是常數, 不能爲變數 若 REMOTE_ 爲 TRUE, ADR_ 爲遠端 I/O 模組的位址 (1 ~ 255). 若 REMOTE_ 爲 FALSE, ADR_ 爲插槽編號 0~7 (W-8xx7/8xx6 則爲插槽 1~7).

傳回值 :

Q_ :	Boolean	若正常運作, 傳回 TRUE 若 Q_ 爲 FALSE, 表示通訊不良, 則下列回傳值無意義.
CJC_ :	Integer	傳回類比輸入值 (2 的補數格式) D3B4 ---> 0000 ---> 7FFF (十六進位制) 代表值 : -11340 ---> 0 ---> 32767 (十進位制) 溫度範圍 : -45 ---> 0 ---> +130 (攝氏度)

注意 : (非常重要)

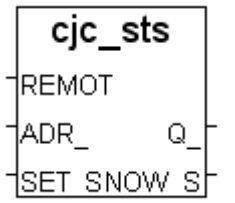
1. 若 I-87018 / 87019 是插在主控制器上, 請先連接 87018 / 87019 I/O 卡, 否則設定的 "CJC" 函式方塊會無法運作.

2.若 I-7018 / 7019 和 I-87018 / 87019 做爲 RS-485 遠端 I/O, 請先以 DCON utility 將格式設定爲 2 的補數.

3. W-8xx7/8xx6 適用於他的驅動程式 3.21 版本以上

CJC_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

啓動/停止 I-7018/7019 和 I-87018/87019 模組的冷端補償
(Cold-Junction Compensation, 簡稱 CJC)

輸入參數 :

REMOTE_ :	Boolean	必需是常數, 不能爲變數 若 I-7018/7019 和 I-87018/87019 作爲遠端 I/O 模組 則 REMOTE_ 爲 TRUE. 若 I-87018/87019 插在主要控制器上 則 REMOTE_ 爲 FALSE.
ADR_ :	Integer	必需是常數, 不能爲變數 若 REMOTE_ 爲 TRUE, ADR_ 爲遠端 I/O 模組的位址 (1 ~ 255). 若 REMOTE_ 爲 FALSE, ADR_ 爲插槽編號 0~7.
SET_STS_ :	Boolean	若 SET_STS_ 爲 FALSE, 可停止 CJC 補償 若 SET_STS_ 爲 TRUE, 可啓動 CJC 補償

傳回值 :

Q_ :	Boolean	若正常運作, 傳回 TRUE 若 Q_ 爲 FALSE, 表示通訊不良, 下列回傳值無意義.
NOW_STS_ :	Boolean	若 NOW_STS_ 爲 FALSE, CJC 補償目前狀態爲停止 若 NOW_STS_ 爲 TRUE, CJC 補償目前狀態爲啓動

注意 (非常重要):

1. 若 I-87018 / 87019 是插在主控制器上, 請先連接 87018 / 87019 I/O 卡, "CJC_STS" 函式方塊才會運作.
2. 若是使用爲 RS-485 Remote I/O, 連接之前請注意下列各項 :

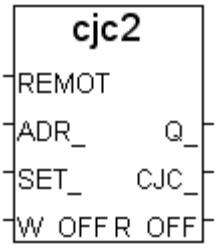
A. 在 I/O 模組端, 請使用 "DCON Utility" 來設定 :

*1. 設定 "address" 爲獨一無二的位址編號. (1~255)
*2. 類比輸入模塊需設爲 "2's complement" 格式, 類比輸出模塊需設爲 "Engineering" 格式
*3. 設定通訊參數的 "baud rate" 及 "8,N,1"
*4. 若爲類比板卡要設定爲 "Range Type"

B. 在 I-8xx7/I-7188EG/I-7188XG /W-8xx7 主控端 :

*1. 連結 I/O 請選擇 “complex equipment “ 及 "bus7000B", 並設定其 "baud rate" 與 I/O 模組的 "baud rate" 相同, Checksum 也需相同.
*2. I-8xx7, 7188EG/XG 最多可連接 64 個遠程 I/O 模組, W-8xx7 最多 255 個
*3. 程式內連接 RS-485 Remote I/O 模組請選用 i_7*** function block 或 i_87*** Function Block.

CJC2



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function Block

讀取 I-7018/7019 和 I-87018/87019 模組偏移的冷端補償值

輸入參數 :

REMOTE_ :	Boolean	必需是常數, 不能爲變數 若 I-7018/7019 和 I-87018/87019 作爲遠端 I/O 模組 則 REMOTE_ 爲 TRUE. 若 I-87018/87019 插在主要控制器上 則 REMOTE_ 爲 FALSE.
ADR_ :	Integer	必需是常數, 不能爲變數 若 REMOTE_ 爲 TRUE, ADR_ 爲遠端 I/O 模組的位址 (1 ~ 255). 若 REMOTE_ 爲 FALSE, ADR_ 爲插槽編號 0~7.
SET_ :	Boolean	若爲 TRUE 則開始設定 CJC 溫度的偏移
W_OFFSET_	Integer	CJC 溫度的偏移量, 十進位值, -4096 ~ +4096, 以 0.01 攝氏度爲一單位

傳回值 :

Q_ :	Boolean	若正常運作, 傳回 TRUE 若 Q_ 爲 FALSE, 表示通訊不良, 下列回傳值無意義.
CJC_ :	Integer	傳回類比輸入值 (2 的補數格式) D3B4 ---> 0000 ---> 7FFF (十六進位制) 代表值 : -11340 ---> 0 ---> 32767 (十進位制) 溫度範圍 : -45 ---> 0 ---> +130 (攝氏度)
R_OFFSET_:	Integer	讀取偏移的 CJC 溫度值

注意 (非常重要):

1. 若 I-87018 / 87019 是插在主控制器上, 請先連接 87018 / 87019 I/O 卡, "CJC2 " 函式方塊才會運作.
2. 若是使用爲 RS-485 Remote I/O, 連接之前請注意下列各項 :

A. 在 I/O 模組端, 請使用 "DCON Utility" 來設定 :

*1. 設定 "address" 爲獨一無二的位址編號. (1~255)

*2. 類比輸入模塊需設爲 "2's complement" 格式, 類比輸出模塊需設爲 "Engineering" 格式

*3. 設定通訊參數的 "baud rate" 及 "8,N,1"

*4. 若爲類比板卡要設定爲 "Range Type"

B. 在 I-8xx7/I-7188EG/I-7188XG /W-8xx7 主控端 :

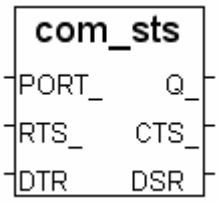
*1. 連結 I/O 請選擇 “complex equipment “ 及 "bus7000B", 並設定其 "baud rate" 與 I/O 模組的 "baud rate" 相同, Checksum 也需相同.

*2. I-8xx7, 7188EG/XG 最多可連接 64 個遠程 I/O 模組, W-8xx7 最多 255 個

*3. 程式內連接 RS-485 Remote I/O 模組請選用 i_7*** function block 或 i_87*** Function Block.

COM_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

設定COM埠的 RTS, DTR 並取得 CTS, DSR 狀態

輸入參數 :

PORT_ :	Integer	COM 埠號. 3:COM3, 4:COM4, 5:COM5… I-8417/8817/8437/8837 有效埠號: 3, 4 & 5 I-7188XG & I-7188EG 有效埠號: 4
RTS_ :	Boolean	TRUE : 設定 RTS 啓用, FALSE : 設定 RTS 閒置
DTR_ :	Boolean	TRUE : 設定 DTR 啓用, FALSE : 設定 DTR 閒置

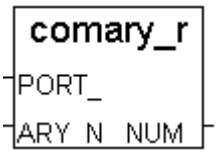
傳回值 :

Q_ :	Boolean	TRUE 表示 OK , FALSE 表示沒有成功
CTS_ :	Boolean	取得 CTS 狀態
DSR_ :	Boolean	取得 DSR 狀態

注意 :

1. 使用 Comopen( ) 開啓 COM 埠.

COMARY_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 COM 埠讀取全部已經收到的 byte (unsigned 8-bit), 並存入 byte 陣列內

輸入參數 :

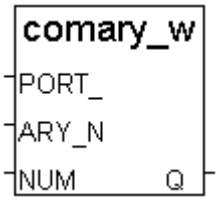
PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …
ARY_NO_	Integer	Byte 陣列編號(I-8xx7 & I-7188EG/XG:1-24, W-8xx7:1-48), 該陣列會儲存從 Com 收到的 byte

傳回值 :

NUM_	Integer	從 Com 埠讀到的 byte 的數量 (0~256)

重要 : 串列通訊資料需要時間來傳遞, COMARY_R 被呼叫當時若已經有多少 byte 傳進來, 就會存入同樣數量的 byte 到指定的陣列內, 還未傳進來的資料, 當然不會在陣列內 (因爲還沒收到).

COMARY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫出 byte 陣列內的 數個 byte (unsigned 8-bit) 到 1 個 COM 埠

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …
ARY_NO_	Integer	要寫出的 Byte 陣列編號(I-8xx7 & I-7188EG/XG:1 - 24) , **(**W-8xx7:1 - 48)
NUM_	Integer	從該 Byte 陣列的第 1 號位置起寫出多少個 byte

傳回值 :

Q_	Boolean	成功回傳 TRUE

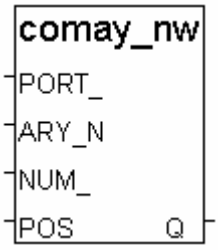
注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

範例:

請參考第 11 章 - Demo_21, 22 & 23.
請參考 附錄 A.4 : “ARRAY_R” & “ARRAY_W” 的說明

COMAY_NW



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫出 Integer 陣列內的 數個長整數 (signed 32 bit) 到 1 個 COM 埠

每個長整數包含 4 個 byte. 而且是有正負號的.
寫出長整數到 Com 會以 INTEL 格式內的長整數排法來寫出.
[lowest byte] [ ] [ ] [highest byte]
例: 假如有 3 個長整數要寫出, 第 1 個爲 16#04030201 (67,305,985), 第 2 個爲 16#08070605 (134,678,021) , 第 3 個爲 16#FFFFFFFE (-2).
則寫出的 12 個 byte 依序爲 [01] [02] [03] [04] [05] [06] [07] [08] [FE] [FF] [FF] [FF]

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8 W-8xx7:2,3, or …
ARY_NO_	Integer	要寫出的 integer 陣列編號(I-8xx7 & I-7188EG/XG:1 - 6) , **(**W-8xx7:1 - 18)
NUM_	Integer	要寫出多少個 integer
POS_	Integer	從該 integer 陣列內的第幾號位置起開始寫出 (1-256) 假如 POS_+NUM_ > 257, 則只有 (257-POS_) 個長整數會寫出 例. POS_=255, NUM_=3, 則只有 2 個長整數會寫出, 分別爲第 255 及 256 號位置.

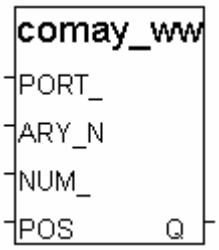
傳回值 :

Q_	Boolean	成功回傳 TRUE

注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援
* Integer 陣列和 word 陣列使用同一塊記憶區. 請小心安排使用 (參考 Ary_N_W)

COMAY_WW



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫出 Word 陣列內的 數個 Word (signed 16 bit) 到 1 個 COM 埠

每個 Word 包含 2 個 byte. 而且是有正負號的 (-32768 ~ +32767).
寫出 Word 到 Com 會以 INTEL 格式內的 word 排法來寫出.
[low byte] [high byte]
例: 假如有 3 個 word 要寫出, 第 1 個爲 16#0403 (1,027), 第 2 個爲 16#0807 (2,055) , 第 3 個爲 16#FFFE (-2).
則寫出的 6 個 byte 依序爲 [03] [04] [07] [08] [FE] [FF]

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8 W-8xx7:2,3, or …
ARY_NO_	Integer	要寫出的 word 陣列編號(I-8xx7 & I-7188EG/XG:1 - 12) , **(**W-8xx7:1 - 36)
NUM_	Integer	要寫出多少個 word
POS_	Integer	從該 word 陣列內的第幾號位置起開始寫出 (1-256) 假如 POS_+NUM_ > 257, 則只有 (257-POS_) 個 word 會寫出 例. POS_=255, NUM_=3, 則只有 2 個 word 會寫出, 分別爲第 255 及 256 號位置.

傳回值 :

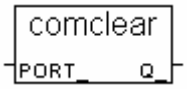
Q_	Boolean	成功回傳 TRUE

注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援
* Integer 陣列和 word 陣列使用同一塊記憶區. 請小心安排使用 (參考 Ary_W_W)

COMCLEAR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

清空 Com 埠的 接收 buffer

輸入參數 :

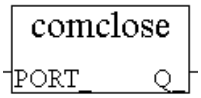
PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …

傳回值 :

Q_	Boolean	成功回傳 TRUE

COMCLOSE

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

關閉 COM 埠

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …

傳回值 :

Q_	Boolean	成功回傳 TRUE

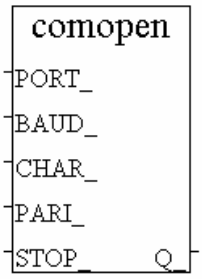
注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

範例:

請參考 COMOPEN 的範例

COMOPEN



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

開啓 COM 埠

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8 W-8xx7:2,3, or …
BAUD_	Integer	通訊速率, 可設成 300, 600, 1200, 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	Integer	字元大小, 7 或 8
PARI_	Integer	同位檢查, 0: 沒有, 1: 偶數, 2: 奇數, 3: 標記, 4: 空白字元 **3 和 4 只適用於 I-8xx7: COM3~20; I-7188EG/XG: COM3~8; Wincon-8xx7: COM2, 或, …**
STOP_	Integer	停止位元大小, 1 或 2

傳回值 :

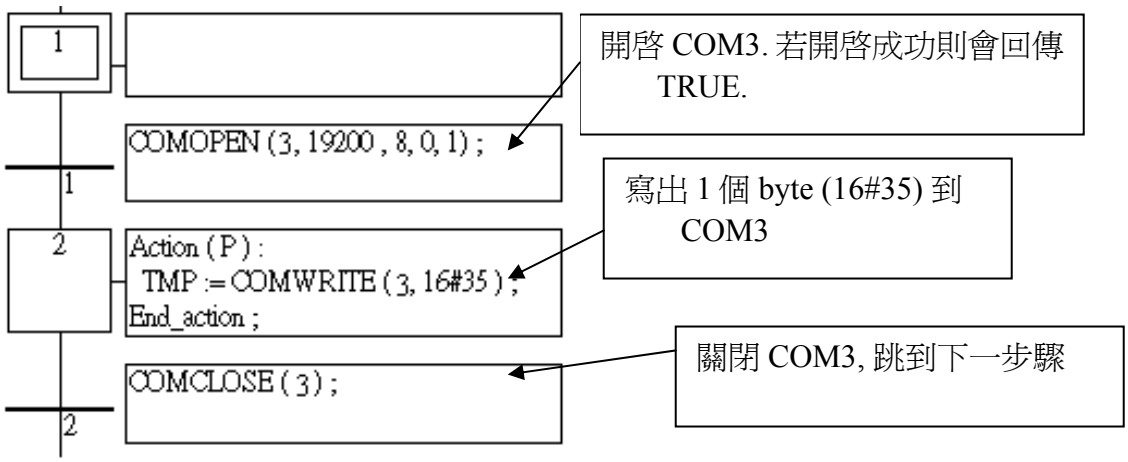
Q_	Boolean	成功回傳 TRUE

注意:

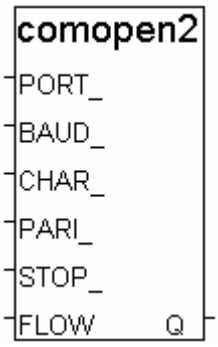
* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

範例:

請參考第 11 章 - Demo_21, 22 & 23.



COMOPEN2



□ I-8417/8817 □ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

開啓可做 Flow Control 的 RS232 埠

輸入參數 :

PORT_	Integer	port 編號, I-7188EG/XG:3~8; W-8xx7:2, 或 …
BAUD_	Integer	通訊速率, 可設成 300, 600, 1200, 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	Integer	字元大小, 7 或 8
PARI_	Integer	同位檢查, 0: 沒有, 1: 偶數, 2: 奇數, 3: 標記, 4: 空白字元 **3 和 4 只適用於 I-8xx7: COM3 ~ 20; I-7188EG/XG: COM3 ~8; Wincon-8xx7: COM2, 或, …**
STOP_	Integer	停止位元大小, 1 或 2
FLOW_	Boolean	True: 硬體 flow control (CTS / RTS) (7188EG/XG 3 ~ 5), False: 軟體 flow control (XON / XOF) (7188EG/XG 3 ~ 8)

傳回值 :

Q_	Boolean	成功回傳 TRUE

注意:

若使用 W-8xx7 的 COM2, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 “快速上手手冊”)

COMREAD

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 Com 埠讀取 1 個 byte (unsigned 8-bit)

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …

傳回值 :

Q_	Integer	讀到的 byte 值 (0~255)

注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 "快速上手手冊")
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

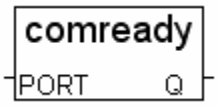
*** 需先使用 COMREADY 去測試有無資料從 COM 進來. 有, 才能使用, 不然使用本 Function 可能會造成通訊鎖死的狀況(Dead Lock).**

範例:

請參考 "COMREADY" 的範例

COMREADY

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-80xx7/8xx6



型態 : C_Function

測試 COM 埠有無資料進來

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …

傳回值 :

Q_	Boolean	只要有資料進來就回傳 TRUE. (即使只進來一個 byte)

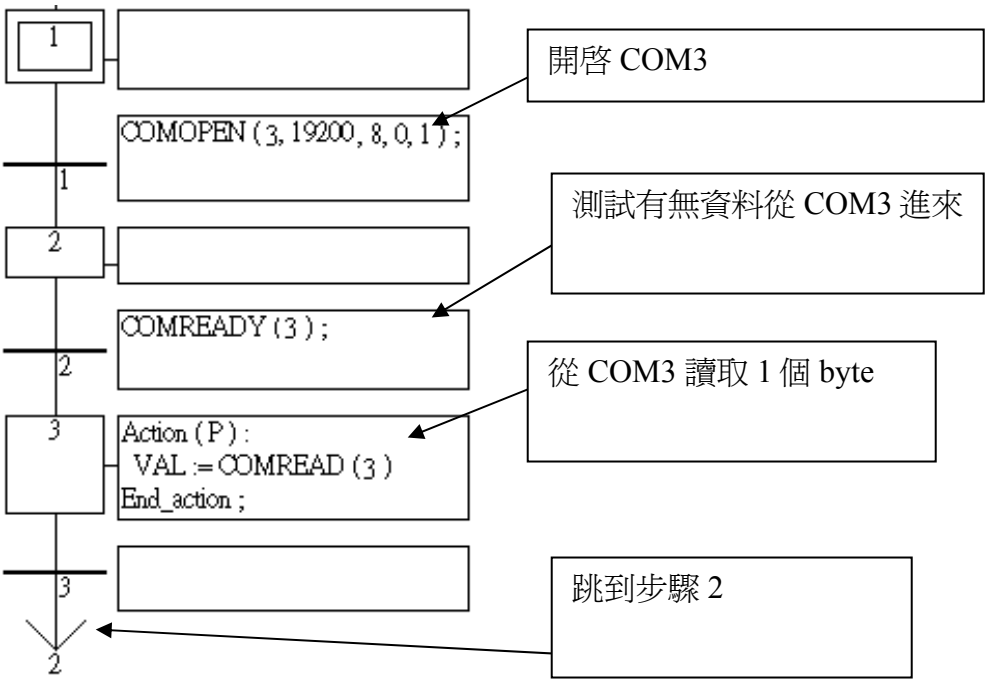
注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 "快速上手手冊")
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

<u>* 使用 COMREAD 之前要先使用 COMREADY 去測試有無資料從 COM 進來. 有, 才能使用, 不然使用 COMREAD 可能會照成通訊鎖死的狀況(Dead Lock)</u>

範例:

請參考第 11 章 - Demo_21, 22 & 23.



COMSTR_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個字串到 COM 埠

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2,3, 或 …
STR_	Message	要寫出的字串(最大長度爲 255).

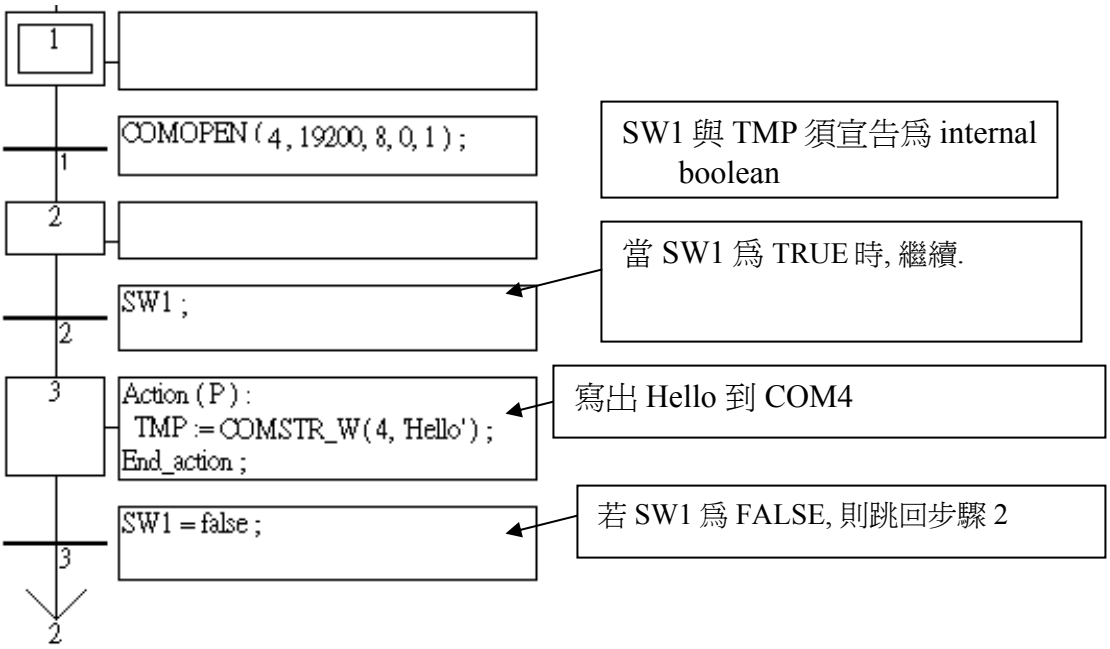
傳回值 :

Q_	Boolean	成功回傳 TRUE

注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 "快速上手手冊")
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

範例:



COMWRITE

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個 byte (unsigned 8-bit) 到 COM 埠

輸入參數 :

PORT_	Integer	port 編號, I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8 W-8xx7:2,3, or …
DATA_	Integer	要寫出的 byte (0 ~ 255)

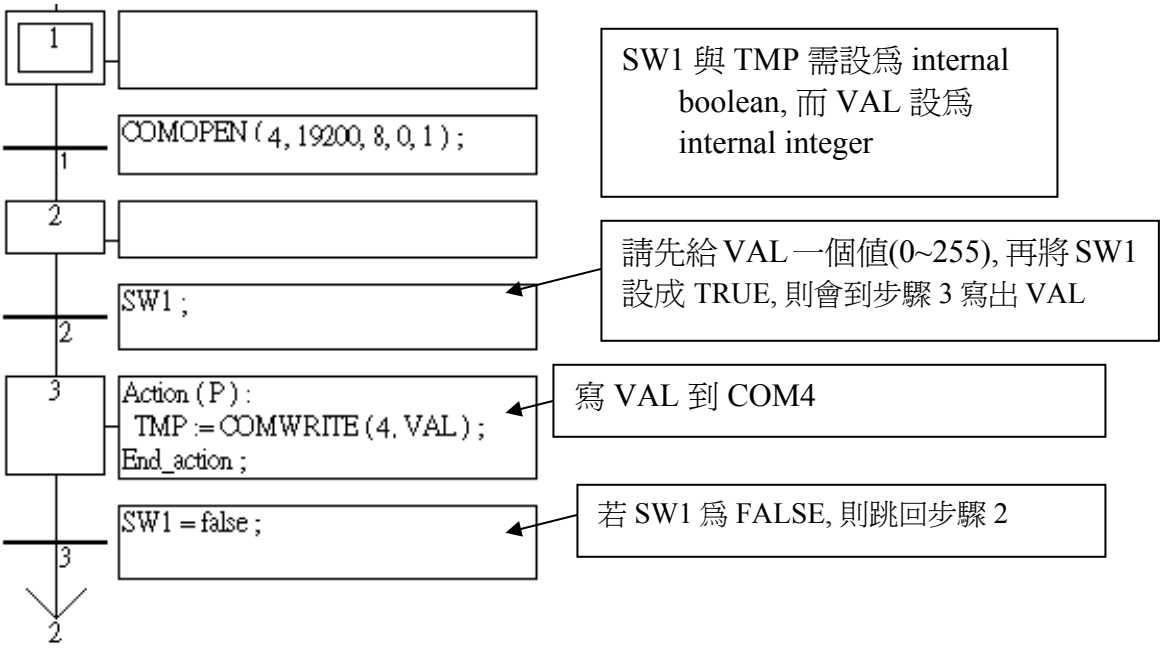
傳回值 :

Q_	Boolean	成功回傳 TRUE

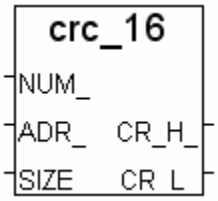
注意:

* 若使用 I-8xx7 & I-7188EG 的 COM1, 請先設定 COM1 爲 非 Modbus-RTU port(參考附錄 C.1)
* 若使用 W-8xx7 的 COM2 或 COM3, 請先設定他們爲 非 Modbus-RTU port (參考 W-8xx7 的 "快速上手手冊")
* I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 不支援

範例:



CRC_16



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6
採用第 5.3 節的演算法.

型態 : C_Function Block

計算 CRC-16 檢查碼

輸入參數 :

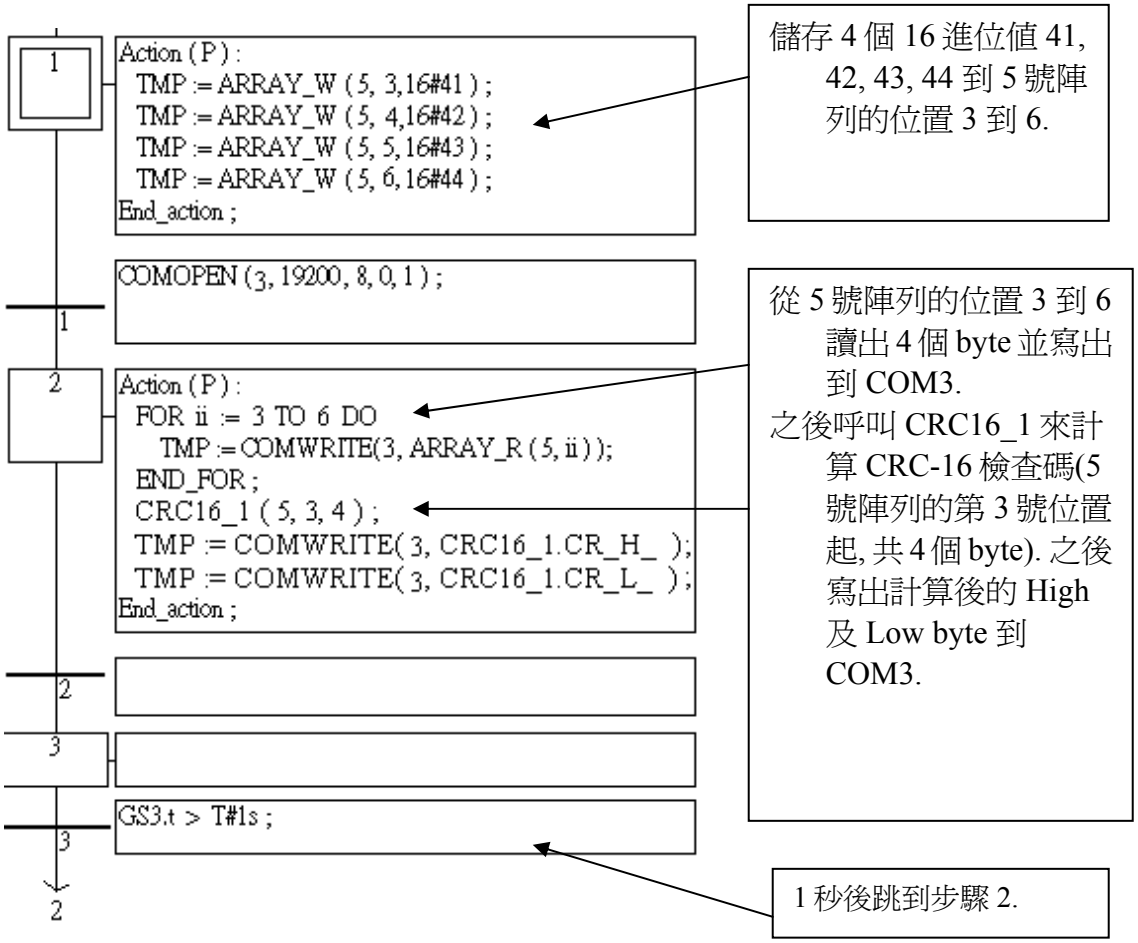
NUM_	Integer	要計算的 byte 陣列編號, 範圍爲 I-8xx7 & I-7188EG/XG: 1 到 24; Wincon-8xx7: 1 到 48;
ADR_	Integer	從陣列內的那個位置起開始計算(1~256)
SIZE_	Integer	共有幾個 byte 要計算

傳回值 :

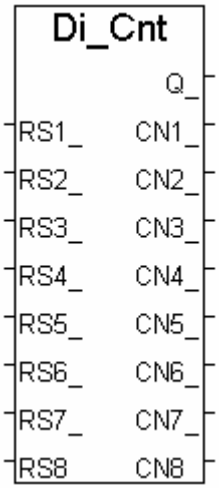
CR_H_	Integer	計算結果, high byte
CR_L_	Integer	計算結果, low byte

範例:

TMP 宣告爲 internal boolean. ii, CR_H_ 及 CR_L_ 爲 internal integer, CRC16_1 宣告爲 FB instance 其型態爲 CRC_16.



DI_CNT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

讀取 Slot 0 的 DI Counter 值. Wincon 則爲 Slot 1, 請參閱第 3.8 節

輸入參數:

RS1_ ~ RS8_	Boolean	當由 FALSE 上升到 TRUE 時重置該點的 counter 值爲 0.

傳回值:

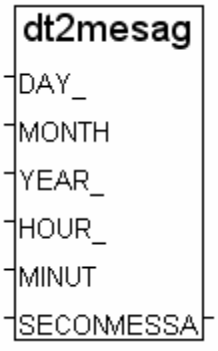
Q_	Boolean	正確: TRUE. 如果 Q_ 傳回 FALSE, 表示 "slot 0 找不到並列輸入點 "
CN1_ ~ CN8_	Integer	第 1 到 8 點的 counter 值. 範圍介於 0 到 2,147,483,647. 假如值超過 2,147,483,647, 會從 0 開始.

注意 :

1. 只有插在 slot 0 的並列輸入(parallel D/I)卡可使用 "Di_Cnt", 其它 slot 不行.
2. WinCon W-8xx7 必需插在 slot 1, 其它 slot 不行.
3. 只有前 8 個 D/I 點可使用 "Di_Cnt".
4. I-7188EG/XG 必須在 slot 0 連結 Xxxx 卡, "Di_Cnt" 才能使用.
5. 每台 I-8xx7 & I-7188EG/XG 控制器最多只能使用 8 個並列 D/I counter. Counter 的輸入頻率最高爲 500Hz. 最小脈波長度需大於 1 ms.
6. 每台 W-8xx7 控制器最多只能使用 8 個並列 D/I counter. Counter 的輸入頻率最高爲 250Hz. 最小脈波長度需大於 2 ms.

範例: W-8xx7: Wdemo_22 , I-8xx7: demo_63

DT2MESAG



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 Date 和 Time 成爲 Message 資料型態

輸入參數 :

DAY_ :	Integer	日期 (1~31)
MONTH_	Integer	月份 (1~12)
YEAR_ :	Integer	年, 例如. 01,02
HOUR_ :	Integer	時 (0~23)
MINUTE_ :	Integer	分 (0~59)
SECOND_ :	Integer	秒 (0~59)

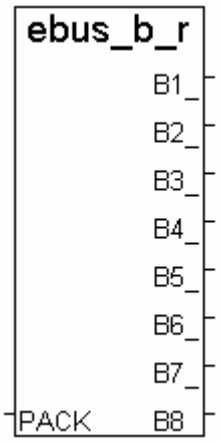
傳回值 :

MESSAGE_:	Message	"日.月.年 時:分:秒", 例如: "20.01.07 11:05:40"

注意 :

若輸入的參數不正確, 傳回的 Message 爲 '' (空訊息 NULL).

EBUS_B_R



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Ebus 上讀取 boolean 封包

輸入參數:

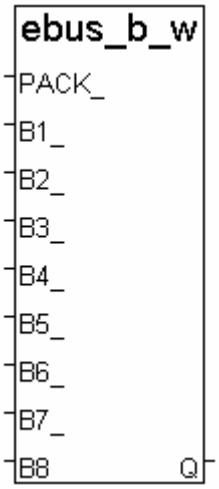
PACK_	Integer	要讀哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256

傳回值 :

B1_ ~ B8_	Boolean	讀到該封包內的 8 個 boolean 值

注意: 請參考第 7.5 節

EBUS_B_W



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 boolean 封包到 Ebus 上

輸入參數:

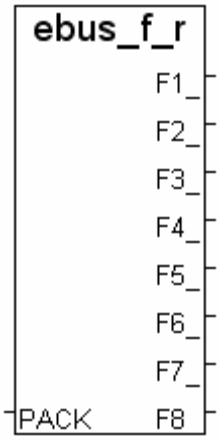
PACK_	Integer	寫出哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256
B1_ ~ B8_	Boolean	要寫出的 8 個 boolean 值

傳回值 :

Q	Boolean	永遠回傳 TRUE.

注意: 請參考第 7.5 節

EBUS_F_R



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Ebus 上讀取 Real 封包

輸入參數:

PACK_NO_	Integer	要讀哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256

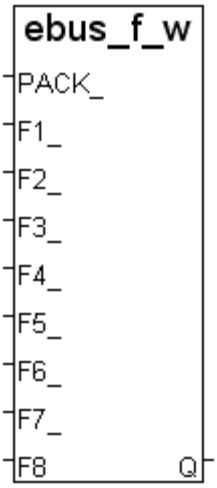
傳回值 :

F1_ ~ F8_	Real	讀到該封包內的 8 個 Real 值 如有錯誤發生, 傳回 1.23E-20

注意:

1. "EBUS_F_R" 與 "EBUS_N_R" 使用相同記憶體
2. Integer 和 REAL 不要使用同一個封包號碼. 否則資料會錯誤, 有時並會產生錯誤: "ERROR 115: EBUS_F_R float error"

EBUS_F_W



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 Real 封包到 Ebus 上

輸入參數:

PACK_	Integer	寫出哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256
N1_ ~ N8_	Boolean	要寫出的 8 個 Real 值

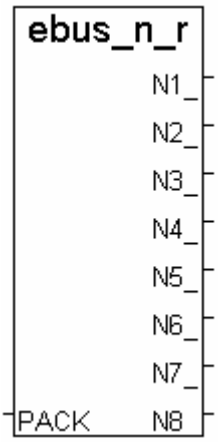
傳回值 :

Q	Boolean	永遠回傳 TRUE

注意:

1. "EBUS_F_W" 與 "EBUS_N_W" 使用相同記憶體區
2. Integer 和 REAL 不要使用同一個封包號碼. 否則資料會錯誤, 有時並會產生錯誤: "ERROR 115: EBUS_F_R float error"
3. 從以下版本起的驅動程式才有支援 EBUS_F_R 與 EBUS_F_W:
   - I-7188EG: 2.17 版起
   - I-7188XG: 2.15 版起
   - I-8XX7 : 3.19 版起
   - W-8XX7 : 建議更新至 3.36 版或更高版

EBUS_N_R



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Ebus 上讀取 integer 封包

輸入參數:

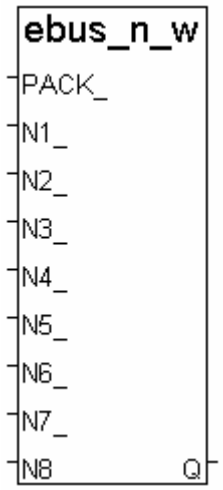
PACK_	Integer	要讀哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256

傳回值 :

N1_ ~ N8_	Integer	讀到該封包內的 8 個 integer 值

注意: 請參考第 7.5 節

EBUS_N_W



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 integer 封包到 Ebus 上

輸入參數:

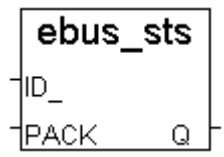
PACK_	Integer	寫出哪個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256
N1_ ~ N8_	Boolean	要寫出的 8 個 integer 值

傳回值 :

Q	Boolean	永遠回傳 TRUE

注意: 請參考第 7.5 節

EBUS_STS



I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

取得 Ebus 封包的傳送狀態

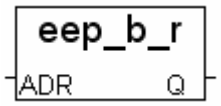
輸入參數:

ID_	Integer	取得什麼? (0 ~ 1) , 0: Boolean 封包 , 1: Integer 封包
PACK_	Integer	取得那個編號的封包, I-8xx7, I-7188EG/XG : 1 ~ 128 W-8xx7 : 1~256

傳回值 :

Q_	Boolean	TRUE: 該封包通訊正常, FALSE: 該封包通訊異常 封包通訊異常的原因可能是, Ebus_m 控制器沒有啓動該封包編號, 通訊線中斷, 發送該封包的工控器死機了, Ebus_m 控制器死機了, 或其它

EEP_B_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 EEPROM 讀出 1 個 boolean

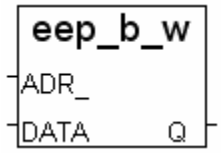
輸入參數 :

ADR_	Integer	讀哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 256) , W-8xx7: (1 ~ 1024)

傳回值 :

Q_	Boolean	讀到的 boolean 值

EEP_B_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個 boolean 值到 EEPROM

輸入參數 :

ADRES_	Integer	寫到哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 256); W-8xx7: (1 ~ 1024).
DATA_	Boolean	要寫的 boolean 值

傳回值 :

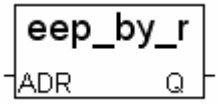
Q_	Boolean	正確回傳 TRUE.

注意:

* 讀 EEPROM 沒有次數限制
* 寫 EEPROM 有次數限制 (請參考 10.2 節)
* 需用 EEP_EN 開啓 EEPROM, 才寫的進去
* 讀/寫 EEPROM 很耗 CPU 時間, 會造成 Scan Time 時間大幅增加, 請小心使用.

範例: 請參考 demo_17

EEP_BY_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 EEPROM 讀出 1 個 byte (unsigned 8-bit)

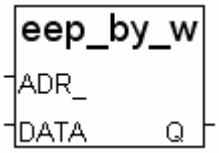
輸入參數 :

ADR_	Integer	讀哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 1512) , W-8xx7: (1 ~ 14272)

傳回值 :

Q_	Integer	讀到的 byte 值 (0~255)

EEP_BY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個 byte (unsigned 8-bit) 值到 EEPROM

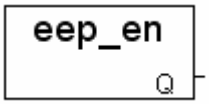
輸入參數 :

ADR_	Integer	寫到哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 1512) , W-8xx7: (1 ~ 14272)
DATA_	Integer	要寫的 byte 值 (0 ~ 255)

傳回值 :

Q_	Boolean	正確回傳 TRUE.

EEP_EN



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

開啓 EEPROM 以便可以寫入

傳回值 :

Q_	Boolean	成功回傳 TRUE.

注意 :

* EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用. 下面的位址編號使用相同的記憶体位址。

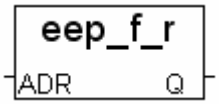
Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, … *)
Word	2n-1, 2n	
Integer, Real	n	

* 寫 EEPROM 有次數限制 (請參考 10.2 節)
* 需用 EEP_EN 開啓 EEPROM, 才寫的進去
* 讀/寫 EEPROM 很耗 CPU 時間, 會造成 Scan Time 時間大幅增加, 請小心使用.

範例: 請參考第 11 章 demo_17

EEP_F_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

從 EEPROM 讀取 1 個 Real 值

輸入參數 :

ADR_	Integer	讀哪個位置 (與 EEP_N_R 使用相同 EEPROM 位址) I-8xx7 & I-7188EG/XG: (1 ~ 378) , W-8xx7: (1 ~ 3568)

傳回值 :

Q_	Real	讀到的 Real 值. 若 ADR_ 超出有效範圍, 則 Q_ = 1.23E-20. 若儲存在 EEPROM 的不是 REAL, 則 Q_ 的值會是錯的, 有時並會發生錯誤: "ERROR 114 : EEP_F_R float error".

重要 :

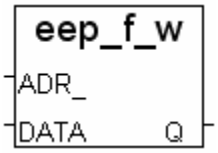
1. "EEP_F_R" 與 "EEP_N_R" 使用相同的 EEPROM 位址
2. Integer 和 REAL 不要讀/寫同一個 EEPROM 位址. 否則有可能會產生錯誤: "ERROR 114: EEP_F_R float error".
3. 每次讀/寫 EEPROM 皆耗費許多 CPU 時間, 尤其是寫的動作, 請小心使用.

注意 :

1. 這個函式即使沒有呼叫 EEP_EN 也可以使用.
2. EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用.
   例如: EEP_N_R 的 ADR_ 2 使用 4 bytes, 和 EEP_WD_R 的 ADR_ 3, 4 及 EEP_BY_R 的 ADR_ 5, 6, 7, 8 使用相同的記憶區.
3. W-8xx7/8xx6 使用 EEPROM 的 16 - 31 區來儲存 Boolean (每區有 64 bytes), 第 32 - 254 區用來儲存 byte, word 和 long. 第 0 – 15 則沒有使用. 第 255 區保留.
4. 從以下版本起的驅動程式才有支援 EEP_F_R 與 EEP_F_W:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

EEP_F_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

寫 1 個 Real 值到 EEPROM

輸入參數 :

ADR_	Integer	寫到哪個位置 (與 EEP_N_W 使用相同 EEPROM 位址) I-8xx7 & I-7188EG/XG: (1 ~ 378) , W-8xx7: (1 ~ 3568)
DATA_ :	Real	要寫入的 REAL 值

傳回值 :

Q_	Boolean	TRUE : 成功, FALSE: 失敗.

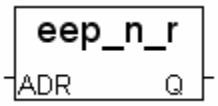
重要 :

1. "EEP_F_W" 與 "EEP_N_W" 使用相同的 EEPROM 位址
2. Integer 和 REAL 不要讀/寫同一個 EEPROM 位址. 否則有可能會產生錯誤: "ERROR 114: EEP_F_R float error".
3. 每次讀/寫 EEPROM 皆耗費許多 CPU 時間, 尤其是寫的動作, 請小心使用.

注意 :

1. 需使用 EEP_EN 開啓 EEPROM, 才寫的進去.
2. EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用.
   例如: EEP_N_R 的 ADR_ 2 使用 4 bytes, 和 EEP_WD_R 的 ADR_ 3, 4 及 EEP_BY_R 的 ADR_ 5, 6, 7, 8 使用相同的記憶區.
3. 小心使用此函式! 當寫入超過 100,000 次, EEPROM 將會損毀.
4. 寫入之前須先呼叫 EEP_EN( ) 來解除 EEPROM 的寫入保護, 如此之後才能寫入.
5. 寫入之後, 需呼叫 EEP_PR( ) 來保護 EEPROM
6. W-8xx7/8xx6 使用 EEPROM 的 16 - 31 區來儲存 Boolean (每區有 64 bytes), 第 32 - 254 區用來儲存 byte, word 和 long. 第 0 - 15 則沒有使用. 第 255 區保留.
7. 從以下版本起的驅動程式才有支援 EEP_F_R 與 EEP_F_W:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

EEP_N_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 EEPROM 讀出 1 個 長整數 (signed 32-bit)

輸入參數 :

ADR_	Integer	讀哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 378) , W-8xx7: (1 ~ 3568)

傳回值 :

Q_	Integer	讀到的長整數值

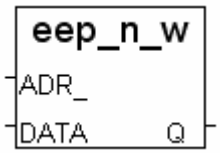
* EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用. 下面的位址編號使用相同的記憶体位址。

Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, … *)
Word	2n-1, 2n	
Integer, Real	n	

* 讀 EEPROM 沒有次數限制, 不管有無使用 EEP_EN 開啓 EEPROM, 都可讀
* 寫 EEPROM 有次數限制 (請參考 10.2 節)

範例: 請參考第 11 章 demo_17

EEP_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個 長整數(signed 32-bit) 值到 EEPROM

輸入參數 :

ADR_	Integer	寫到哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 378) , W-8xx7: (1 ~ 3568)
DATA_	Integer	要寫的長整數值

傳回值 :

Q_	Boolean	正確回傳 TRUE.

* EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用. 下面的位址編號使用相同的記憶体位址。

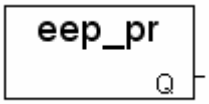
Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, … *)
Word	2n-1, 2n	
Integer, Real	n	

* 寫 EEPROM 有次數限制 (請參考 10.2 節)
* 需用 EEP_EN 開啓 EEPROM, 才寫的進去
* 讀/寫 EEPROM 很耗 CPU 時間, 會造成 Scan Time 時間大幅增加, 請小心使用.

範例: 請參考第 11 章 demo_17

EEP_PR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

關閉 EEPROM 以防止寫入

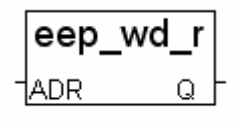
傳回值 :

Q_	Boolean	成功回傳 TRUE

* 寫 EEPROM 有次數限制 (請參考 10.2 節)
* 需用 EEP_EN 開啓 EEPROM, 才寫的進去

範例: 請參考第 11 章 demo_17

EEP_WD_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 EEPROM 讀出 1 個 word (signed 16-bit)

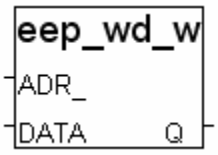
輸入參數 :

ADR_	Integer	讀哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 756) , W-8xx7: (1 ~ 7136)

傳回值 :

Q_	Integer	讀到的 word 值 (-32768 ~ +32767)

EEP_WD_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫 1 個 word (signed 16-bit) 值到 EEPROM

輸入參數 :

ADR_	Integer	寫到哪個位置 I-8xx7 & I-7188EG/XG: (1 ~ 756) , W-8xx7: (1 ~ 7136)
DATA_	Integer	要寫的 word 值 (-32768 ~ 32767)

傳回值 :

Q_	Boolean	正確回傳 TRUE.

* EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R 與 EEP_F_W 等 functions 使用相同的記憶區, 請小心配置使用. 下面的位址編號使用相同的記憶体位址。

Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, … *)
Word	2n-1, 2n	
Integer	n	

* 寫 EEPROM 有次數限制 (請參考 10.2 節)
* 需用 EEP_EN 開啓 EEPROM, 才寫的進去
* 讀/寫 EEPROM 很耗 CPU 時間, 會造成 Scan Time 時間大幅增加, 請小心使用.

範例: 請參考第 11 章 demo_17

F_APPEND

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

將 1 個檔案之內容 附加到 1 個檔案 的後方

輸入參數 :

SRC_	Message	來源檔名. 例如 '\CompactFlash\data.txt'
DES_	Message	目地檔名. 例如 '\CompactFlash\data1.txt'

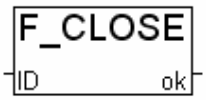
傳回值 :

Q_	Boolean	True: Ok, False: 失敗

注意:

1. 如果有任一個檔案不存在, 回傳 False.
2. 來源檔 與 目地檔的狀態 必需是 Close 的. 未 Open
3. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
4. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
5. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

F_CLOSE



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

關閉以 F_ROPEN, F_WOPEN 或 F_CREAT 開啓的二進位檔案

ISaGRAF 的 simulator 不包含此函式, 無法模擬(仿真)

輸入參數 :

ID	Integer	F_ROPEN, F_WOPEN 或 F_CREAT 傳回的檔案編號: .

傳回值 :

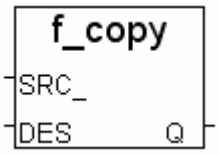
ok	Boolean	狀態回傳 TRUE :檔案關閉 ok; FALSE : 失敗

範例 :

```
 (* ST 程式: *)
file_id := F_ROPEN('\CompactFlaesh\data.bin');
ok := F_CLOSE(file_id);

(* IL 相等式: *)
LD          '\CompactFlaesh\data.bin'
F_ROPEN
ST          file_id
F_CLOSE                          (* file_id 已經在 IL 程式結果中 *)
ST          ok
```

F_COPY



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

複製 1 個檔案

輸入參數 :

SRC_	Message	來源檔名. 例如 '\CompactFlash\data.txt'
DES_	Message	目地檔名. 例如 '\CompactFlash\data1.txt'

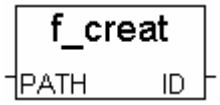
傳回值 :

Q_	Boolean	True: Ok, False: 失敗

注意:

1. 複製 來源檔 到目地檔.
2. 來源檔 與 目地檔 的狀態 必需是 Close 的. 未 Open
3. 如果目地檔已經存在, 會被整個替換掉.

F_CREAT



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

建立 1 個新的 空檔案 以便日後 讀 或 寫

輸入參數 :

Path_	Message	檔名. 例如 '\CompactFlash\data.txt'

傳回值 :

ID_	Integer	檔案代碼, 若爲 0, 表示 建立檔案 失敗

注意:

1. 如果檔案已經存在, 呼叫此函數會 清空 原先檔案內的資料.
2. 若要 讀取 已經存在的檔案, 請用 ISaGRAF 的標準函數 – "F_ROPEN( )"
3. 若要 寫入 已經存在的檔案, 請用 ISaGRAF 的標準函數 – "F_WOPEN( )"
4. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
5. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
6. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

範例: Wincon CD-ROM :\napdos\isagraf\wincon\demo\ 內的 Wdemo_11 & Wdemo_12

F_DELETE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

刪除 1 個檔案

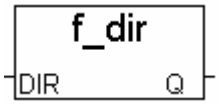
輸入參數 :

Name_	Message	檔名. 例如 '\CompactFlash\data.txt'

傳回值 :

Q_	Boolean	True: Ok, False: 失敗

F_DIR



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

建立一個新的目錄 (Directory)

輸入參數 :

Dir_	Message	Directory 的名稱. 例如 **'\DATA21'**

傳回值 :

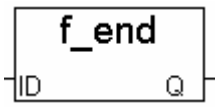
Q_	Boolean	True: Ok, False: 失敗 (例如 Directory 已存在)

範例:

```
(* INIT 宣告爲 Internal Boolean, 初值爲 True*)
(* TMP 宣告爲 Internal Boolean*)

if INIT then
    INIT := False ;
    TMP := f_dir('\DATA21') ;
End_if ;
```

F_END



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

移動檔案目前位置 到檔案的結尾

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)

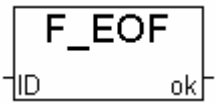
傳回值 :

Q_	Boolean	True: Ok, False: 失敗

注意:

1. 請參考 F_seek 來移動檔案目前位置 到一個指定的位置
2. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
3. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
4. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

F_EOF



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

測試是否 檔案目前位置 已抵達 檔案的結尾

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)

傳回值 :

Ok	Boolean	True: 已抵達 檔案的結尾 ， False: 還未抵達

F_READ_B

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從檔案內的目前位置讀出 1 個 byte 值

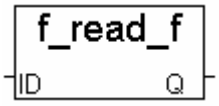
輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)

傳回值 :

Q_	Integer	讀到的 byte 值 (0 ~ 255)

F_READ_F



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從檔案內的目前位置讀出 1 個 實數值 (32-bit float)

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)

傳回值 :

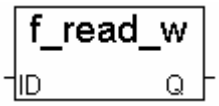
Q_	Real	讀到的 實數值 (32-bit float), 若該位置存的不是實數值, 會傳回錯誤的值, 有時還會發生 “ERROR 117:F_READ_ error”.

注意:

1. 請使用 ISaGRAF 標準函式 –“FA_READ” & “FA_WRITE” 來 讀/寫 長整數 (signed 32-bit)
2. 請使用 ISaGRAF 標準函式 –“FM_READ” & “FM_WRITE” 或 “F_writ_s” 來 讀/寫 字串 (string)
3. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
4. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
5. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

範例: 請參考 Wincon CD:\napdos\isagraf\wincon\demo\ “wdemo_01” & “wdemo_02”

F_READ_W



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從檔案內的目前位置讀出 1 個 Word 值 (signed 16-bit)

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)

傳回值 :

Q_	Integer	讀到的 Word 值 (-32768 ~ +32767)

F_ROPEN

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

以 Read 模式開啓一個二進位檔案. 之後該檔案只能做讀的動作.

輸入參數 :

Path_	Message	檔案名稱 可包含執行路徑, 使用 \ 或 / (兩者相同) 符號指定路徑.

傳回值 :

ID_	Integer	檔案號碼 0 : 有錯誤, 檔案不存在.

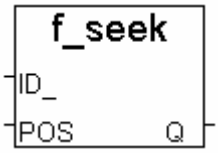
範例 :

```
 (* ST 程式: *)
file_id := F_ROPEN(' c:\CompactFlash\ISaGRAF\data.bin ');
error := (file_id=0);

(* IL 相等式: *)
LD          'c:\CompactFlash\ISaGRAF\data.bin'
F_ROPEN
ST          file_id
EQ          0
ST          error
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

F_SEEK



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

移動檔案內的 目前位置 到 指定位置…

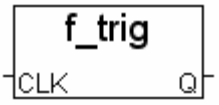
輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)
POS_	Integer	移到那個位置, 單位爲 byte (1 ~ …)

傳回值 :

Q_	Boolean	True: 成功. False: 失敗

F_TRIG



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

偵測 boolean 變數的 falling edge (下降邊緣)…

輸入參數 :

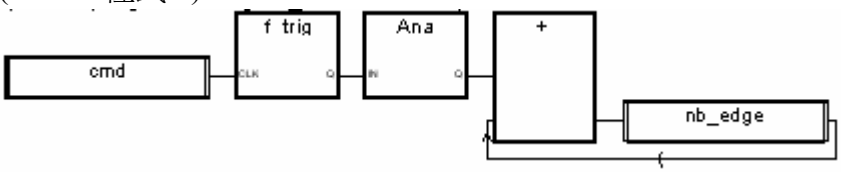
CLK_	Boolean	任何布林變數

傳回值 :

Q_	Boolean	TRUE : 若 CLK 由 TRUE 變爲 FALSE FALSE : 其他狀態

範例 :

(* FBD 程式 *)

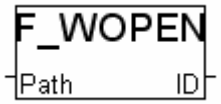



```
(* ST 相等式: 假設 F_TRIG1 爲 F_TRIG 函式方塊 *)
F_TRIG1(cmd);
nb_edge := ANA(F_TRIG1.Q) + nb_edge;
```

```
(* IL 相等式: *)
LD      cmd
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

F_WOPEN

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

以 Write 模式開啓一個二進位檔案. 之後該檔案可作讀/寫動作.
ISaGRAF 的 simulator 不包含此函式

輸入參數 :

Path	Message	檔案名稱 可包含執行路徑, 使用 \ 或 / (兩者相同) 符號指定路徑.

傳回值 :

ID	Integer	檔案號碼 0 : 表示有錯誤. 若檔案已存在, 將被覆蓋.

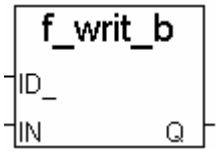
範例 :

```
 (* ST 程式: *)
file_id := F_WOPEN('\CompactFlash\hello.dat');
error := (file_id=0);


(* IL 相等式: *)
LD          '\CompactFlash\hello.dat'
F_WOPEN
ST          file_id
EQ          0
ST          error
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

F_WRIT_B



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫入 1 個 Byte 值到檔案內…

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 那個回傳值)
IN_	Integer	要寫入的 Byte 值 (0 ~ 255), 如果 > 255 或 < 0, 最低的 byte 值被寫入

傳回值 :

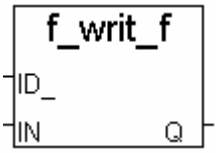
Q_	Boolean	True: 成功. False: 失敗

注意:

1. 請使用 ISaGRAF 標準函式 –"FA_READ" & "FA_WRITE" 來 讀/寫 長整數 (signed 32-bit)
2. 請使用 ISaGRAF 標準函式 –"FM_READ" & "FM_WRITE" 或 "F_writ_s" 來 讀/寫 字串 (string)
3. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
4. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
5. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

範例: 請參考 Wincon CD:\napdos\isagraf\wincon\demo\ "wdemo_01" & "wdemo_02"

F_WRIT_F



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫入 1 個 實數值 (32-bit float) 到檔案內…

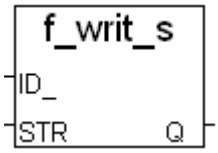
輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)
IN_	Real	要寫入的 實數值

傳回值 :

Q_	Boolean	True: 成功. False: 失敗

F_WRIT_S



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫入 1 個 字串 到檔案內 (字串結尾不含 <CR> <LF>)

輸入參數 :

ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)
STR_	Message	要寫入的字串

傳回值 :

Q_	Boolean	True: 成功. False: 失敗

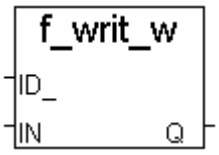
注意:

1. F_writ_s 不會在 字串結尾加入 <CR> <LF>, FM_write 則會在 字串結尾加入 <CR> <LF>
2. 請使用 ISaGRAF 標準函式 –"FA_READ" & "FA_WRITE" 來 讀/寫 長整數 (signed 32-bit)
3. 請使用 ISaGRAF 標準函式 –"FM_READ" & "FM_WRITE" 或 "F_writ_s" 來 讀/寫 字串 (string)
4. 可參考其它 ISaGRAF 的標準函數 – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
5. 可參考 ICP DAS 加入的函數 – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
6. 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

範例:

請參考 Wincon CD:\napdos\isagraf\wincon\demo\ "wdemo_01" & "wdemo_02"

F_WRIT_W



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫入 1 個 Word (signed 16-bit) 到檔案內…

輸入參數 :

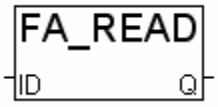
ID_	Integer	檔案代碼, (使用 F_ROPEN , F_WOPEN 或 F_CREAT 的回傳值)
IN_	Integer	要寫入的 Word 值 (-32768 ~ +32767)

傳回值 :

Q_	Boolean	True: 成功. False: 失敗

FA_READ

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

讀取檔案內的一個長整數 (32-bit signed)
ISaGRAF 的 simulator 不包含此函式

輸入參數 :

ID	Integer	F_ROPEN, F_WOPEN 或 F_CREAT 傳回的檔案編號.

傳回值 :

Q	Integer	由檔案讀出的整數值

範例 :

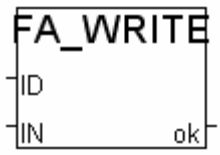
```
 (* ST 程式: *)
file_id := F_ROPEN('\CompactFlash\data.dat');
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);

(* IL 相等式: *)
LD          '\CompactFlash\data.dat'
F_ROPEN
ST          file_id
LD          file_id
FA_READ                     (* 讀取 vinc *)
ST          vinc
LD          file_id
FA_READ                     (* 讀取 timer: delta_tim *)
TMR                         (* 轉換格式爲 timer *)
ST          delta_tim
LD          file_id
F_CLOSE
ST          ok
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

FA_WRITE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

寫一個長整數(32-bit signed) 到檔案內
ISaGRAF 的 simulator 不包含此函式

輸入參數 :

ID	Integer	F_WOPEN, F_WOPEN 或 F_CREAT 傳回的檔案編號.
IN	Integer	要寫入檔案的整數值

傳回值 :

OK	Boolean	執行狀態: TRUE 表示 ok

範例 :

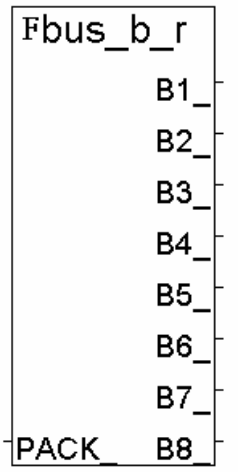
```
 (* ST 程式: *)
file_id := F_WOPEN('\CompactFlash\data.dat');
nb_written  := 0;
nb_written := nb_written + ana(FA_WRITE(file_id,vstart));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
                ERROR := ERR_FILE;
END_IF;

(* IL 相等式: 請參考 ISaGRAF Projects 的 Help/ language Reference*)
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

FBUS_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

從 Fbus 上讀取 boolean 封包

輸入參數 :

PACK_	Integer	封包編號, 1 ~ 128

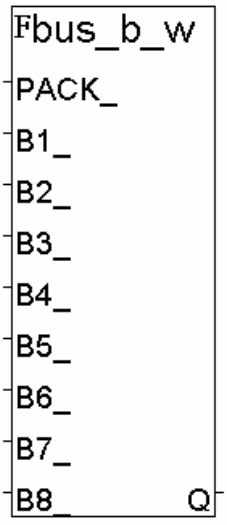
傳回值 :

B1_ ~ B8_	Boolean	讀到的封包內的 8 個 boolean 值

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

寫出 Boolean 封包到 Fbus 上

輸入參數 :

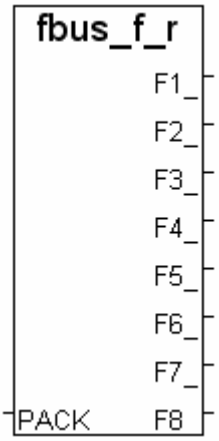
PACK_	Integer	封包編號, 1 ~ 128
B1_ ~ B8_	Boolean	要寫出的 8 個 boolean 值

傳回值 :

Q_	Boolean	只回傳 TRUE.

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_F_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function Block

從 Fbus 上讀取 Real 封包

輸入參數 :

PACK_NO_	Integer	封包編號, 1 ~ 128

傳回值 :

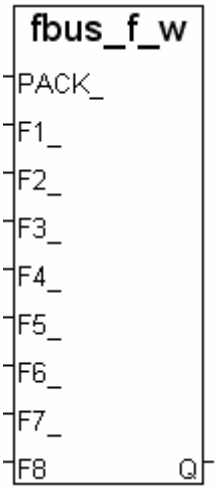
F1_ ~ F8_	Real	讀到的封包內的 8 個 Real 值 錯誤則傳回 1.23E-20

重要 :

1. "FBUS_F_R" 與"FBUS_N_R" 使用相同的記憶體區.
2. Integer 和 Real 資料型態請不要使用同一個封包號碼. 否則資料會錯誤, 有時並會產生:
   "ERROR 116 : FBUS_F_R float error"
3. 請使用 "FBUS_N_R" 和 "FBUS_N_W" 來傳送 Integer 值

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_F_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function Block

寫出 Real 封包到 Fbus 上

輸入參數 :

PACK_NO_	Integer	封包編號, 1 ~ 128
F1_ ~ F8_	Real	要寫出的 8 個 Real 值

傳回值 :

Q_	Boolean	只回傳 TRUE.

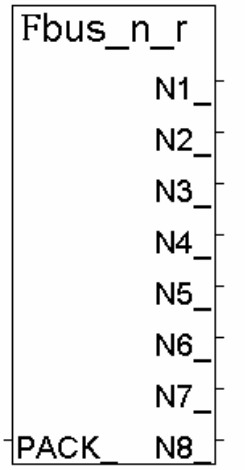
重要 :

1. "FBUS_F_W" 與"FBUS_N_W" 使用相同的記憶體區.
2. Integer 和 Real 資料型態請不要使用同一個封包號碼. 否則資料會錯誤, 有時並會產生:
   "ERROR 116 : FBUS_F_R float error"
3. 請使用 "FBUS_N_R" 和 "FBUS_N_W" 來傳送 Integer 值
4. 從以下版本起的驅動程式才有支援 FBUS_F_R 與 FBUS_F_W:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

從 Fbus 上讀取 Integer 封包

輸入參數 :

PACK_	Integer	封包編號, 1 ~ 128

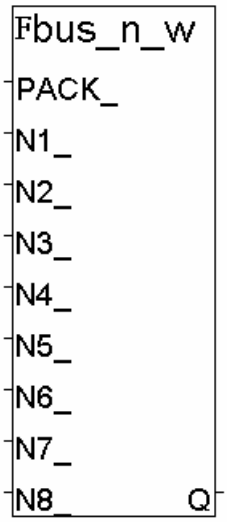
傳回值 :

N1_ ~ N8_	Integer	讀到的封包內的 8 個 integer 值

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

寫出 Integer 封包到 Fbus 上

輸入參數 :

PACK_	Integer	封包編號, 1 ~ 128
N1_ ~ N8_	Boolean	要寫出的 8 個 Boolean 值

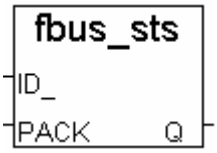
傳回值 :

Q_	Boolean	只回傳 TRUE.

範例: 請參考第 7 章或 demo_11a & demo_11b

FBUS_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function Block

取得 Fbus 封包的通訊狀態

輸入參數 :

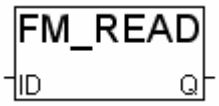
ID_	Integer	取得什麼? (0 ~ 1) , 0: Boolean 封包 , 1: Integer 封包
PACK_	Integer	取得那個編號的封包. 1 ~ 128

傳回值 :

Q_	Boolean	TRUE: 該封包通訊正常, FALSE: 該封包通訊異常 封包通訊異常的原因可能是, Fbus_m 控制器沒有啓動該封包編號,通訊線中斷, 發送該封包的控制器死機了, Fbus_m 控制器死機了, 或其它 …

FM_READ

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

從二進位檔讀取 MESSAGE 變數.
ISaGRAF 的 simulator 不包含此函式

輸入參數 :

ID	Integer	F_ROPEN, F_WOPEN 或 F_CREAT 傳回的檔案編號.

傳回值 :

Q	Message	讀出的 message 值

範例 :

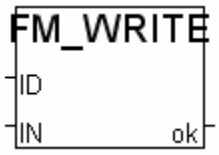
```
(* ST 程式: *)
file_id := F_ROPEN('\CompactFlash\m1.txt');
status1 := FM_READ(file_id);
ok := F_CLOSE(file_id);


(* IL 相等式: *)
LD          '\CompactFlash\m1.txt'
F_ROPEN
ST          file_id
FM_READ                         (* 讀取 status1 *)
ST          status1
LD          file_id
F_CLOSE
ST          ok
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

FM_WRITE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

將 MESSAGE 寫入二進位檔.

ISaGRAF 的 simulator 不包含此函式

輸入參數 :

ID	Integer	F_WOPEN 傳回的檔案號碼.
IN	Message	要寫入檔案的訊息

傳回值 :

OK	Boolean	執行狀態 : TRUE 表示成功

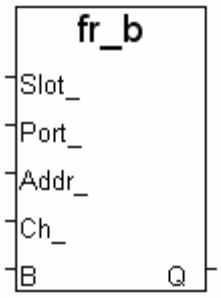
範例 :

```
(* ST 程式: *)
file_id := F_WOPEN('\CompactFlash\m1.txt ');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);

(* IL 相等式: *)
LD          'trace.txt'
F_WOPEN
ST          file_id
FM_WRITE    'First message'     (*寫第一個訊息 *)
ST          ok
LD          file_id
FM_WRITE    'Last message'      (*寫第二個訊息*)
ST          ok
LD          file_id
F_CLOSE
ST          ok
```

注意: 在 WinCon 的 \CompactFlash\ 路徑內操作 File 會消耗很多 CPU 時間, 若是在 RAM Disk 內則不會, 比如 \Temp\ 內, 但關機後, File 就會消失.

FR_B



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

設定 ISaGRAF Boolean 變數對應到一個 FRNET I/O 點

輸入參數 :

Slot_	:	Integer	所插入的 I-8172 的插槽編號 (1 - 7)
Port_	:	Integer	使用那個 I-8072 的埠號 (0 或 1)
Addr_ :		Integer	模組位址, D/O (0 - 7) , D/I (8 - 15)
Ch_ :		Integer	使用的 Channel 編號 (1 - 16)
B_ :		Boolean	boolean 變數名稱

傳回值 :

Q_ :		Boolean	True: Ok. False: 參數錯誤.

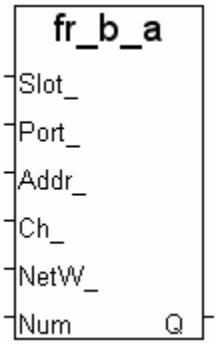
注意 :

1. I-8172 請連接到 Wincon 的 slot 1 到 7 之中, 然後再以其 Port0 或 Port1 連接 FRNET I/O 模組
2. Fr_B , Fr_B_A 必須在第一次 PLC scan 時呼叫. 第二次以後的 PLC scan 則無法呼叫成功.
3. FRNET D/O 模組不支援通訊狀態偵測, 而 FRNET D/I 模組則支援通訊狀態偵測.
4. 每個 FRNET 輸出模組的 Dip switch 上有一個 'RESET' dip 或特殊 Jumper. 需先將 'RESET' dip 或 Jumper 設到 'ON' 的位置 (或 enable), 如此一來, 當 I-8172 與 FRNET D/O 模組通訊中斷時,便可重置輸出 channel 爲 OFF 狀態.
   例如 : 將 FR-2057 的第 8 個 Dip 切到 'ON' 則可重新啓動.
5. 請參考下列網址 取得更多資訊:
   http://www.icpdas.com/faq/isagraf.htm 'FAQ048' 及
   http://www.icpdas.com/products/Remote_IO/frnet/frnet_list.htm

範例 : 請參考網址 http://www.icpdas.com/faq/isagraf.htm 的 'FAQ048', 範例檔Wdemo_39

Fr_B_A

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定 ISaGRAF Boolean 變數陣列對應到數個 FRNET I/O 點
(請參考 2.6 節關於"變數陣列"的說明)

輸入參數 :

Slot_ :	Integer	所插入的 I-8172 的插槽編號 (1 - 7)
Port_ :	Integer	使用那個 I-8072 的埠號 (0 或 1)
Addr_ :	Integer	起始模組位址, D/O (0 - 7) , D/I (8 - 15)
Ch_ :	Integer	起始 Channel 編號 (1 - 16)
NetW_ :	Integer	對應 "變數陣列" 的網路位址編號 : 1~ 8191
Num_ :	Integer	使用 FRNet I/O 時, boolean "變數陣列" 的數量 : 1 - 255.
		例如: Bi[0..15] 的大小爲 16, 可設定 NUM_ 爲 1 到 16.
		ABC[0..127] 的大小爲 128, 可設定 NUM_ 爲 1 到 128

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

注意 :

1. I-8172 請連接到 Wincon 的 slot 1 到 7 之中, 然後再以其 Port0 或 Port1 連接 FRNET I/O 模組
2. Fr_B , Fr_B_A 必須在第一次 PLC scan 時呼叫. 第二次以後的 PLC scan 則無法呼叫成功.
3. FRNET D/O 模組不支援通訊狀態偵測, 而 FRNET D/I 模組則支援通訊狀態偵測.
4. 每個 FRNET 輸出模組的 Dip switch 上有一個 'RESET' dip 或特殊 Jumper. 需先將 'RESET' dip 或 Jumper 設到 'ON' 的位置 (或 enable), 如此一來, 當 I-8172 與 FRNET D/O 模組通訊中斷時,便可重置輸出 channel 爲 OFF 狀態.
   例如 : 將 FR-2057 的第 8 個 Dip 切到 'ON' 則可重新啓動.
5. 宣告 ISaGRAF 版本 3.4 (或 3.5) 的 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下的 "isa.ini" 檔案頂端加入 2 行. 加入後, 開啓 ISaGRAF 工作平台, 在 Dictionary 宣告視窗內 增加的 "DIM" 欄位中設定.
   *請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   ```
   [DEBUG]
   arrays=1
   ```
6. 請參考下列網址 取得更多資訊:
   http://www.icpdas.com/faq/isagraf.htm 'FAQ048' 及
   http://www.icpdas.com/products/Remote_IO/frnet/frnet_list.htm

範例 : 請參考網址 http://www.icpdas.com/faq/isagraf.htm 的 'FAQ048', 範例檔Wdemo_39

GET_INFO

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

取得控制器資訊

輸入參數 :

TYPE_ : Integer 取得資訊種類

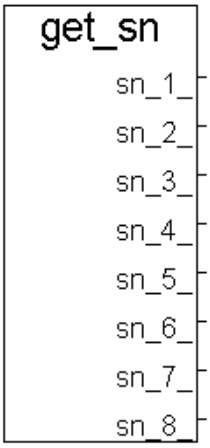
傳回值 :

DATA_ : Integer 傳回資訊的值.

"TYPE_" V.S. "DATA_" 對照表

TYPE_		DATA_
1	取得控制器的 ID (slave No.)	1 - 255
2	保留給未來設定	0
3	保留給未來設定	0
4	保留給未來設定	0

GET_SN



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

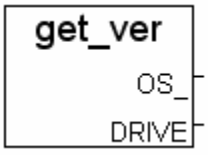
型態 : C_Function Block

取得 硬體唯一的 serial No. 共 8 個整數

傳回值 :

SN_1_ ~ SN_8_ : Integer 硬體唯一的 serial No

GET_VER



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

取得硬體驅動程式版本

(適用 I-8xx7: v2.19 , I-7188EG: v1.10 , I-7188XG: v1.08 或更新版本,)

傳回值 :

OS_ : Message 應該使用的 OS 版本 (長度: 48)
例如: "Must use 8n020704.img" 應更新爲 8n020704.img

DRIVER_ : Message 目前的驅動程式版本 (長度: 48)
例如: "I-8xx7 : isa.exe - 2.19 , Dec.09,2002"

GETCTS

□ I-8417/8817 □ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function Block

取得 COM 埠的 CTS , 有效 COM 埠爲 3 ~ 5

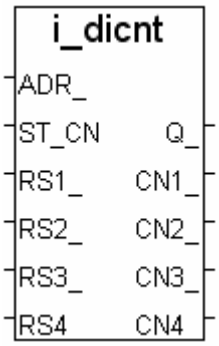
輸入參數 :

PORT_ : Integer 3:COM3 , 4:COM4, 5:COM5

傳回值 :

Q_ : Boolean 成功.: TRUE , 失敗 : FALSE

I_DICNT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

取得 與 I-8xx7 的 COM3/4 或 與 I-7188XG/EG 的 COM2 或 與 W-8xx7 的 COM3 連接的外接 RS-485 remote DI 模組的 4 個 DI counter 值.

輸入參數 :

ADR_ :	Integer	I/O 模組的位址 (1-255), 需爲常數, 不能爲變數
ST_CN_ :	Integer	起始 Channel 編號. 需爲常數, 不能爲變數. 有效值 1 到 13. Ex: I-87052 有 8 個 DI. 若 ST_CN_ 設定爲 5, 則 CN1_ 取得 Ch.5 的 counter 值, CN2_ 取得 Ch.6, CN3_ 取得 Ch.7, CN4_ 取得 Ch.8 的值
RS1_ ~ RS4_ :	Boolean	當 False 上升爲 True, 重置相對應的 D/I counter 爲 0.

傳回值 :

Q_ :	Boolean	成功.: TRUE , 失敗 : FALSE 若 Q_ 傳回 FALSE, 表示通訊失敗, 下列回傳值無意義
CN1_ ~ CN4_ :	Integer	由 ST_CN_起始的 channel 編號的 DI Counter 值爲 0.

****下列模組有 DI counters (最大 100 Hz) :**

"I_DICNT" 支援 : i-87051 , 87052 , 87053 , 87054 , 87055 , 87058 , 87063
i-7041, 7044, 7050 , 7051, 7052 , 7053 , 7055, 7058, 7060 , 7063 , 7065

"I_DICNT2" 支援 : i-87040

Counter input channels: 4 , 有效值 0 到 65535 (最大 100 Hz)

Reset Counter channels: 4 , 當 False 上升爲 True, 重置相對應 channel 的 D/I counter 爲 0.

注意 :

1. i-DiCnt 函式方塊適用版本 : I-8xx7:2.18 , I-7188EG:1.10 , I-7188XG:1.08 , W-8x37:3.20C 或更新版本
2. 遠程 I-87041 模組(32 D/I) 請使用 "i_DiCnt2" 函式方塊

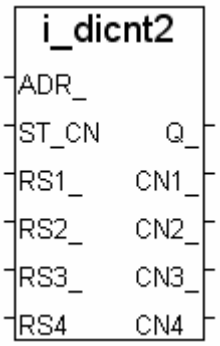
特別重要 :

使用 RS-485 Remote I/O, 連接之前請注意下列各項 :

A. 在 I/O 模組端, 請使用 "DCON Utility" 來設定 :
   *1. 設定 "address" 爲獨一無二的位址編號. (1~255)
   *2. 類比輸入模塊需設爲 "2's complement" 格式, 類比輸出模塊需設爲 "Engineering" 格式
   *3. 設定通訊參數的 "baud rate" 及 "8,N,1"
   *4. 若爲類比板卡要設定爲 "Range Type"

B. 在 I-8xx7/I-7188EG/I-7188XG /W-8xx7 主控端 :
   *1. 連結 I/O 請選擇 “complex equipment “ 及 "bus7000B", 並設定其 "baud rate" 與 I/O 模組的 "baud rate" 相同, Checksum 也需相同.
   *2. I-8xx7, 7188EG/XG 最多可連接 64 個遠程 I/O 模組, W-8xx7 最多 255 個
   *3. 程式內連接 RS-485 Remote I/O 模組請選用 i_7*** function block 或 i_87*** Function Block.

I_DICNT2



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

取得 與 I-8xx7 的 COM3/4 或 與 I-7188XG/EG 的 COM2 或 與 W-8xx7 的 COM3 連接的外接 RS-485 remote DI counter 值.

輸入參數 :

ADR_ :	Integer	I/O 模組的位址 (1-255), 需爲常數, 不能爲變數
ST_CN_ :	Integer	起始 Channel 編號. 需爲常數, 不能爲變數. 有效值 1 到 29. Ex: I-87040 有 32 個 DI. 若 ST_CN_ 設定爲 15, 則 CN1_ 取得 Ch.15 的 counter 值, CN2_ 取得 Ch.16, CN3_ 取得 Ch.17, CN4_ 取得 Ch.18 的值
RS1_ ~ RS4_ :	Boolean	當 False 上升爲 True, 重置相對應的 D/I counter 爲 0.

傳回值 :

Q_ :	Boolean	成功.: TRUE , 若 Q_ 傳回 FALSE, 表示通訊失敗, 下列回傳值無意義
CN1_ ~ CN4_ :	Integer	由 ST_CN_起始的 channel 編號的 DI Counter 值爲 0.

****下列模組有 DI counters (最大 100 Hz) :**

"I_DICNT" 支援 : i-87051 , 87052 , 87053 , 87054 , 87055 , 87058 , 87063
i-7041, 7044, 7050 , 7051, 7052 , 7053 , 7055, 7058, 7060 , 7063 , 7065

"I_DICNT2" 支援 : i-87040,

Counter input channels: 4 , 有效值 0 到 65535 (最大 100 Hz)

Reset Counter channels: 4 , 當 False 上升爲 True, 重置相對應 channel 的 D/I counter

注意 :

1. i-DiCnt 函式方塊適用版本 : I-8xx7:2.18 , I-7188EG:1.10 , I-7188XG:1.08 , W-8x37:3.20C 或更新版本
2. 遠程 I-87041 模組(32 D/I) 請使用 "i_DiCnt2" 函式方塊

特別重要 :

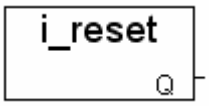
使用 RS-485 Remote I/O, 連接之前請注意下列各項 :

A. 在 I/O 模組端, 請使用 "DCON Utility" 來設定 :
   *1. 設定 "address" 爲獨一無二的位址編號. (1~255)
   *2. 類比輸入模塊需設爲 "2's complement" 格式, 類比輸出模塊需設爲 "Engineering" 格式
   *3. 設定通訊參數的 "baud rate" 及 "8,N,1"
   *4. 若爲類比板卡要設定爲 "Range Type"

B. 在 I-8xx7/I-7188EG/I-7188XG /W-8xx7 主控端 :
   *1. 連結 I/O 請選擇 “complex equipment “ 及 "bus7000B", 並設定其 "baud rate" 與 I/O 模組的 "baud rate" 相同, Checksum 也需相同.
   *2. I-8xx7, 7188EG/XG 最多可連接 64 個遠程 I/O 模組, W-8xx7 最多 255 個
   *3. 程式內連接 RS-485 Remote I/O 模組請選用 i_7*** function block 或 i_87*** Function Block.

I_RESET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

重新開機

傳回值 :

Q_ : Boolean 無作用

注意 : 使用本函式須非常小心謹慎.

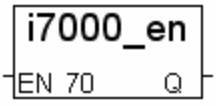
1. 對於 I-8417/8817/8437/8837 , I-7188EG , I-7188XG :
   若控制器一直重置, 請參考 "User's Manual Of The I-8417/8817/8437/8837" 使用手冊 第 1.3.7 節的說明 刪除控制器內的專案.
2. 對於 W-8xx7/8xx6 (Wincon ISaGRAF 版本) :
   若控制器一直重置, 請暫時拔除 CF 卡然後重新啓動 WinCon 電源, 接著再次插上 CF 卡然後刪除以下檔案 - "\CompactFlash\ISaGRAF\ISA11". 之後, 再次重新啓動 WinCon 電源.

範例 :

```
(* OK1 宣告爲 boolean input, TMP 宣告爲 boolean internal *)
if  OK1=TRUE  then
  TMP := i_reset();
end_if;
```

I7000_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

啓動 / 停止 "Bus7000"

輸入參數 :

EN_7000_ : Boolean TRUE: 啓動, FALSE: 停止

傳回值 :

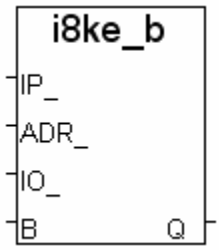
Q_ : Boolean 永遠傳回 TRUE.

注意 :

1. 預設值是 啓動.
2. 只在 IO 複合設備 "Bus7000" 與 "Bus7000B" 連接狀態下才有效.

I8KE_B

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定 ISaGRAF Boolean 變數對應到一個 i8KE4/8-MTCP 的 Boolean I/O 點

輸入參數 :

IP_ :	Message	相關 i8KE4/8-MTCP 的 IP 位址, 例如 : '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 內 DI 或 DO 的 Modbus 位址, 0 到 267
IO_ :	Boolean	True: 輸入, False: 輸出
B_ :	Boolean	布林變數名稱

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

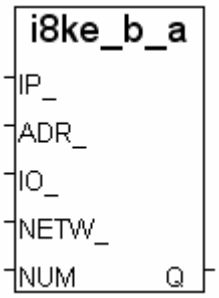
注意 :

1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.

範例程式: Wdemo_30 & Wdemo_31 at http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_B_A

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定 ISaGRAF Boolean 變數陣列對應到數個 i8KE4/8-MTCP 的 Boolean I/O 點. (請參考 2.6 節 關於變數陣列的說明)

輸入參數 :

IP_ :	Message	相關 i8KE4/8-MTCP 的 IP 位址, 例如 : '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 內 DI 或 DO 的 Modbus 位址, 0 到 267
IO_ :	Boolean	True: 輸入, False: 輸出
NetW_ :	Integer	"變數陣列"第一個元素的網路位址編號. 1 ~ 8191
Num_ :	Integer	使用乙太網路 IO 時設定變數陣列布林的數量, 有效範圍: 1 ~ 255. (ADR_ + Num_) 不能大於 264. Ex: Bi[0..15] 大小爲 16, NUM_ 可設爲 1 ~ 16. ABC[0..7] 大小爲 8, NUM_ 可設爲 1 ~ 8.

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

注意 :

1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.
3. ISaGRAF 版本 3.4 (或 3.5) 的 "變數陣列" 宣告方式, 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 之下 "isa.ini" 檔案的最頂端加 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictionary 的宣告視窗裡會增加一個 "DIM" 欄位，在此設定陣列.

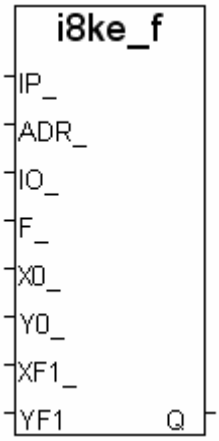
請在 c:\isawin\exe\isa.ini 檔案最頂端, 加進 2 行程式碼 :

```
[DEBUG]
arrays=1
```

範例程式 : Wdemo_30 和 Wdemo_31 請參考 http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_F

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定 ISaGRAF Real 變數對應到一個 i8KE4/8-MTCP 的類比 I/O 點. 並轉換爲 Real 格式

輸入參數 :

IP_ :	Message	相關 i8KE4/8-MTCP 的 IP 位址, Ex: '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 的 AI/AO Modbus 位址: 0~127
IO_ :	Boolean	True: 輸入 , False: 輸出
F_ :	REAL	REAL 變數名稱
----- 下列參數供數值轉換之用, 若不需轉換, 參數請設爲 (0 , 0 , 0.0 , .0) -----		
X0_ :	Integer	類比輸入/輸出卡的原始值. X0_ **不能等於** Y0_. 有效範圍: -32768 <= X0_ <= +32767
Y0_ :	Integer	類比輸入/輸出卡的原始值.. X0_**不能等於** Y0_. 有效範圍: -32768 <= Y0_ <= +32767
XF1_ :	REAL	轉換後的工程值. XF1_**不能等於** YF1_ .
YF1_ :	REAL	轉換後的工程值. XF1_**不能等於** YF1_ .

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

範例 :

Ex 1: 若 i-8017h 設定的 range_type 爲 '+/- 10 V' (i-8017h's input value is -32768 to +32767). 使用者要將 (0 , 10 V) 轉換爲工程值 (0 , 1000 Psi). 請設定 (X0_ , Y0_) = (0 , +32767) , (XF1_ , YF1_) = (0.0 , 1000.0)

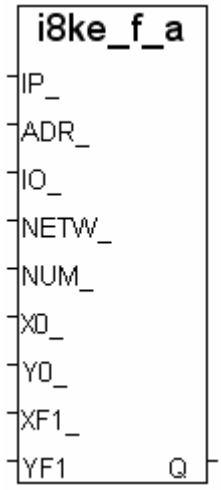
Ex 2: 若 i-8024 設定 range_type 爲 '0 to 20 mA' (i-8024's output value is 0 to +32767). 使用者要將 (4 , 20 mA) 轉換爲工程值 (0 , 3000 rpm). 請設定 (X0_ , Y0_) = (6553 , +32767) , (XF1_ , YF1_) = (0.0 , 3000.0)

注意 :

1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.

範例程式: Wdemo_30 和 Wdemo_31 請參考 http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_F_A



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

設定 ISaGRAF Real 變數陣列對應到數個 i8KE4/8-MTCP 的類比 I/O 點.
並轉換爲 REAL 格式 (請參考 2.6 節 關於變數陣列的說明)

輸入參數 :

IP_ :	Message	對應 i8KE4/8-MTCP 的 IP 位址,ex: '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 內 AI 或 AO 的 Modbus 位址: 0~127
IO_ :	Boolean	True: 輸入 , False: 輸出
NetW_ :	Integer	"變數陣列" 第一個元素的網路位址. 1~8191.
Num_ :	Integer	使用乙太網路 IO 時設定變數陣列 REAL 的數量, 有效範圍: 1 ~ 255. (ADR_ + Num_) 不能大於 128. Ex: R1[0..31] 大小爲 32, NUM_ 可設爲 1 ~ 32. R3[0..7] 大小爲 8, NUM_ 可設爲 1 ~ 8.
----- 下列參數供數值轉換之用, 若不需轉換, 參數請設爲 (0 , 0 , 0.0 , .0) -----		
X0_ :	Integer	類比輸入/輸出卡的原始值. X0_ **不能等於** Y0_. 有效範圍: -32768 <= X0_ <= +32767
Y0_ :	Integer	類比輸入/輸出卡的原始值.. X0_**不能等於** Y0_. 有效範圍: -32768 <= Y0_ <= +32767
XF1_ :	REAL	轉換後的工程值. XF1_**不能等於** YF1_ .
YF1_ :	REAL	轉換後的工程值. XF1_**不能等於** YF1_ .

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

範例 :

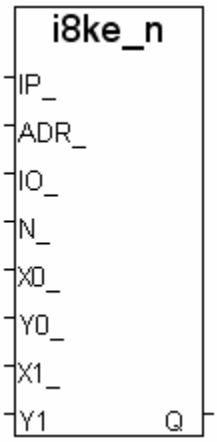
Ex : 若 i-87024 設定 range_type 爲 '4 to 20 mA' (i-87024 輸出值爲 0 ~ +32767). 使用者要將 (4 , 20 mA) 轉換爲工程值(0 , 5000 rpm). 請設定 (X0_ , Y0_) = (0 , +32767) , (XF1_ , YF1_) = (0.0 , 5000.0)

注意 :

1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.
3. ISaGRAF 版本 3.4 (或 3.5) 的 "變數陣列" 宣告方式, 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 之下 "isa.ini" 檔案的最頂端加 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictionary 的宣告視窗裡會增加一個 "DIM" 欄位，在此設定陣列.
   請在 c:\isawin\exe\isa.ini 檔案最頂端, 加進 2 行程式碼 :
   [DEBUG]
   arrays=1

範例程式 : Wdemo_30 及 Wdemo_31 請參考 http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_N



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

設定 ISaGRAF Integer 變數對應到一個 i8KE4/8-MTCP 的類比 I/O 點.
並轉換爲 Integer 格式

輸入參數 :

IP_ :	Message	對應 i8KE4/8-MTCP 的 IP 位址, Ex: '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 的 AI/AO Modbus 位址: 0~127
IO_ :	Boolean	True: 輸入 , False: 輸出
N_ :	Integer	Integer 變數名稱

----- 下列參數供數值轉換之用, 若不需轉換, 參數請設爲 (0 , 0 , 0.0 , .0) -----

X0_ :	Integer	類比輸入/輸出卡的原始值. X0_ **不能等於** Y0_. 有效範圍: -32768 <= X0_ <= +32767
Y0_ :	Integer	類比輸入/輸出卡的原始值.. X0_**不能等於** Y0_. 有效範圍: -32768 <= Y0_ <= +32767
X1_ :	Integer	比例後的工程值. X1_**不能等於** Y1_ . 有效範圍: -30000 <= X1_ <= +30000
Y1_ :	Integer	比例後的工程值. X1_**不能等於** Y1_ . 有效範圍: -30000 <= Y1_ <= +30000

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

範例 :

Ex 1: i-8017h 設定的 range_type 爲 '+/- 10 V' (i-8017h 輸入值爲 -32768 ~ +32767). 使用者要將 (0 , 10 V) 轉換爲工程值 (0 , 1000 Psi). 請設定 (X0_ , Y0_) = (0 , +32767) , (X1_ , Y1_) = (0 , 1000)

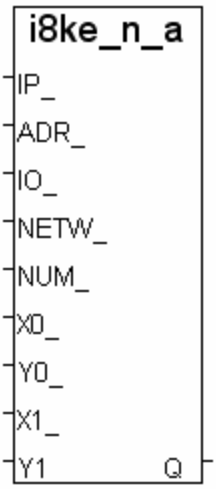
Ex 2: i-8024 設定 range_type 爲 '0 to 20 mA' (i-8024 輸出值爲 0 ~ +32767). 使用者要將(4 , 20 mA) 轉換爲工程值 (0 , 3000 rpm). 請設定 (X0_ , Y0_) = (6553 , +32767) , (X1_ , Y1_) = (0 , 3000)

注意 :

1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.

範例程式: Wdemo_30 及 Wdemo_31 請參考 http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_N_A



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

設定 ISaGRAF Integer 變數陣列對應到數個 i8KE4/8-MTCP 的類比 I/O 點. 並轉換爲 Integer 格式 (請參考 2.6 節 關於變數陣列的說明)

輸入參數 :

IP_ :	Message	對應 i8KE4/8-MTCP 的 IP 位址,ex: '192.168.100.123'
ADR_ :	Integer	i8KE4/8-MTCP 內 AI 或 AO 的 Modbus 位址 . 0~127
IO_ :	Boolean	True: 輸入 , False: 輸出
NetW_ :	Integer	"變數陣列" 第一個元素的網路位址. 1~8191.
Num_ :	Integer	使用乙太網路 IO 時設定變數陣列 Integer 的數量, 有效範圍: 1 ~ 255. (ADR_ + Num_) 不能大於 128. Ex: ENG1[0..63] 大小爲 64, NUM_ 可設爲 1 ~ 64. Ai[0..7] 大小爲 8, NUM_ 可設爲 1 ~ 8.
----- 下列參數供數值轉換之用, 若不需轉換, 參數請設爲 (0 , 0 , 0.0 , .0) -----		
X0_ :	Integer	類比輸入/輸出卡的原始值. X0_ **不能等於** Y0_. 有效範圍: -32768 <= X0_ <= +32767
Y0_ :	Integer	類比輸入/輸出卡的原始值.. X0_**不能等於** Y0_. 有效範圍: -32768 <= Y0_ <= +32767
X1_ :	Integer	轉換後的工程值. X1_**不能等於** Y1_ . 有效範圍: -30000 <= X1_ <= +30000
Y1_ :	Integer	轉換後的工程值. X1_**不能等於** Y1_ . 有效範圍: -30000 <= Y1_ <= +30000

傳回值 :

Q_ :	Boolean	True: Ok. False: 參數錯誤.

範例 :

若 i-87018R 設定的 range_type 爲 'Thermo-Couple K-type: -270 to +1372 degree celsius' (i-87018R 輸入值爲 -6448 ~ +32767). 使用者要將(-270 , +1372 degree) 轉換爲工程值(-2700 , +13720). 請設定 (X0_ , Y0_) = (-6448 , +32767) , (X1_ , Y1_) = (-2700 , +13720)

注意 :

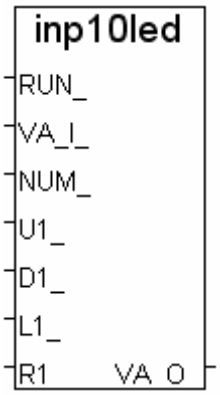
1. 請參考下列網址取得更詳細資料
   http://www.icpdas.com/faq/isagraf.htm 'FAQ042' 及
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A 需在第一次 PLC scan 時呼叫. 第二次以後呼叫無效.
3. ISaGRAF 版本 3.4 (或 3.5) 的 "變數陣列" 宣告方式, 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 之下 "isa.ini" 檔案的最頂端加 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictionary 的宣告視窗裡會增加一個 "DIM" 欄位，在此設定陣列.

   請在 c:\isawin\exe\isa.ini 檔案最頂端, 加進 2 行程式碼 :

   ```
   [DEBUG]
   arrays=1
   ```

範例程式 : Wdemo_30 及 Wdemo_31 請參考 http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

INP10LED



■ I-8417/8817 ■ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從 S-MMI 輸入 1 個 10 進位的整數

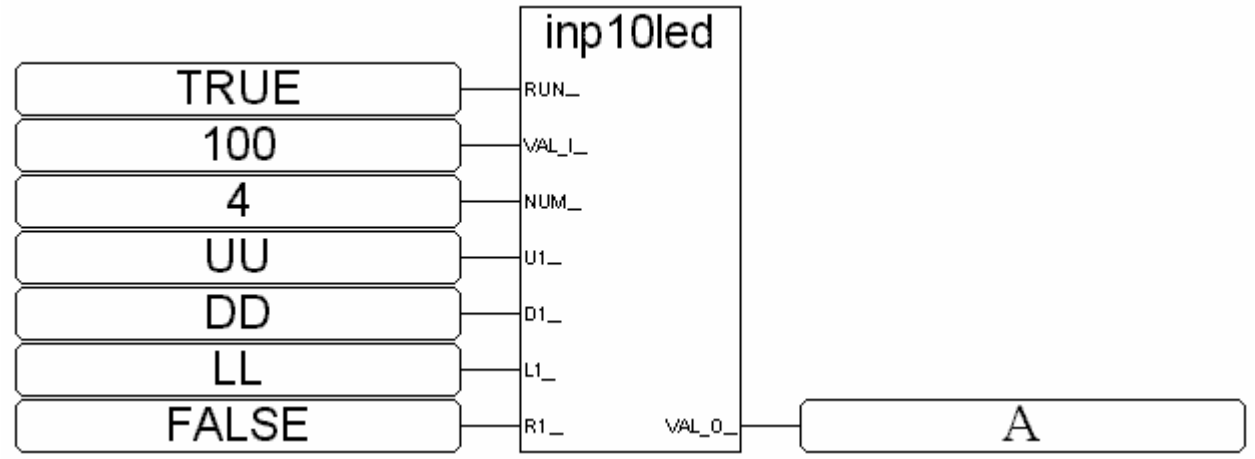
輸入參數 :

RUN_	Boolean	當爲 TRUE 時才動作
VAL_I_	Integer	顯示在 S-MMI 的初值 (0 ~ 99999)
NUM_	Integer	要顯示幾位數 (1 ~ 5)
U1_	Boolean	由 FALSE 升到 TRUE 時,目前位置的值加 1
D1_	Boolean	由 FALSE 升到 TRUE 時,目前位置的值減 1
L1_	Boolean	由 FALSE 升到 TRUE 時,目前位置往左移
R1_	Boolean	由 FALSE 升到 TRUE 時,目前位置往右移

傳回值 :

VAL_O_	Integer	經過操作後的整數值

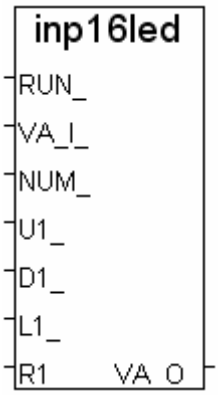
範例: 請參考 demo_08 及 demo_11a.



ST 相等式:

```
A := INP10LED(TRUE,100,4,UU,DD,LL,FALSE);
(* A 需宣告 integer , 屬性爲 internal *)
(* UU,DD,LL 可宣告爲 boolean, 屬性爲 input, 可連接至 push4key *)
```

INP16LED



■ I-8417/8817 ■ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從 S-MMI 輸入 1 個 16 進位的整數

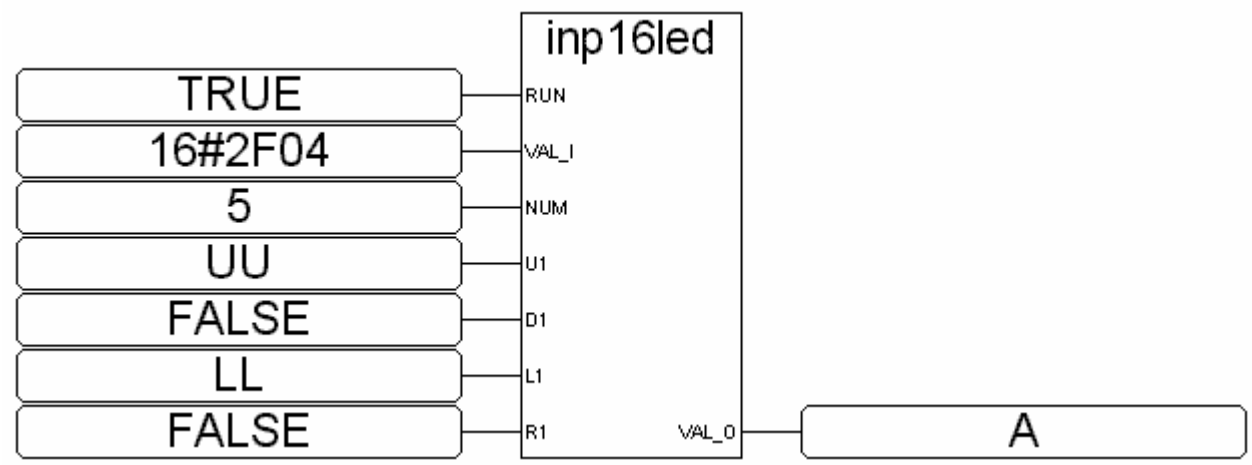
輸入參數 :

RUN_	Boolean	當爲 TRUE 時才動作
VAL_I_	Integer	顯示在 S-MMI 的初值 (0 ~ 16#FFFFF)
NUM_	Integer	要顯示幾位數 (1 ~ 5)
U1_	Boolean	由 FALSE 升到 TRUE 時,目前位置的值加 1
D1_	Boolean	由 FALSE 升到 TRUE 時,目前位置的值減 1
L1_	Boolean	由 FALSE 升到 TRUE 時,目前位置往左移
R1_	Boolean	由 FALSE 升到 TRUE 時,目前位置往右移

傳回值 :

VAL_O_	integer	經過操作後的整數值

範例:



ST 相等式:

```
A := INP16LED(TRUE,16#2F04,4,UU,FALSE,LL,FALSE);
(* A 需宣告 integer , 屬性爲 internal *)
(* UU,LL 可宣告爲 boolean, 屬性爲 input, 可連接至 push4key *)
```

INT_REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

對應 1 個長整數成爲 1 個實數. 以 C 語言的語法表示爲 Real_ = *((float *)&Long_)

輸入參數 :

Long_	Integer	要被對應的 integer 值

傳回值 :

Real_	Real	對應完後的 real 值

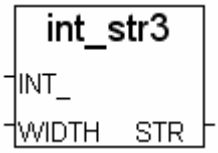
注意:

1. “Real_Int" 可用來對應 real 成 integer
2. 如果只是要 轉換 一個整數爲 實數, 比如 32 → 32.0 , 請用 Real( )

非常重要:

誤用 "**int_real**(L1)" 可能會使控制器產生 Float Error, 請參考 ISaGRAF 手冊 10.6 節.

INT_STR3



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

將 Integer 轉換爲 String, 有長度限制

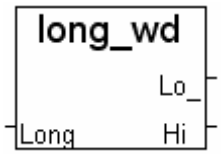
輸入參數 :

INT_ :	Integer	要轉換的 integer 值.
WIDTH_ :	Integer	指定最大顯示位數, 1 ~ 13

傳回值 :

STR_ :	Message	傳回的字串 (最大長度: 13). Ex. 若 WIDTH_ = 4 12 ---> ' 12' 123456 ---> '****'

LONG_WD



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

將 1 個長整數 (signed 32-bit) 轉換成 2 個 word (signed 16-bit)

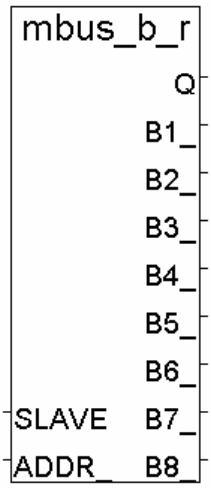
輸入參數 :

LONG_ :	Integer	要轉換的 32-bit integer 值.

傳回值 :

LO_	Integer	轉換後的 low word (-32768 ~ +32767)
HI_	Integer	轉換後的 high word (-32768 ~ +32767)

MBUS_B_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Modbus 設備一次讀取 8 個 Boolean 值

使用 Modbus function 1

輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 位址開始讀(0~65535). , 需爲常數值, 非變數

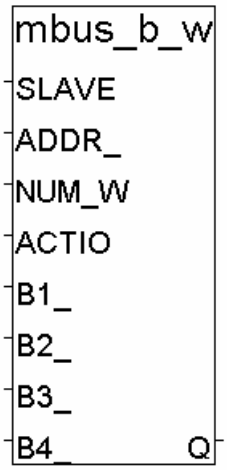
傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
B1_ ~ B8_	Boolean	讀到的 8 個 Boolean 值

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_B_R + MBUS_BR1).
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章

MBUS_B_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 1 到 4 個 Boolean 值到 Modbus 設備

當 NUM_W=1 時, 使用 Modbus function 5
當 NUM_W=2 到 4 時, 使用 Modbus function 15

輸入參數 :

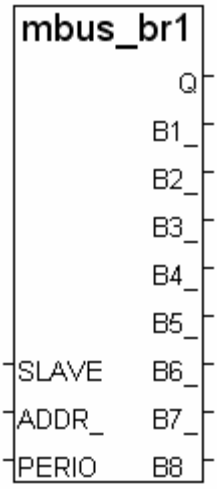
SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始寫(0~65535) 需爲常數值, 非變數
NUM_W_	Integer	要寫幾個 boolean 值 (1 ~ 4) , 需爲常數值, 非變數
ACTION_	Boolean	設爲 TRUE 才寫
B1_ ~ B4_	Boolean	要寫出的 boolean 值

傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_B_W + MBUS_WB).
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章



MBUS_BR1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

每隔一段時間 從 Modbus 設備一次讀取 8 個 Boolean 值

使用 Modbus function 1

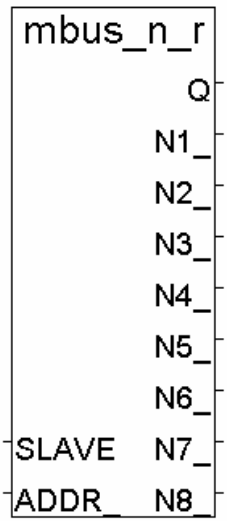
輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) 需爲常數值, 非變數
PERIOD_	Integer	每隔多久去讀 (1~600), 單位爲 秒

傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
B1_ ~ B8_	Boolean	讀到的 8 個 Boolean 值

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_B_R + MBUS_BR1).
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個



MBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Modbus 設備一次讀取 8 個 Word (signed 16-bit)

使用 Modbus function 3

輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數

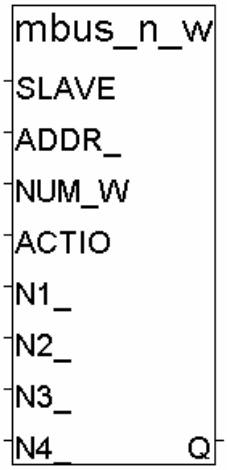
傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
N1_ ~ N8_	Integer	讀到的 8 個 Word 值 (-32768 ~ 32767)

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_N_R + MBUS_R + MBUS_NR1 + MBUS_R1).
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章

MBUS_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 1 到 4 個 word (signed 16-bit) 值到 Modbus 設備

當 NUM_W=1 時, 使用 Modbus function 6
當 NUM_W=2 到 4 時, 使用 Modbus function 16

輸入參數 :

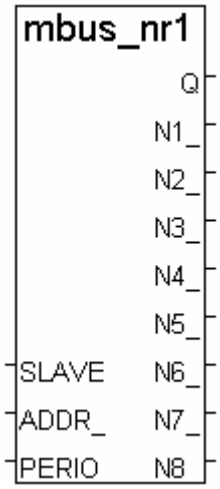
SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數
NUM_W_	Integer	要寫幾個 word 值 (1 ~ 4) , 需爲常數值, 非變數
ACTION_	Boolean	設爲 TRUE 才寫
N1_ ~ N4_	Integer	要寫出的 word 值 (-32768 ~ 32767)

傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 MBUS_N_W.
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章



MBUS_NR1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

每隔一段時間從 Modbus 設備一次讀取 8 個 Word (signed 16-bit)

使用 Modbus function 3

輸入參數 :

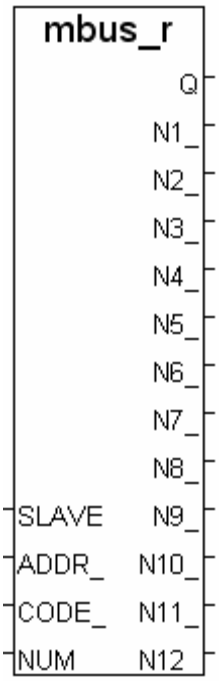
SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數
PERIOD_	Integer	每隔多久去讀 (1~600), 單位爲 秒

傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
N1_ ~ N8_	Integer	讀到的 8 個 Word 值 (-32768 ~ 32767)

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1)
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章



MBUS_ R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

從 Modbus 設備 讀取資料

* 可選擇使用 Modbus function 1 或 2 或 3 或 4

輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數
CODE_	Integer	使用那個 Modbus function 編號, 1 – 4 需爲常數值, 非變數
NUM_	Integer	如果 CODE_=1 或 2 表示詢問幾個 bit ? 1-192 如果 CODE_=3 或 4 表示詢問幾個 word ? 1- 12 需爲常數值, 非變數

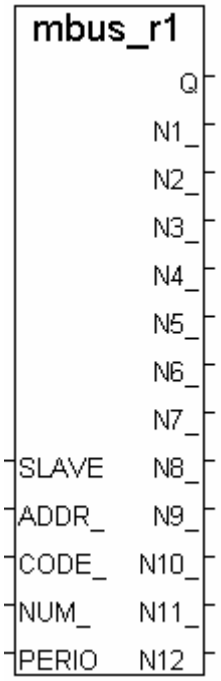
傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
N1_ ~ N12_	Integer	讀到的 Bit 或 Word 值 如果 **CODE_**=1 或 2, N1_傳回 bit 1 到 16, N2_ 傳回 bit 17 到 32, ... N12_傳回 bit 177 到 192 如果 **CODE_**=3 或 4, N1_ 到 N12_ 傳回 Word 1 到 12 (值介於-32768 到 32767).

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1)

W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章



MBUS_R1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

每隔一段時間 從 Modbus 設備 讀取資料

* 可選擇使用 Modbus function 1 或 2 或 3 或 4

輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數
CODE_	Intege	使用那個 Modbus function 編號, 1 – 4 需爲常數值, 非變數
NUM_	Integer	如果 CODE_=1 或 2 表示詢問幾個 bit ? 1-192 如果 CODE_=3 或 4 表示詢問幾個 word ? 1- 12 需爲常數值, 非變數
PERIOD_	Integer	每隔多久去讀 (1~600), 單位爲 秒

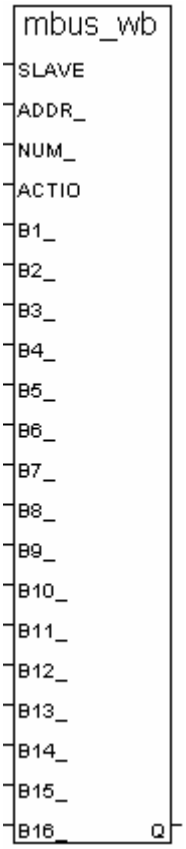
傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE
N1_ ~ N12_	Integer	讀到的 Bit 或 Word 值 如果 **CODE_**=1 或 2, N1_傳回 bit 1 到 16, N2_ 傳回 bit 17 到 32, ... N12_傳回 bit 177 到 192 如果 **CODE_**=3 或 4, N1_ 到 N12_ 傳回 word 1 到 12 (值介於-32768 ~ 32767).

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1)
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章

MBUS_WB



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

寫出 1~16 個 boolean 值到 Modbus 設備

使用 Modbus function 15

輸入參數 :

SLAVE_	Integer	值需爲常數值, 非變數 Modbus 設備的站號 = SLAVE_ / 1000 的餘數 ISaGRAF Controller 使用的 COM port = SLAVE_ / 1000 的商數 I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5) I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3) I-7188XG : (2:COM2, 3:COM3) W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 到 14:COM14) 例如: SLAVE_ = 3001, 則使用 COM3, 設備的站號是 1.
ADDR_	Integer	從那個 Modbus 資料位址開始讀(0~65535) , 需爲常數值, 非變數
NUM_W_	Integer	要寫幾個 boolean 值 (1 ~ 16) , 需爲常數值, 非變數
ACTION_	Boolean	設爲 TRUE 才寫
B1_ ~ B16_	Boolean	要寫出的 boolean 值

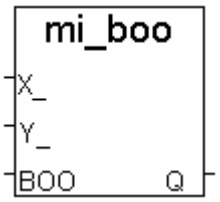
傳回值 :

Q_	Boolean	正確傳回 TRUE, 失敗傳回 FALSE

注意: 同一台 I-8xx7 或 I-7188EG/XG 控制器最多只能使用 64 個 (MBUS_B_W + MBUS_WB)
W-8xx7 控制器的每一個 COM Port 則最多使用 256 個

範例: 請參考第 8 章

MI_BOO



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

在 MMICON 上顯示 1 個 Boolean 值

輸入參數 :

X_	Integer	位置 X, 1-30
Y_	Integer	位置 Y, 1-8
BOO_	Boolean	要顯示的 boolean 值. TRUE 顯示“ON”, FALSE 顯示 “OFF”

傳回值 :

Q_	Boolean	Ok. 傳回 TRUE, 失敗爲 FALSE

範例: 請參考第 16 章 & demo_38, demo_39

MI_INP_N

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

在 MMICON 上輸入 1 個 integer 值

輸入參數 :

EN_	Boolean	TRUE: 才動作
INIT_	Integer	初值

傳回值 :

INT_	Integer	傳回的 integer 輸入值. 如果 EN_ 設爲 FALSE , 傳回 0

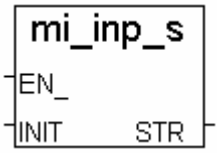
注意 :

MI_INP_N 及 MI_INP_S 只可在 1 個 ISaGRAF Project 內使用 1 次, 同時在同個 Project 內的兩個或以上的地方出現會無法正常工作.

範例: 請參考第 16 章 & demo_38, demo_39

MI_INP_S

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

在 MMICON 上輸入 1 個字串

輸入參數 :

EN_	Boolean	TRUE: 才動作
INIT_	Message	字串初值

傳回值 :

STR_	Message	傳回的字串輸入值. 如果 EN_ 設爲 FALSE , 傳回 '' (空字串)

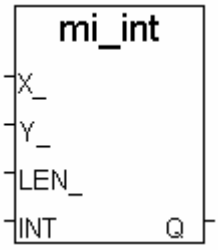
注意:

MI_INP_N 及 MI_INP_S 只可在 1 個 ISaGRAF Project 內使用 1 次, 同時在同個 Project 內的兩個或以上的地方出現會無法正常工作.

範例: 請參考第 16 章 & demo_38, demo_39

MI_INT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

在 MMICON 上顯示 1 個 integer 值

輸入參數 :

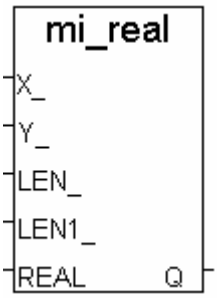
X_	Integer	位置 X, 1-30
Y_	Integer	位置 Y, 1-8
LEN_	Integer	最多可顯示幾個字元, 1-11
INT_	Integer	要顯示的 integer 值, 長度超過 LEN_會顯示'******'

傳回值 :

Q_	Boolean	Ok. 傳回 TRUE, 失敗爲 FALSE

範例: 請參考第 16 章 & demo_38, demo_39

MI_REAL



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

在 MMICON 上顯示 1 個 real 值

輸入參數 :

X_	Integer	位置 X, 1-30
Y_	Integer	位置 Y, 1-8
LEN_	Integer	最多可顯示幾個字元, 1-13
LEN1_	Integer	小數點後可顯示幾個字元,0~4 且值需比 LEN_小. 例, 若 LEN_=7, LEN1_=2, "123.4567" 會顯示成 " 123.45"
REAL_	Real	要顯示的 Real 值. 若位數超過 LEN_, 會顯示'******'

傳回值 :

Q_	Boolean	Ok. 傳回 TRUE, 失敗爲 FALSE

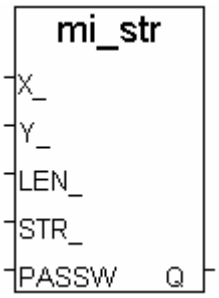
注意:

假如要顯示的實數的絕對值 (>= 1,000,000) 或 (不等於 0 且 < 0.0001) , 請設定 LEN_ 爲 13.

例, -123,456,789, 請設 LEN_ 爲 13, 它會顯示爲 -1.23457e+008. 又如 0.0000123456, 請設 LEN_ 爲 13, 它會顯示爲 1.23456e-005

範例: 請參考第 16 章 & demo_38, demo_39

MI_STR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

在 MMICON 上顯示 1 個字串

輸入參數 :

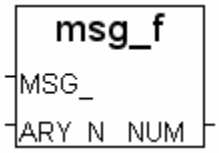
X_	Integer	位置 X, 1-30
Y_	Integer	位置 Y, 1-8
LEN_	Integer	最多可顯示幾個字元, 1-240
STR_	Message	要顯示的字串. 假如字元數量超過 LEN_, 超過的部份不顯示
PASSWD_	Boolean	設爲 TRUE 表示要顯示成密碼, 所有字元以 * 代替, 設爲 FALSE 則爲正常顯示.

傳回值 :

Q_	Boolean	Ok. 傳回 TRUE, 失敗爲 FALSE

範例: 請參考第 16 章 & demo_38, demo_39

MSG_F



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從字串(Message) 讀出一些 REAL 值(浮點值), 並儲存在浮點陣列

輸入參數 :

MSG_	Message	要讀出的 Message(字串), 1~255 bytes (characters)
ARY_NO_	Integer	存放讀出的 REAL 值的浮點陣列編號 I-8xx7, I-718xEG/XG : 1 ~ 6 (其 **Float 陣列和 Integer 陣列**使用同一個的記憶體, 請小心使用) **W**-8xx7/8xx6 : 1 ~ 18

傳回值 :

NUM_	Integer	讀到的 REAL 值的總數 (0 ~ 128), 若傳回 -1, 表示格式錯誤

注意:

1. REAL 值可以使用 "ARY_F_R" 和 "ARY_F_W" 函式讀出和寫入.
2. 字串的 REAL 值之間隔開需用"空格"或"逗號"或"TAB" 或 "ENTER" 或 "NEW LINE" 字元
3. 若字串中對應的值不是正確的 REAL 格式，例如: ' 1.23 ， 2.45A ， 3.0 , 2+3 ' 中的第二和第四個值不是正確的 REAL 格式, 函式傳回的值就會是 -1.
4. 從以下版本起的驅動程式才有支援 MSG_F 與 MSG_N:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

範例:

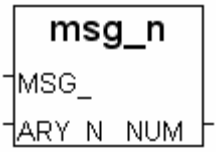
(1.) 若指定 MSG_ = '2.3 , -567.002 , +0.0025 , 1 ， 1.4E-5' , 而 ARY_NO_ = 1 , 則傳回 NUM_ 爲 5. 浮點陣列中儲存的 5 個 REAL 值, 由編號 1 到 5 的位址分別是:
addr.1 = 2.3 ， addr.2 = -567.002 ， addr.3 = 0.0025 ， addr.4 = 1.0 ， addr.5 = 1.4E-5

(2.) 若指定 MSG_ = '4.01 ， -8.09 ， +-3.45' , 而 ARY_NO_ = 2 , 則會傳回 -1 , 因爲第 3 個值+-3.45 是不正確的 REAL 格式.

範例程式 :

1. wdemo_52.pia 放置於 W-8xx7 CD-ROM:\napdos\isagraf\wincon\demo\
2. wdemo_52.pia 放在 ftp://ftp.icpdas.com/pub/cd/wincon_isagraf/napdos/isagraf/wincon/demo/

MSG_N



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從字串(Message) 讀出一些 Integer 值, 並儲存在 Integer 陣列

輸入參數 :

MSG_	Message	要讀出的 Message(字串), 1~255 bytes (characters)
ARY_NO_	Integer	存放讀出的 Integer 值的 Integer 陣列編號 I-8xx7, I-718xEG/XG : 1 ~ 6 (其 **Float 陣列和 Integer 陣列**使用同一個的記憶體, 請小心使用) **W**-8xx7/8xx6 : 1 ~ 18

傳回值 :

NUM_	Integer	讀到的 Integer 值的總數 (0 ~ 128), 若傳回 -1, 表示格式錯誤

注意:

1. Integer 值可以使用 "ARY_N_R" 和 "ARY_N_W" 函式讀出和寫入.
2. 字串的 Integer 值之間隔開需用"空格"或“逗號”或“TAB” 或 “ENTER” 或 “NEW LINE” 字元
3. 若字串中對應的值不是正確的 Integer 格式，例如: ' 123 ， -8G ， 3 ' 中的第二個值不是正確的 Integer 格式, 函式傳回的值就會是 -1.
4. 從以下版本起的驅動程式才有支援 MSG_F 與 MSG_N:
   I-7188EG: 2.17 版起
   I-7188XG: 2.15 版起
   I-8XX7 : 3.19 版起
   W-8XX7 : 建議更新至 3.36 版或更高版

範例:

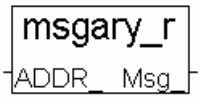
(1.) 若指定 MSG_ = ' 3 , -567 +2 ' , 而 ARY_NO_ = 1 , 則傳回 NUM_ 爲 3.
Integer 陣列中儲存的 3 個 Integer 值, 由編號 1 到 3 的位址分別是:
addr.1 = 3 ， addr.2 = -567 ， addr.3 = 2

(2.) 若指定 MSG_ = '401 ， 3A ， +-345' ， 而 ARY_NO_ = 2 , 則會傳回 -1 , 因爲第 2 個值 ‘3A’ 與第 3 個值 ‘+-345’ 是不正確的 Integer 格式.

範例程式 :

1. wdemo_53.pia 放置於 W-8xx7 CD-ROM:\napdos\isagraf\wincon\demo\
2. wdemo_53.pia 放在 ftp://ftp.icpdas.com/pub/cd/wincon_isagraf/napdos/isagraf/wincon/demo/

MSGARY_R



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從 Message 陣列讀出 1 個字串

輸入參數 :

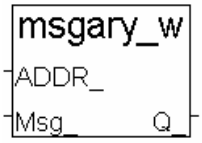
ADDR_	Integer	位置, 1 - 1024

傳回值 :

Msg_	Message	讀到的字串 (字串長度範圍 0 ~ 255)

範例: 請參考第 11 章 Wincon-8xx7 的 wdemo_06

MSGARY_W



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

寫入 1 個字串到 Message 陣列

輸入參數 :

ADDR_	Integer	位置, 1 - 1024
MSG_	Message	要寫入的的字串 (字串長度範圍 0 ~ 255)

傳回值 :

Q_	Boolean	True: 成功, False:失敗

範例: 請參考第 11 章 Wincon-8xx7 的 wdemo_06

PID_AL

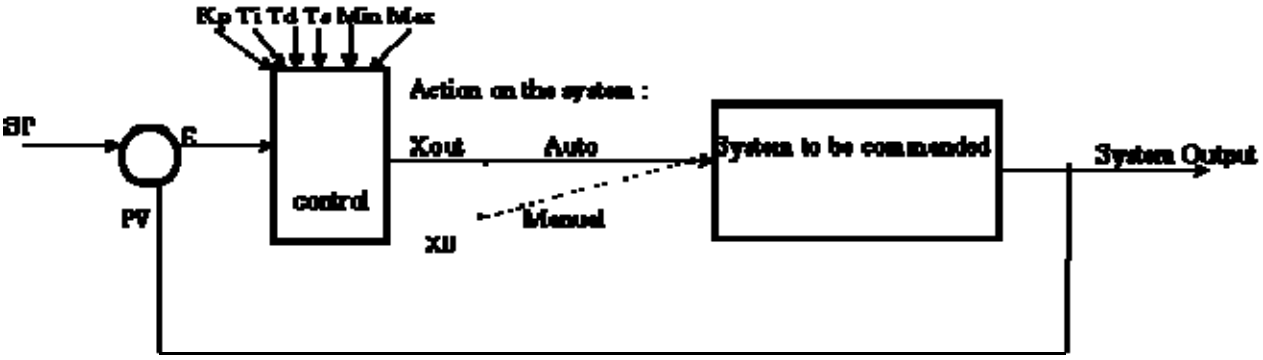
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6
型態 : C_Function Block
範例:

請參考第 11 章 - Demo_18, 與 ICP DAS 光碟 :
\Napdos\ISaGRAF\8000\English_Manu\PID_AL.Complex PID algorithm implementation.htm

PID_AL 原是由 CJ Internal 公司提供的 function, 以下爲 CJ 公司的說明(部分譯爲中文)

```
Author  : EDS
Product : ISaGRAF V3
Date    : 26 aug 96
File    : PID_AL.Complex PID algorithm implementation.htm
Subject : Complex PID algorithm implementation
Keywords: PID - PID_AL
```

PID 是一個進程的調節校正. 以反饋的觀念 將輸出(output)在實際數值與期望數值的差距間做調節.



PV 是輸出(Output). 入口處的 **e** 是 **SP-PV** 間計算而得的差距. **Control** 則計算 系統維持進程調節應做的動作.

System to command : 在模擬(仿真)作業中, 是個第二順位系統.

相對的, 在沒有模擬的系統, 則是如下圖:

PID 的主要各點描述如下:

SP : 是設定點, 設定期望的輸出值.

X0 (手動模式中, 開啓迴圈的 case) 是進入系統的未調節值.

Xout (已調節用以結束迴圈的 case) 是進入系統的已調節值.

CJ PID-Al 說明:

輸入參數:

Auto:自動或手動模式

Pv : 程序的輸出值

Sp : 設定爲設定點的輸出期望值

X0 : 調節值: 手動模式時, 輸出 PID 控制元等於 X0

Kp : 比例性常數

Ti : 整體性(積分式)時間常數(Integral time constant)

Td : 衍生性時間常數(derivative time constant)

Ts : 採樣週期

Min, Max : Xout 值接受的範圍

傳回值:

Xout : 指令

原型: PID(Auto,Pv,Sp,X0,Kp,Ti,Td,Ts, Min, Max);

Command:= PID.Xout;

注意: 自動模式的初始狀態必須設爲 false

- 完成式運算原型 :

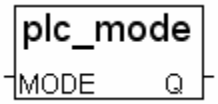
$$u(t) = Kp(e(t) + \frac{1}{Ti}\int_0^t e(t)d(t) + Td\frac{de(t)}{dt})$$ Theoretical Continue (理論持續) 方程式

$$u(k) = Kp(e(k) + \frac{Ts}{Ti}I(k) + \frac{Td}{Ts}[e(k) - e(k-1)])$$ Implemented Discret (完成離散)方程式

$$I(k) = I(k-1) + e(k)Ts$$

PLC_mode

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

改變 Wincon PLC 的速度

輸入參數 :

MODE_ :	Integer	Can be 0 , 1, 2, or 3 0: 快速模式, 預設值爲此模式, 最小 PLC scan 時間是 4 ms 1: 慢速模式, 最小 PLC scan 時間是 6 到 7 ms 2: 更慢速模式, 最小 PLC scan 時間是 9 到 11 ms 3 或其他值: 最慢速模式, 最小 PLC scan 時間是 19 到 21 ms

傳回值 :

Q_ :	Boolean	永遠傳回 TRUE

注意 :

1. 3.24B 版本才開始支援 "PLC_mode" 函式
2. 預設的模式是 "快速模式"
3. 使用者可以在第一次 PLC scan 就呼叫 "PLC_mode( )" function 來改變 PLC 速度.
4. 通常降慢 PLC 速度的原因是爲了增進其他與 ISaGRAF 程式同時執行的 HMI 程式的執行效率.
   例如: 一台 WinCon 中同時執行 Indusofr 和 ISaGRAF 程式的狀況.

範例 :

```
(* TMP 宣告爲布林內部變數 *)
(* INIT 宣告爲布林內部變數, 初始值 TRUE *)
if INIT then
  INIT := False ;  (* 只在第一回 PLC scan 中執行*)
  TMP := PLC_mode(2) ;  (* 設定 PLC 速度爲 2: 更慢速模式*)
end_if ;
```

下列各 PWM 函式皆爲 C Function, 詳細內容 與 範例 請參考手冊 第 3.7 節

PWM_DIS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 關閉 PWM 輸出

PWM_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 開啓 PWM 輸出.

PWM_EN2

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 開啓 PWM 輸出一給定數量的脈波

PWM_OFF

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 立刻將 parallel D/O 輸出爲 FALSE

PWM_ON

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 立刻將 parallel D/O 輸出爲 TRUE

PWM_SET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 動態更改 ON_, OFF_ & NUM_ 的設定

PWM_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 測試 PWM 的狀態

PWM_STS2

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

Function 取得"pwm_en2" 與 "pwm_en" 已輸出的 pulse 數量

R_MB_ADR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

使用 Modbus 位址讀取 boolean 或 integer 變數的值

輸入參數 :

TYPE_ :	Integer	0: 布林變數 , 1: 整數變數
ADR_ :	Integer	讀取的 Modbus 資料位址 , 有效範圍: I-8xx7 & I-7188xG 是 1~4095, Wincon 是 1~8191

傳回值 :

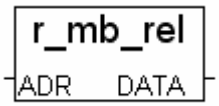
DATA_ :	Integer	整數值 (若 TYPE 是 boolean, 1 代表 True, 0 代表 False)

注意 :

1. 請使用 R_MB_REL function 讀取 "REAL" 變數.
2. 若沒有該 Modbus 位址定義的變數, 則傳回 0.
3. 若給予的 TYPE_ 爲整數 而對應的變數是 "Boolean" 型態, 則傳回值: 0:表 False , 1:表 True
4. 若給予的 TYPE_ 爲整數 而對應的變數是 "Real" , 則對應的 32-bit 會複製到 DATA_ 中. 您可使用 "int_real" 函式將 32-bit 整數對應到實數值. (最好使用 "R_MB_REL" 來讀取 “Real” 變數)
5. 若給予的 TYPE_ 爲布林 而對應的變數不是 "Boolean" 型態, 則傳回值: 0 .
6. 若長整數 (32-bit 整數) 透過 Modbus 通訊協定傳遞到 HMI, 則必需佔用 2 個 Modbus 位址, 請參考 ISaGRAF 進階使用手冊 第 4.2 節.

範例 : 請參考 ISaGRAF Projects/ Tools/ Libraries 說明

R_MB_REL



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

使用 Modbus 位址讀取 Real 變數的值

輸入參數 :

ADR_ :	Integer	讀取的 Modbus 資料位址 , 有效範圍: I-8xx7 & I-7188xG 是 1~4095, Wincon 是 1~8191

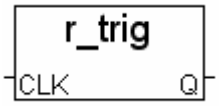
傳回值 :

DATA_ :	Real	讀取的實數值

注意 :

1. 請確認對應的變數型態爲 "Real". 若爲 "Integer", 請使用 "R_MB_ADR" function.
2. 若對應的變數型態不是 "Analog" (Real 或 integer), 則傳回 1.23E-20
3. 若沒有以該 Modbus 位址定義的變數, 則傳回 1.23E-20
4. 若實數值透過 Modbus 通訊協定傳遞到 HMI, 則必需佔用 2 個 Modbus 位址, 請參考 ISaGRAF 進階使用手冊 第 4.2 節.
5. 若該 Modbus 位址變數不是實數, 有可能會產生 Local Fault 編號 103, 請參考 10.6 節.

R_TRIG



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : Standard_Function

偵測布林變數的上昇變化

輸入參數 :

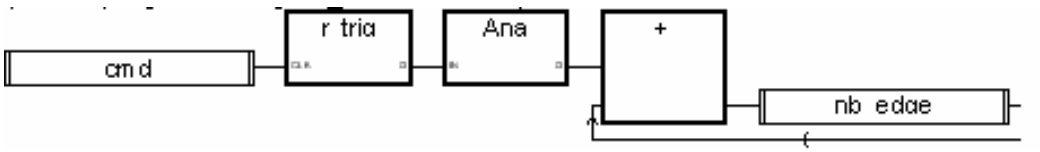
CLK	Boolean	任何布林變數

傳回值 :

Q	Boolean	TRUE : CLK 由 FALSE 上昇爲 TRUE FALSE : 其他所有狀況

範例 :

(* FBD 程式 *)



(* ST 相等式 : 假設 R_TRIG1 相當於 R_TRIG 功能方塊 *)

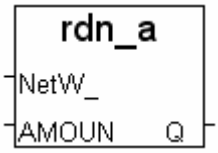
```
R_TRIG1(cmd);
nb_edge := ANA(R_TRIG1.Q) + nb_edge;
```

(* IL 相等式 : *)

```
LD      cmd
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

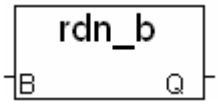
下列各 RDN 函式皆爲 Redundant 相關函式, 詳細內容 與 範例 請參考 ISaGRAF 手冊第 20 章

RDN_A



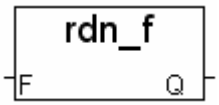
□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6
C_Function 設定 1 個變數陣列爲 Redundant 同步資料

RDN_B



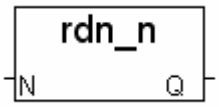
□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6
C_Function 設定 1 個 Boolean 變數爲 Redundant 同步資料

RDN_F



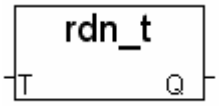
□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6
C_Function 設定 1 個 Real 變數爲 Redundant 同步資料

RDN_N



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6
C_Function 設定 1 個 Integer 變數爲 Redundant 同步資料

RDN_T



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6
C_Function 設定 1 個 Timer 變數爲 Redundant 同步資料

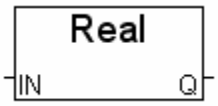
注意:

1. Redundant 系統的那 2 台 Wincon 內 Run 的 ISaGRAF project 是一模一樣的 project
2. 所有 I-7K 與 I-87K Remote IO 的功能塊必需放在 ISaGRAF project 的最上方
3. 使用以上的 RDN_ function 之前, 需在 IO connection 內連接 “rdn”
4. RDN_B, RDN_F, RDN_N, RDN_T, RDN_A 必需使用在一開機後的第 1 個 PLC Scan 內
5. 最大可傳遞的 同步資料量爲 6000 byte . 每個 Boolean 佔 1 byte, 每個 Real & Interger & Timer 各佔 4 byte.
6. 更多的訊息請參考第 20 章

範例: Wincon CD-ROM :\napdos\isagraf\wincon\demo\ 內的 Wdemo_18a & Wdemo_18b

REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : Standard_Function

轉換任何 1 個變數成爲 1 個實數

輸入參數 :

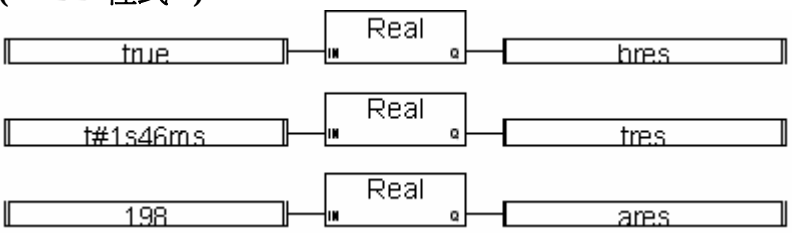
IN	Any	任何一個非實數的類比值 (非訊息)

傳回值 :

Q	Real	當 IN 爲 FALSE 傳回 0.0 /當 IN 爲 TRUE 傳回 1.0 計時器型態則傳回毫秒的數據 類比整數則傳回等值的實數

範例 :

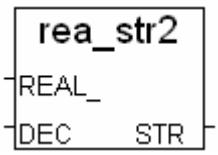
(* FBD 程式 *)




```
(* ST 相等式: *)
bres := REAL (true);              (* bres 爲 1.0 *)
tres := REAL (t#1s46ms);          (* tres 爲 1046.0 *)
ares := REAL (198);               (* ares 爲 198.0 *)
```

```
(* IL  相等式 : *)
LD      true
REAL
ST      bres
LD      t#1s46ms
REAL
ST      tres
LD      198
REAL
ST      ares
```

REA_STR2



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 Real 爲 1 個字串, 小數點後的位數固定

輸入參數 :

REAL_	Real	要被轉換的 Real 值
DEC_	Integer	小數點後的位數, 0 ~ 5

傳回值 :

STR_	Message	轉換後的字串(最長爲 13 個字元).

比如 DEC_ = 2

1.234 ---> '1.23'

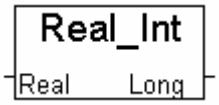
123456.0 ---> '1.23456.00'

0.00001234 ---> '0.00'

注意: "STR_REAL" 可用來轉換字串爲 Real 值.

範例: 請參考第 16 章 & demo_38, demo_39

REAL_INT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

對應 1 個實數成爲 1 個長整數

輸入參數 :

Real_	Real	要被對應的 real 值

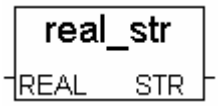
傳回值 :

Long_	Integer	對應完後的 integer 值

注意:

1. "Int_Real" 可用來對應 integer 成 real.
2. 如果只是要 轉換 一個實數爲 整數, 比如 -76.345 → -76 , 請用 ANA( )

REAL_STR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 Real 爲 1 個字串

輸入參數 :

REAL_	Real	要被轉換的 Real 值

傳回值 :

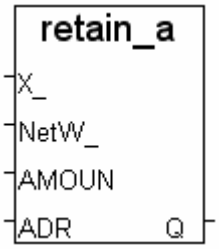
STR_	Message	轉換後的字串(最長爲 13 個字元).

1.234 ---> '1.234'

123456789.0 ---> '1.23457E+008'

0.00001234 ---> '1.234E-005'

RETAIN_A



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

設定 “變數陣列” 爲 “可保留變數” - 以第 1 個元素的網路位址編號來設定

* 請參考第 10 章與第 2.6 節
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

X_ :	Message	'B' 或 'b' : 布林, 'N' 或 'n' : 整數, 'F' 或 'f' : 實數, 'T' 或 't' : 計時器
NetW_ :	Integer	“變數陣列” 第一個元素的網路位址編號. Target 1/2/3 : 1~4095 , Target 4: 1~8191
Amount_ :	Integer	“變數陣列” 的大小, 有效範圍: 1~255, 例如: CNT[0..15] 大小爲 16, ABC[0..7] 大小爲 8.
ADR_ :	Integer	設定爲變數要儲存的位址. Target 1/2/3 : 布林和計時器爲 1~256, 整數和實數爲 1~1024 . Target 4 : 布林和計時器爲 1~1024 , 整數和實數爲 1~4096 .

傳回值 :

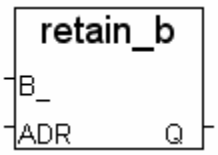
Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   [DEBUG]
   arrays=1
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

RETAIN_B

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定布林變數爲 “可保留變數”

* 請參考第 10 章
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

B_ :	Boolean	B_ 必須爲布林變數, 不能是常數.
ADR_ :	Integer	設定此布林變數的保留位址. Target 1 和 2 和 3 : 1 ~ 256; Target 4: 1 ~ 1024

傳回值 :

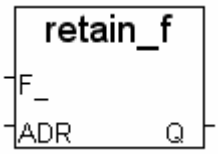
Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   ```
   [DEBUG]
   arrays=1
   ```
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

RETAIN_F

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定實數變數爲 “可保留變數”

* 請參考第 10 章
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

F_ :	Real	F_ 必須爲實數變數, 不能是常數.
ADR_ :	Integer	設定此實數變數的保留位址. Target 1 和 2 和 3 : 1 ~ 1024; Target 4: 1 ~ 4096

傳回值 :

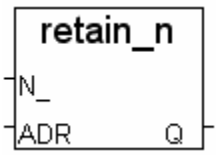
Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   [DEBUG]
   arrays=1
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

RETAIN_N

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定整數變數爲“可保留變數”

* 請參考第 10 章
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

N_ :	Integer	N_ 必須爲整數變數, 不能是常數.
ADR_ :	Integer	設定此整數變數的保留位址. Target 1 和 2 和 3 : 1 ~ 1024; Target 4: 1 ~ 4096

傳回值 :

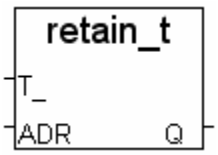
Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   [DEBUG]
   arrays=1
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

RETAIN_T

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定計時器變數爲“可保留變數”

* 請參考第 10 章
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

T_ :	Timer	T_ 必須爲計時器變數, 不能是常數.
ADR_ :	Integer	設定此計時器變數的保留位址. Target 1 和 2 和 3 : 1 ~ 256; Target 4: 1 ~ 1024

傳回值 :

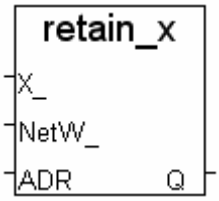
Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   [DEBUG]
   arrays=1
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

RETAIN_X

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

設定一個變數爲 “可保留變數” – 以變數的網路位址編號來設定

* 請參考第 10 章
* 各種 ISaGRAF 控制器於下列版本起 開始適用此 ”可保留變數” 設定方式:
  * Target 1: I-8417/8817/8437/8837 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.13
  * Target 2: I-7188EG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.11
  * Target 3: I-7188XG + X607 或 X608 (保留變數), 自驅動程式版本 Ver 2.09
  * Target 4: W-8xx7/8xx6 + S-256 或 S-512 (保留變數), 自驅動程式版本 Ver 3.26

輸入參數 :

X_ :	Message	'B' 或 'b' : 布林, 'N' 或 'n' : 整數, 'F' 或 'f' : 實數, 'T' 或 't' : 計時器
NetW_ :	Integer	對應變數的網路位址編號. Target 1/2/3 : 1 ~ 4095; Target 4: 1 ~ 8191
ADR_ :	Integer	設定爲變數保留的位址.. Target 1/2/3 : 1 ~ 1024; Target 4: 1 ~ 4096

傳回值 :

Q_ :	Boolean	Ok 傳回 True, 錯誤則傳回 False (例如 : 找不到保留記憶體)

注意 :

1. 使用 Retain_X, Retain_A, Retain_N , Retain_B , Retain_F 和 Retain_T 函式之前, 請確認 ISaGRAF IO 連結視窗中 "IO complex equipment" 欄位的下列各設備已正確連接.
   a. I-7188EG/XG : "X607_608"
   b. I-8417/8817/8437/8837 和 W-8037/8337/8737 : "S256_512"
2. 要在 ISaGRAF 3.4 (或 3.5) 版本中宣告 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下 "isa.ini" 檔案的最頂端加入 2 行程式. 之後, 開啓 ISaGRAF 工作平台, 在 Dictonary 的宣告視窗新增的 “DIM” 欄位內指定陣列個數. 請在 c:\isawin\exe\isa.ini 檔案頂端, 加入下列 2 行 :
   [DEBUG]
   arrays=1
3. 使用 "Retain_A" 之前, 請在 ISaGRAF dictionary 視窗中 設定"變數陣列"的 “Network Address” 欄位內 指定一個網路位址編號. 請務必<u>不要</u>勾選 “Retain” 欄位. 例如, 指定 整數 ”變數陣列” CNT[0..7] 的 “Network Address”.= 1A (16 進位制. 十進位制是 26) .
4. 請務必確認可保留記憶體 (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) 正確成功的安插在控制器上.

S_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從電池保持式 SRAM 讀出 1 個 Boolean 值

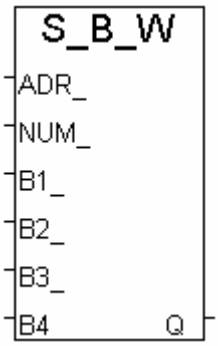
輸入參數 :

ADR_	Integer	從那個位置讀出, 1 個 Boolean 佔有 1 個 Byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

傳回值 :

BOO_	Boolean	讀到的 Boolean 值, 0=FALSE, 非 0 = TRUE

S_B_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

寫入 Boolean 值到電池保持式 SRAM

輸入參數 :

ADR_ :	Integer	從那個位置開始寫, 1 個 Boolean 佔有 1 個 byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_ :	Integer	要寫幾個 Boolean, 0 ~ 4
B1_~B4_ :	Boolean	要寫入的 Boolean 值

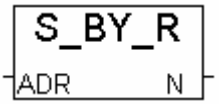
傳回值 :

Q_ :	Boolean	正確: TRUE, 失敗: FALSE 寫入的值爲 FALSE=0, TRUE=1

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_BY_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從電池保持式 SRAM 讀出 1 個 Byte 值

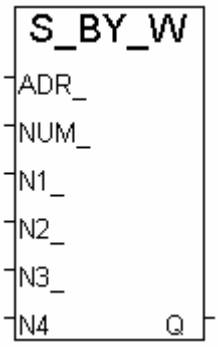
輸入參數 :

ADR_ :	Integer	從那個位置讀出, 1 個 Byte 佔有 1 個 Byte.
		S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

傳回值 :

N_ :	Integer	讀到的 Byte 值, 0 ~ 255

S_BY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

寫入 Byte 值到電池保持式 SRAM

輸入參數 :

ADR_ :	Integer	從那個位置開始寫, 1 個 Byte 佔有 1 個 byte.
		S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_ :	Integer	要寫幾個 Byte, 0 ~ 4
N1_~N4_ :	Boolean	要寫入的 Byte 值, 0~255

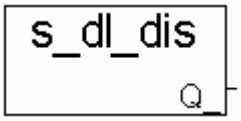
傳回值 :

Q_ :	Boolean	正確: TRUE, 失敗: FALSE

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_DL_DIS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

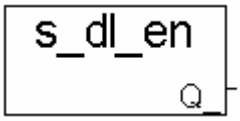
型態 : C_Function

關閉下載的授權, 如此 PC 無法下載資料到 電池保持式 SRAM 內

傳回值 :

Q_ : Boolean 正確: TRUE, 失敗: FALSE

S_DL_EN



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

開啓下載的授權, 如此 PC 可以下載資料到 電池保持式 SRAM 內

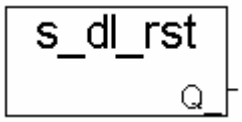
傳回值 :

Q_ : Boolean 正確: TRUE, 失敗: FALSE

注意:

控制器開機後的預設值爲 “關閉”, 程式必須呼叫 “ S_DL_EN”, PC 才能下載資料進去 SRAM 內

S_DL_RST



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

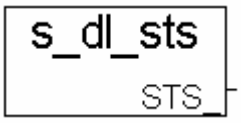
型態 : C_Function

重置 SRAM 下載狀態 爲 “-1: 無動作”

傳回值 :

Q_ : Boolean 正確: TRUE, 失敗: FALSE

S_DL_STS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

取得 PC 對 SRAM 的 下載狀態

傳回值 :

STS_ : Integer

-1: 無動作
1: PC 正在下載資料到 SRAM
2: 下載完畢
3: PC 有要求下載資料, 但資料從未下載完成 (可能通訊出了問題)

可呼叫 S_DL_RST 將 下載狀態 設爲 -1 (無動作)

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_FL_AVL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

設定 SRAM 檔案的目前可取得資料的所在位置

傳入參數 :

ID_ :	Integer	檔案編號 (1 ~ 8)
HEAD_ :	Integer	起頭位置的 Byte 編號
TAIL_ :	Integer	結尾位置的 Byte 編號

(HEAD_, TAIL_) 必需坐落於 檔案配置區 內(請參考 "S_FL_INI"), 否則 Q_ 回傳 FALSE

S256: 1 - 249856 (1 - 16#3D000)
S512: 1 - 512000 (1 - 16#7D000)
X607: 1 - 118784 (1 - 16#1D000)
X608: 1 - 512000 (1 - 16#7D000)

例如 :

某 SRAM 檔案的 檔案配置區爲 1 ~ 20000 , 表示此檔最多可存 20000 個 byte.

1. 若任何一個 HEAD_ 及 TAIL_ 爲 -1, 表示此檔目前無資料.
2. 若 HEAD_=1, TAIL_=1000, 表示此檔目前 的資料 位於 1 ~ 1000. 共 1000 個 byte
3. 若 HEAD_=10001, TAIL_=5000, 表示此檔目前 的資料 開始於 10001 ~ 20000, 緊接著由 1 ~ 5000, 共 15000 個 byte.
4. 若 HEAD_=1000, TAIL_=1000, 表示此檔目前無資料.

回傳值:

Q_ :	Boolean	TRUE: 正確 , FALSE: 失敗

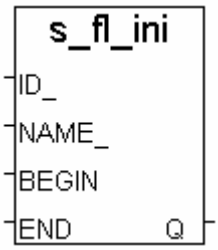
注意: S_FL_INI 需在使用 S_FL_AVL 之前先被呼叫

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_FL_INI

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function

設定 SRAM 檔案的 檔案配置區

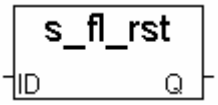
傳入參數 :

ID_ :	Integer	檔案編號 (1 ~ 8)
NAME_ :	Message	檔案名稱, 名稱最多 8 個字+延伸檔名 3 個字. 例如, "data1.txt", "A1234567.bin". 合法的字爲 : A ~ Z , a ~ z , _ , 0 ~ 9, 且第 1 個字需爲 A ~ Z 或 a ~ z
BEGIN_ :	Integer	檔案配置區的 起頭位置的 Byte 編號. BEGIN_ 需小於 END_
END_ :	Integer	檔案配置區的 結尾位置的 Byte 編號. BEGIN_ 需小於 END_ S256: 1 ~ 249,856 S512: 1 ~ 512,000 X607: 1 ~ 118,784 X608: 1 ~ 512,000 例, BEGIN_=101, END_=5000 : 此 SRAM 檔案的檔案配置區位於 Byte 編號 101 ~ 5000

傳回值:

Q_ :	Boolean	TRUE: 正確, FALSE: 失敗

S_FL_RST



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

重置 SRAM 檔案狀態爲 "PC 未上載資料過"

傳入參數 :

ID_ :	Integer	檔案編號 (1 ~ 8)

回傳值:

Q_ :	Boolean	TRUE: 正確, FALSE: 失敗

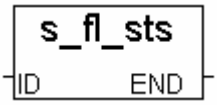
注意:

1. S_FL_INI 需先被呼叫過.
2. S_FL_RST 可用來重置 狀態 爲 -1 ("PC 未上載資料過")

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_FL_STS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

取得 SRAM 檔案的狀態, PC 上載資料的結尾 byte 編號

傳入參數 :

ID_ :	Integer	檔案編號 (1 ~ 8)

回傳值:

END_ :	Integer	PC 上載資料的結尾 Byte 編號 若 PC 未上載資料過 : -1 S256: 1 ~ 249,856 S512: 1 ~ 512,000 X607: 1 ~ 118,784 X608: 1 ~ 512,000

例如,

某 SRAM 檔案的檔案配置區爲 1~20000, 而此檔案目前可取得資料位於 1001~10000

1. 若 END_ 爲 -1, 表示 PC 未上載資料
2. 若 END_ 爲 10000, (通常此值會等於檔案目前可取得資料的結尾位置的 Byte 編號), 表示 PC 已經上載取得 1001 ~ 10000 內的資料
3. 若 END_ 爲 8000, 表示 PC 已經上載取得 1001 ~ 8000 內的資料

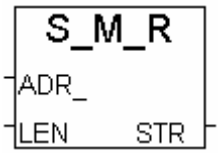
注意:

1. S_FL_INI 需先被呼叫過.
2. S_FL_RST 可用來重置 狀態 爲 -1 ("PC 未上載資料過")

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_M_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從電池保持式 SRAM 讀出 1 個 字串

傳入參數 :

ADR_ :	Integer	從那個位置讀出. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_ :	Integer	要讀取的字串的長度, 0 ~ 255

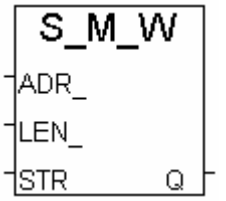
傳回值 :

STR_ :	Message	讀到的字串

例如 : 若存於 SRAM 內的資料爲 16#31, 16#32, 16#33, 16#34, 16#35, **0**, 16#37, 16#38, ...

LEN_=0 ----> STR_= '' (空字串)
LEN_=3 ----> STR_= '123'
LEN_=5 ----> STR_= '12345'
LEN_=6 ----> STR_= '12345'
LEN_=7 ----> STR_= '12345'
LEN_=100 ----> STR_= '12345'

S_M_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

寫入 字串 到電池保持式 SRAM

傳入參數 :

ADR_ :	Integer	從那個位置開始寫. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_ :	Integer	要寫的字串的長度, 0 ~ 255
STR_ :	Message	要寫的字串.

例如 :

LEN_=0 , STR='12345' ----> 沒有資料寫入
LEN_=1 , STR='12345' ----> 16#31 (寫入 1 個 byte)
LEN_=3 , STR='12345' ----> 16#31, 16#32, 16#33 (寫入 3 個 byte)
LEN_=7 , STR='12345' ----> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0 (寫入 7 個 byte)
LEN_=100 , STR='12345' --> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0, 0, … (寫入 100 個 byte)

傳回值 :

Q_ :	Boolean	正確: TRUE, 失敗: FALSE

範例 : demo_40, demo_41, demo_42, demo_44

* 請參考第 10.3 節.

S_MB_ADR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

指定連續的或跳一號的 Modbus 網路位址編號給 “變數陣列” (參閱 2.6.1 節)

控制器分類 :

Target 1: I-8417/8817/8437/8837 , I-7188EG , I-7188XG
Target 2: Wincon-8xx7

傳入參數 :

ADDR_ :	Integer	Target 1: 1-4095, Target 2: 1-8191. 指定到 Dictionary 視窗 "變數陣列" 欄位的 Modbus 起始網路位址編號.
AMOUNT_ :	Integer	有效範圍 : 1- 255, “變數陣列” 的大小, 例如: CNT[0..15] 大小爲 16, ABC[0..7] 大小爲 8.
INTERVAL_ :	Integer	0 : 連續的 modbus 位址. (例如, 10, 11, 12, ...) 1 : 跳號的 modbus 位址. (例如, 10, 12, 14, ...)

傳回值 :

Q_ :	Boolean	True: Ok, False: ADDR_ 或 AMOUNT_ 或 INTERVAL_ 錯誤

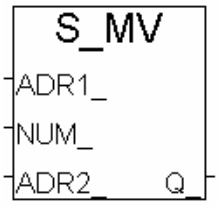
注意:

1. 此函式須在第一個 PLC scan 內呼叫, 第二次之後呼叫無效.
2. Modbus 起始網路位址編號必須在 ISaGRAF Dictionary 視窗 "變數陣列" 欄位裡指定. 例如: 指定起始網路位址編號 21 到 L[0..9] 的 ”變數陣列” .
3. 宣告 ISaGRAF 3.4 (或 3.5) 版本的 "變數陣列", 請在 ISaGRAF 子目錄 "C:\ISAWIN\EXE\" 下的 "isa.ini" 檔案頂端加入 2 行程式, 之後, 開啓 ISaGRAF 工作平台, 在 Dictionary 宣告視窗新增的 “DIM” 欄位指定. 請在 c:\isawin\exe\isa.ini 檔案頂端加入下列 2 行,
   [DEBUG]
   arrays=1
4. 當您在幫控制器程式除錯時, 要觀察或控制 “變數陣列” 的值, 請使用 “Spy list” 並插入陣列變數的名稱. 例如: 插入 L[0], L[1]. 請參考 ISaGRAF 進階使用手冊 第 9.12 節 "Spy list" 部份
5. 請務必小心宣告與使用 “變數陣列”. 僅宣告需要的陣列大小, 宣告太大會無謂的佔據記憶體. 並注意不要使用大於您陣列大小的指標, 例如: 若您宣告變數陣列爲 CNT[0..9], 千萬不可使用超過您宣告大小的指標 (如下列程式碼), 否則將導致程式執行錯誤. Ex:

```
For index := 0 to 10 do
  A := A + CNT[index] ;
End_For ;
(* 本範例 CNT[10] 並不存在 *)
```

6. 請小心指定 Modbus 網路位址, 不可與其他變數相衝突.
7. 手冊 : 有關控制器 冗於備援 與 變數陣列 的詳細資料, 請參考下列手冊或網址中的相關文章 \redundancy.pdf (冗於備援) 及 Variable Array (變數陣列).
   1. Wincon-8xx7 CD-ROM: \napdos\isagraf\wincon\english_manu\
   2. ftp://ftp.icpdas.com/pub/cd/wincon_isagraf/napdos/isagraf/wincon/english_manu/

S_MV



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

在 SRAM 內複製資料

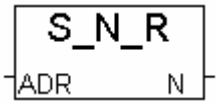
輸入參數 :

ADR1_	Integer	目的地的起頭位置 S256: 1 - 249856 (1 - 16#3D000) S512: 1 - 512000 (1 - 16#7D000) X607: 1 - 118784 (1 - 16#1D000) X608: 1 - 512000 (1 - 16#7D000)
NUM_	Integer	要複製多少個 byte, 0 - 512,000
ADR2_	Integer	從那個位置複製過來

傳回值 :

Q_	Boolean	正確: TRUE, 失敗: FALSE

S_N_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從電池保持式 SRAM 讀出 1 個 Integer 值

輸入參數 :

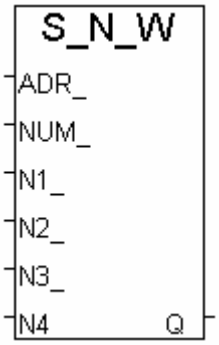
ADR_	Integer	從那個位置讀出, 1 個 Integer 佔有 4 個 Byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

傳回值 :

N_	Integer	讀到的 Integer 值

注意 : 儲存在電池保持式 SRAM 內的 Integer 格式爲 [Lowest byte] [2nd byte] [3rd byte] [High byte], 例如 16#01020304, 會存入爲 [04] [03] [02] [01]

S_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

寫入 Integer 值到電池保持式 SRAM

輸入參數 :

ADR_	Integer	從那個位置開始寫, 1 個 Integer 佔有 4 個 byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_	Integer	要寫幾個 Integer, 0 ~ 4
N1_~N4_	Integer	要寫入的 Integer 值

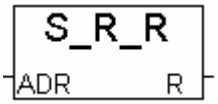
注意 : 儲存在電池保持式 SRAM 內的 Integer 格式爲 [Lowest byte] [2nd byte] [3rd byte] [High byte], 例如 16#01020304, 會存入爲 [04] [03] [02] [01]

傳回值 :

Q_	Boolean	正確: TRUE, 失敗: FALSE

S_R_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function

從電池保持式 SRAM 讀出 1 個 Real 值

輸入參數 :

ADR_	Integer	讀那個位置, 1 個 Real 佔有 4 個 byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

傳回值 :

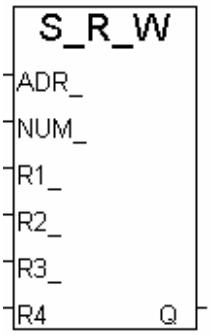
R_	Real	讀到的 Real 值

注意 :

1. 儲存在電池保持式 SRAM 內的 Real 格式爲 [Lowest byte] [2nd byte] [3rd byte] [High byte], 例如: 1.23 , 會存入爲 16#A4 , 16#70 , 16#9D , 16#3F
2. 若存放於電池保持是 SRAM 內的資料不是 Real 型態, 使用 S_R_R 讀出時, 有可能會產生 Local Fault 編號 102 (請參閱第 10.6 節)

S_R_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function

寫入 Real 值到電池保持式 SRAM

輸入參數 :

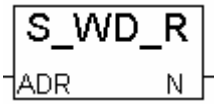
ADR_	Integer	從那個位置開始寫, 1 個 Real 佔有 4 個 byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_	Integer	要寫幾個 Real, 0 ~ 4
R1_~R4_	Real	要寫入的 Real 值

注意 : 儲存在電池保持式 SRAM 內的 Real 格式爲 [Lowest byte] [2nd byte] [3rd byte] [High byte], 例如 1.23 , 會存入爲 16#A4 , 16#70 , 16#9D , 16#3F

傳回值 :

Q_	Boolean	正確: TRUE, 失敗: FALSE

S_WD_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

從電池保持式 SRAM 讀出 1 個 Word 值

輸入參數 :

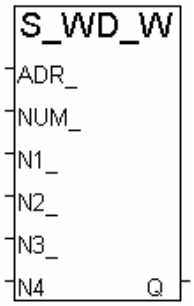
ADR_	Integer	從那個位置讀出, 1 個 Word 佔有 2 個 Byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

傳回值 :

N_	Integer	讀到的 Word 值, -32768 ~ +32767

注意 : 儲存在電池保持式 SRAM 內的 Word 格式爲 [Low byte] [High byte],
例如 16#0102, 會存入爲 [02] [01]

S_WD_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

寫入 Word 值到電池保持式 SRAM

輸入參數 :

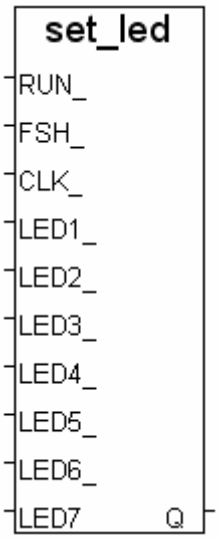
ADR_	Integer	從那個位置開始寫, 1 個 Word 佔有 2 個 byte. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_	Integer	要寫幾個 Word, 0 ~ 4
N1_~N4_	Integer	要寫入的 Word 值, -32768 ~ 32767

注意: 儲存在電池保持式 SRAM 內的寫入的 Word 格式爲 [Low byte] [High byte],
例如 16#0102, 會存入爲 [02] [01]

傳回值 :

Q_	Boolean	正確: TRUE, 失敗: FALSE

SET_LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

顯示訊息到 S-MMI

輸入參數 :

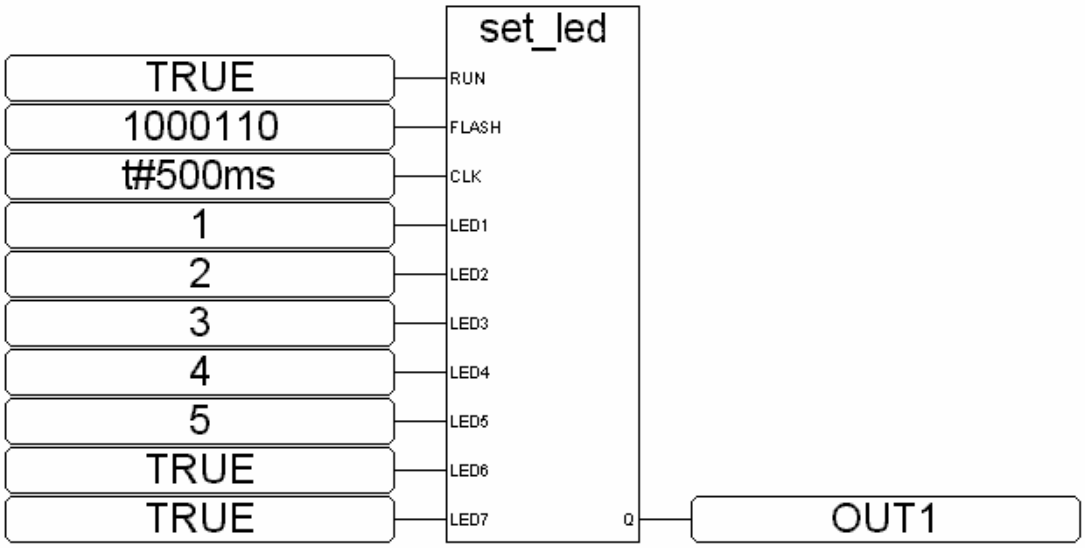
RUN_	Boolean	設爲 TRUE 才動作
FLASH_	Integer	相對應的位置設爲 1 則閃爍. 例如, 設爲 11 (0000011), 則第 6 及第 7 的位置會閃爍. 若設爲 100001 (0100001), 則第 2 及第 7 的位置會閃爍
CLK_	Timer	閃爍的週期時間
LED1_	Integer	位置 1 要顯示的符號
LED2_	Integer	位置 2 要顯示的符號
LED3_	Integer	位置 3 要顯示的符號
LED4_	Integer	位置 4 要顯示的符號
LED5_	Integer	位置 5 要顯示的符號
LED6_	Boolean	設爲 TRUE 則顯示位置 6
LED7_	Boolean	設爲 TRUE 則顯示位置 7

傳回值 :

Q_	Boolean	無作用

***** 請參考附錄 A.3 來查閱符號表

範例:

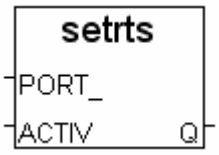


ST 相等式:

```
TMP := SET_LED(TRUE,1000110,t#500ms,1,2,3,4,5,TRUE,TRUE);
            (* TMP 需宣告爲 boolean 型態 *)
```

SETRTS

□ I-8417/8817 □ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6



型態 : C_Function

設定 連接埠的 RTS 訊號, 適用 COM 號 : 3 ~ 5

輸入參數 :

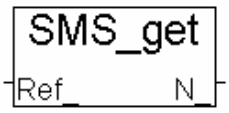
PORT_:	Integer	要設定的 COM 號. 3:COM3, 4:COM4, 5:COM5
ACTIVE_ :	Boolean	TRUE : 設定 RTS 作用中, FALSE : 設定 RTS 不作用

傳回值 :

Q_ :	Boolean	TRUE : OK , FALSE : 失敗

SMS_GET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

取得收到簡訊的日期與時間 (請參閱第 17 章)

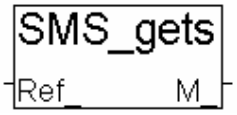
輸入參數 :

REF_	Integer	取得什麼？ 1 ~ 7 1: 年, (N_ = 2000 ~ 2099) 2: 月, (N_ = 1 ~ 12) 3: 日, (N_ = 1 ~ 31) 4: 星期幾, (N_ = 1 ~ 7, 星期天爲 7) 5: 時, (N_ = 0 ~ 23) 6: 分, (N_ = 0 ~ 59) 7: 秒, (N_ = 0 ~ 59) 其他: 回傳 N_=-1

傳回值 :

N_	Integer	回傳對應的資料, 若回傳-1, 可能是 沒有收到簡訊 或 REF_不在 1~7 的範圍內

SMS_GETS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

取得簡訊內容及其它訊息 (請參閱第 17 章)

輸入參數 :

REF_	Integer	取得什麼？ 1 ~ 3 1: 簡訊內容 2: 發送者的電話號碼 3: 日期與時間 (字串格式) 其它: 回傳 M_= 'error'

傳回值 :

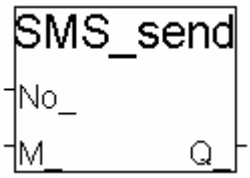
M_	Message	回傳對應的資料, 若回傳 'error', 可能是沒有收到簡訊 或 REF_ 不在 1 ~3 的範圍內

注意:

1. SMS_gets & SMS_get 可用來取得簡訊的相關資料
2. 當呼叫過 SMS_gets(1) 之後 (取得簡訊內容), 簡訊的 buffer 會清除爲 "無簡訊". 因此請在使用 SMS_gets(1)之前先將其它需要的資料先讀出, 如 SMS_get(1~7) 及 SMS_gets(2) & SMS_gets(3)

範例: demo_43, demo_43a

SMS_SEND



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

驅動 工控器 發送 1 封簡訊 (請參閱第 17 章)

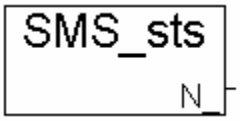
輸入參數 :

No_	Message	發送給誰 , 例如 '+886920119135', 最大長度爲 31 字
M_	Message	要發送的簡訊內容

傳回值 :

Q_	Boolean	True: 正確 , False: 錯誤的號碼或發送失敗

SMS_STS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

取得 簡訊發送狀態 (請參閱第 17 章)

傳回值 :

N_	Integer	狀態碼 0: 等待中, 尚未有發送命令進來 1: 忙碌中. (有 1 封簡訊正在發) 21: 簡訊發送成功 -1: SMS 系統錯誤 (請檢查 GSM Modem 與 SIM 卡) -2: Timeout 或 不明的原因

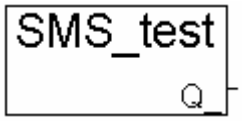
注意:

1. 請先使用 SMS_sts 取得 “簡訊發送狀態” 後, 再使用 SMS_send. 只有在狀態不爲 ”1: 忙碌中” 時才有辦法發送
2. 呼叫 SMS_send 後會設定 "簡訊發送狀態" 爲 "1: 忙碌中", 之後,隨著時間過去, 會設定成適當的值. 如 21 表示 簡訊發送成功, 或-1 或 -2 表示 有錯誤發生

範例: demo_43, demo_43a

SMS_TEST

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

測試有無收到簡訊 (請參閱第 17 章)

傳回值 :

Q_	Boolean	TRUE: 有收到簡訊, FALSE: 無

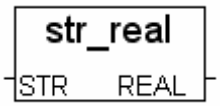
注意:

1. SMS_gets & SMS_get 可用來取得簡訊的相關資料
2. 當呼叫過 SMS_gets(1) 之後 (取得簡訊內容), 簡訊的 buffer 會清除爲 "無簡訊". 因此請在使用 SMS_gets(1)之前先將其它需要的資料先讀出, 如 SMS_get(1~7) 及 SMS_gets(2) & SMS_gets(3)

範例 : demo_43 , demo_43a

STR_REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

轉換字串爲 1 個 Real 值

輸入參數 :

STR_	Message	要被轉換的字串, 例, '-0.2345' , ' +2.13E10' , ' 15.2345E-2'

傳回值 :

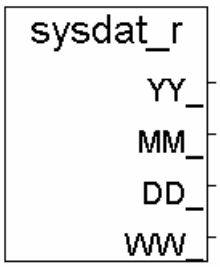
REAL_	Real	轉換後的 Real 值. 若該值爲 1.23E-20 , 表示字串格式錯誤 例, 若 STR_=' 123.AB' 或 '23-45.17' 或 '1.2.345' 則 REAL_ 爲 1.23E-20

注意:

"REAL_STR" 與 "REA_STR2" 可用來轉換 Real 值爲字串.

範例: 請參考第 16 章 & demo_38, demo_39

SYSDAT_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

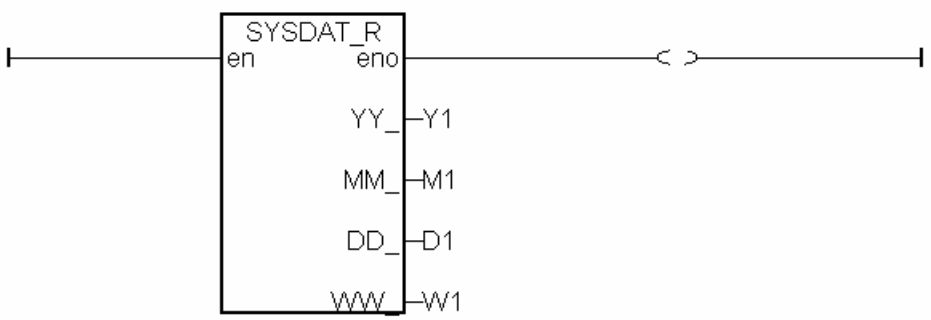
讀取 年, 月, 日 及 星期幾

傳回值 :

YY_	Integer	年, 如 2002, 2003, 2010
MM_	Integer	月, 1 ~ 12
DD_	Integer	日, 1 ~ 31
WW_	Integer	星期幾, 1 ~ 7 , 7 爲星期日

範例: 請參考 demo_03.

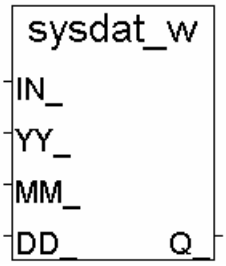
Y1, M1, D1 及 W1 需宣告爲 integer



ST 相等式:

```
DAT_R1( );                  (* 呼叫 DAT_R1 *)
Y1 := DAT_R1.YY_ ;          (* 取得年 *)
M1 := DAT_R1.MM_ ;          (* 取得月 *)
D1 := DAT_R1.DD_ ;          (* 取得日 *)
W1 := DAT_R1.WW_ ;          (* 取得星期幾 *)
(* DAT_R1 需宣告爲 SYSDAT_R 的 FB instance *)
```

SYSDAT_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

設定 年, 月, 日

輸入參數 :

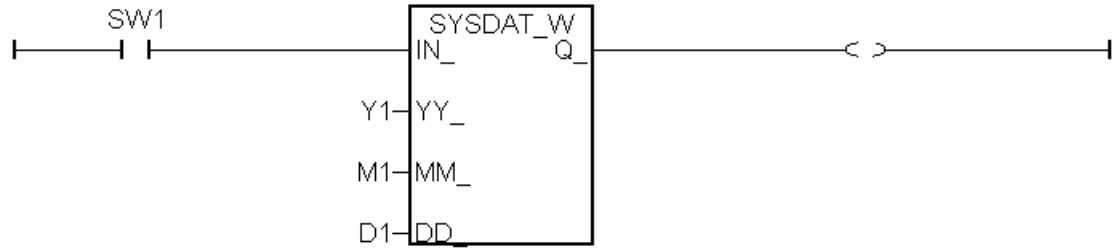
IN_	Boolean	當由 FALSE 上升爲 TRUE 時才設定一次
YY_	Integer	年, 如 2002, 2003, 2010
MM_	Integer	月, 1 ~ 12
DD_	Integer	日, 1 ~ 31

傳回值 :

Q_	Boolean	正確回傳 TRUE

範例: 請參考 demo_03.

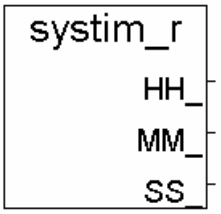
SW1 宣告爲 boolean. Y1, M1, D1 宣告爲 integer.



ST 相等式:

```
DAT_W1( SW1, Y1, M1, D1);        (* 呼叫 DAT_W1 *)
OUT1 := DAT_W1.Q_ ;              (* 取得回傳值 Q_ *)
(* DAT_W1 需宣告爲 SYSDAT_W 的 FB instance *)
(* OUT1 宣告爲 boolean *)
```

SYSTIM_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

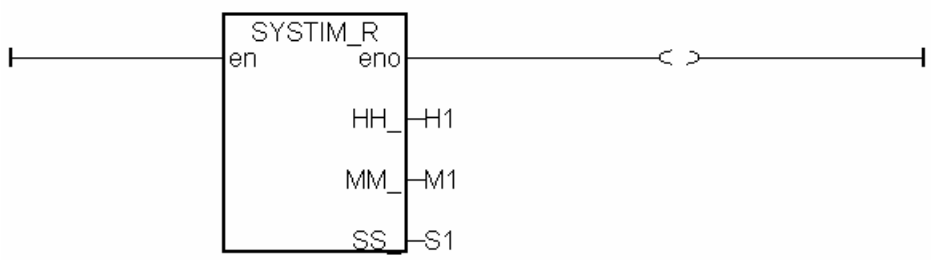
讀取 時, 分 及 秒

傳回值 :

HH_	Integer	時, 0 ~ 23
MM_	Integer	分, 0 ~ 59
SS_	Integer	秒, 0 ~ 59

範例: 請參考 demo_03 及 demo_15b.

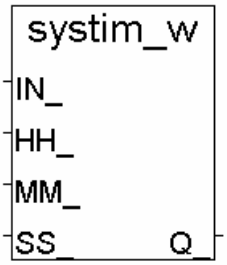
H1, M1 及 S1 宣告爲 integer



ST 相等式:

```
(* TIM_R1 宣告爲 SYSTIM_R 的 FB instance *)
TIM_R1( );                          (* 呼叫 TIM_R1 *)
H1 := TIM_R1.HH_ ;                  (* 取得時 *)
M1 := TIM_R1.MM_ ;                  (* 取得分 *)
S1 := TIM_R1.SS_ ;                  (* 取得秒 *)
```

SYSTIM_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

設定 時, 分 及 秒

輸入參數 :

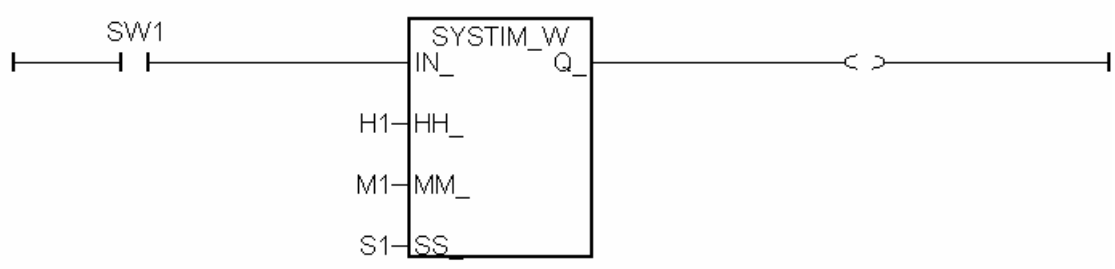
IN_	Boolean	當由 FALSE 上升爲 TRUE 時才設定一次
HH_	Integer	時, 0 ~ 23
MM_	Integer	分, 0 ~ 59
SS_	Integer	秒, 0 ~ 59

傳回值 :

Q_	Boolean	正確回傳 TRUE

範例: 請參考 demo_03

SW1 宣告爲 boolean. H1, M1, S1 宣告爲 integer.

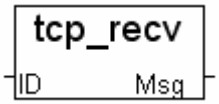


ST 相等式:

```
TIM_W1( Sw1,2000,7,5);          (* 呼叫 TIM_W1 *)
OUT1 := TIM_W1.Q_ ;             (* 取得回傳值 Q_ *)
(* TIM_W1 宣告爲 SYSTIM_W 的 FB instance *)
(* OUT1 宣告爲 boolean *)
```

TCP_RECV

□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6



型態 : C_Function

TCP Client 端接收遠端 PC 或 TCP/IP Server 傳來的 message (透過乙太網路)
(請參閱第 19.3 節)

輸入參數 :

ID_ :	Integer	對應的 "Tcp_Clie" 的連結編號, 可爲 1 到 4. 對應的 "IP 位址" 和 "埠號" 請在 IO 連結設定視窗選 ”設備” 的 "Tcp_clie" 來定義

傳回值 :

Msg_:	Message	接收的訊息. 若 Msg_ = '' (空訊息), 表示沒有訊息進來.

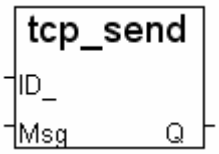
注意 :

1. 在使用"tcp_recv" 和 "tcp_send" 之前, 請先連接 IO connection 視窗中的 "tcp_clie" (TCP_Client)
2. 接收的暫存器大小爲 4096 bytes. 若暫存器滿了, 最先接收到的訊息會被清除, 把空間留給新接收的訊息.

範例 : Wdemo_32 和 Wdemo_33 (Wincon CD_ROM:\napdos\isagraf\wincon\demo\)

TCP_SEND

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8xx7/8xx6 (ver 3.30)



型態 : C_Function

TCP Client 端傳送 message 到遠端 PC 或 TCP/IP Server (透過乙太網路)
(請參閱第 19.3 節)

輸入參數 :

ID_ :	Integer	對應的 "Tcp_Clie" 的連結編號, 可爲 1 到 4. 對應的 "IP 位址" 和 "埠號" 請在 IO 連結設定視窗選 ”設備” 的 "Tcp_clie" 來定義
Msg_ :	Message	傳送的訊息

傳回值 :

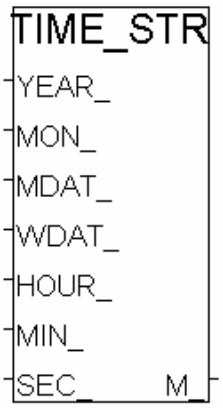
Q_ :	Boolean	True: 傳送 OK, False: 參數錯誤(如, ID_ 設成 8) 或 IO connection 視窗中對應連結的 "Tcp_clie" 沒有設定成功.

注意 :

1. 在使用 "tcp_send" 和 "tcp_recv" 之前, 請先連接 IO connection 視窗中的 "tcp_clie" (TCP_Client)
2. 傳送的暫存器大小爲 4096 byte, 這表示一次 PLC sacn 最多可傳送 4096 byte 到遠端 IP. 若暫存器滿了, 最先接收到的訊息會被清除, 把空間留給新的 "tcp_send( )" 傳送的訊息.
3. 當傳送暫存器裡有訊息, 控制器只會在每次 PLC scan 傳送出一個訊息. 例如: 傳送暫存器裡有 100 個訊息, 則需要 100 次的 PLC scan 來傳送完這 100 個訊息. 然而, 如果把 "Tcp_clie" 的 "Send_Time_Gap" 設定爲較大的值, 例如 100(ms), 則可以每隔 100 毫秒才傳送一個訊息.

範例: Wdemo_32 和 Wdemo_33 (Wincon CD_ROM:\napdos\isagraf\wincon\demo\)

TIME_STR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換 日期 & 時間 爲字串格式

輸入參數 :

YEAR_ :	**I**nteger	年, 2000 ~
MON_ :	Integer	月, 1 ~ 12
MDAY_ :	Integer	日, 1 ~ 31
WDAY_ :	Integer	星期幾, 1 ~ 7 (星期一 ~ 星期日)
HOUR_ :	Integer	時, 0 ~ 23
MIN_ :	Integer	分, 0 ~ 59
SEC_ :	Integer	秒, 0 ~ 59

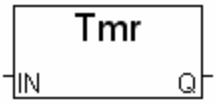
如果輸入參數有錯會回傳 M_ = '' (空字串). 如 MON_=14

傳回值 :

M_ :	**M**essage	字串長度爲 24, 如 'Feb/18/2003,13:25:45,Tue'

注意: 請使用 sysdat_r & systim_r 來取得控制器的 日期 & 時間

TMR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : Standard Function

轉換 任一變數 爲 Timer 格式

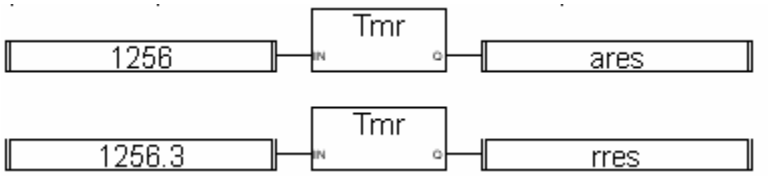
輸入參數 :

IN_ :	Int-real	任何非計時器格式的值 IN (若爲實數, 則指整數部分) 單位爲毫秒

傳回值 :

Q_ :	Timer	IN 的計時器格式值

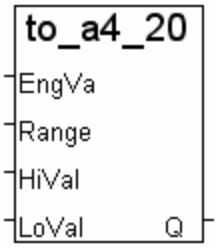
範例: (* FBD example with "Convert to Timer" blocks *)




```
(* ST 相等式: *)
    ares := TMR (1256);          (* ares := t#1s256ms *)
    res := TMR (1256.3);         (*rres := t#1s256ms *)

(* IL equivalence: *)
    LD      1256
    TMR
    ST      ares
    LD      1256.3
    TMR
    ST      rres
```

TO_A4_20



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

將使用者工程單位的值 ("實數" 格式) 轉換爲 IO 板卡類比輸出訊號的值 (類比輸出訊號爲 4 ~ 20mA, 整數格式)

輸入參數 :

EngVal_ :	Real	要轉換的工程單位值.
Range_ :	Integer	類比訊號輸出卡或模組的 Range 設定. 16#0 : 0 ~ 20 mA 16#1 : 4 ~ 20 mA 16#30 : 0 ~ 20 mA 16#31 : 4 ~ 20 mA
HiVal_ :	Real	當類比輸出訊號爲 20 mA 時, 使用者工程單位相對的高對應值
LoVal_ :	Real	當類比輸出訊號爲 4 mA 時, 使用者工程單位相對的低對應值

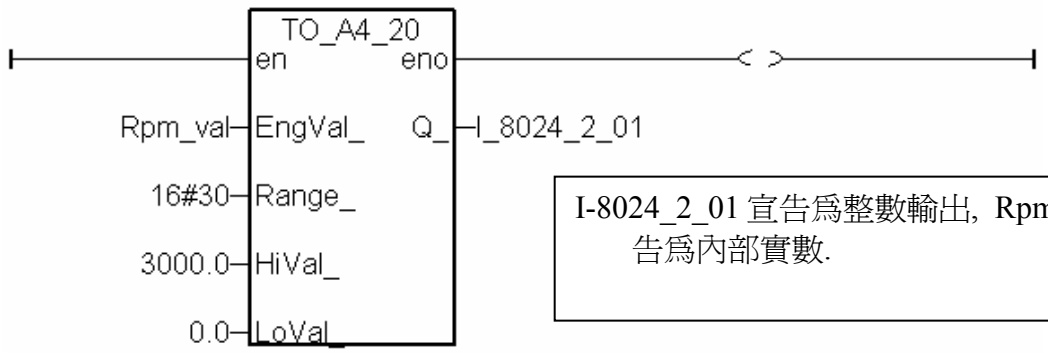
EX: 將 0 - 100 psi 轉換成 I-8024 的 AO 值, 請設定
HiVal_ = 100.0 , LoVal_ = 0.0 及 Range_=16#30 (請依據相關 IO 卡的 range 設定)

傳回值 :

Q_ :	Integer	傳回轉換後的 AO 值 (通常在 0 ~ +32767 之間, 依據 IO 卡的 Range 設定而異). 若輸入不正確的 Range_ 或 (HiVal_ = LoVal_), 則回傳 -1

範例 :

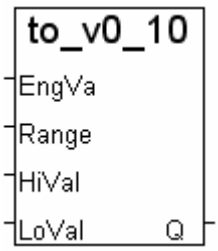
1. 將 (0 ~ 100) psi 轉換爲 (6554 ~ 32767) (若類比輸出卡的 range 設定爲 16#30 : 0 - 20mA)
2. 將 (0 ~ 100) psi 轉換爲 (0 ~ 32767) (若類比輸出卡的 range 設定爲 16#31 : 4 - 20mA)
3. 將 (0 ~ 3000) rpm 轉換輸出到 I-8024 (range 設定爲 30: 0 ~ 20 mA). 0 rpm 要輸出爲 4 mA , 3000 rpm 輸出爲 20 mA.



注意: 1. 類比輸出卡或模組的相關 range 需設定爲 mA .
例如, (-20 , +20mA) , (0 , 20mA) , (4 , 20mA) 等 mA

2. 使用 A4_20_to, To_A4_20, To_V0_10, V0_10_to 等 function, 需將 driver 更新爲 i-7188EG: 2.16 版, i-7188XG:2.14 版, i-8xx7:3.18 版, 或更高的版本, 程式才不會有問題(較舊的 driver 會發生程式 run 一段時間後會停止的現象).

TO_V0_10



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

將使用者工程單位的值 ("實數" 格式) 轉換爲 IO 板卡類比輸出訊號的值 (類比輸出訊號爲 0 ~ 10 V, 整數格式)

輸入參數 :

EngVal_ :	Real	要被轉換的使用者工程單位值.
Range_ :	Integer	類比輸出卡或模組的 Range 設定. 16#2 : 0 ~ 10 V 16#32 : 0 ~ 10 V 16#33 : -10 ~ 10 V 16#34 : 0 ~ 5 V 16#35 : -5 ~ +5 V
HiVal_ :	Real	當類比輸出訊號爲 10 V 時, 使用者工程單位相對的高對應值
LoVal_ :	Real	當類比輸出訊號爲 0 V 時, 使用者工程單位相對的高對應值

Ex: 將 0 - 100 psi 轉換成 I-8024 的 AO 值, 請設定
HiVal_ = 100.0 , LoVal_ = 0.0 及 Range_=16#33 (請依據相關 IO 卡的 range 設定)

傳回值 :

Q_ :	Integer	傳回轉換後的 AO 值 (通常在 0 ~ +32767 之間, 依據 IO 卡的 Range 設定而異). 若輸入不正確的 Range_ 或 (HiVal_ = LoVal_), 則回傳 -1

範例 :

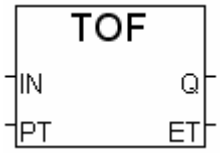
1. 將 (0 - 100) psi 轉換爲 (0 - 32767) (若類比輸出卡的 range 設定爲 16#32 : 0 ~ 10V)
2. 將 (0 - 100) psi 轉換爲 (0 - 32767) (若類比輸出卡的 range 設定爲 16#33 : -10 ~ +10V)

注意: 1. 類比輸出卡或模組的相關 range 需設定爲 Voltage range.
例如, (0 , 10 V) , (-10 , 10 V)等

2. 使用 A4_20_to, To_A4_20, To_V0_10, V0_10_to 等 function, 需將 driver 更新爲 i-7188EG: 2.16 版, i-7188XG:2.14 版, i-8xx7:3.18 版, 或更高的版本, 程式才不會有問題(較舊的 driver 會發生程式 run 一段時間後會停止的現象).

TOF

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : Standard Function

OFF-Delay (延遲關閉) 的控制.

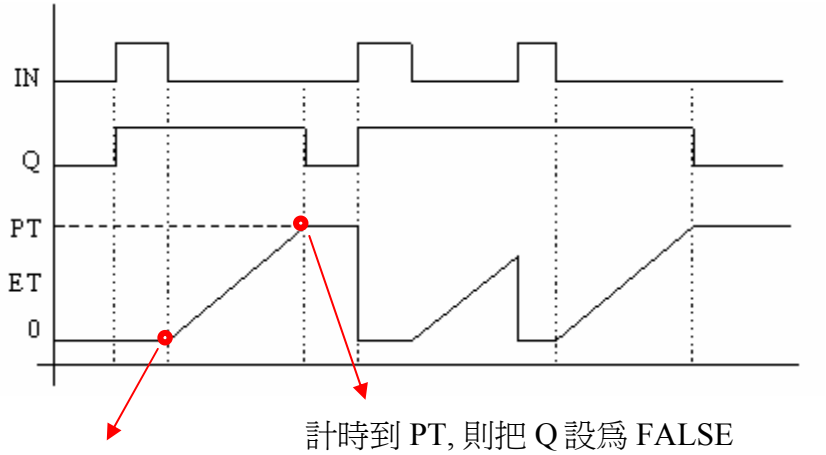
輸入參數 :

IN :	Boolean	控制 off-delay 的輸入, 若爲上升, 則 Q 馬上輸出爲 ON 若爲下降, 則啓動 ET 的 off-delay 開始計時
PT :	Timer	設定的 delay (延遲) 時間, 當 ET 計時到 PT 時, 就將 Q 關閉(設爲 False)

傳回值 :

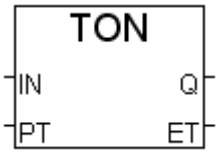
Q :	Boolean	輸出
ET :	Timer	TOF 開始動作時 經過的時間

時序圖: (延遲關閉)



TON

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6



型態 : Standard Function

ON-Delay (延遲啓動) 的控制.

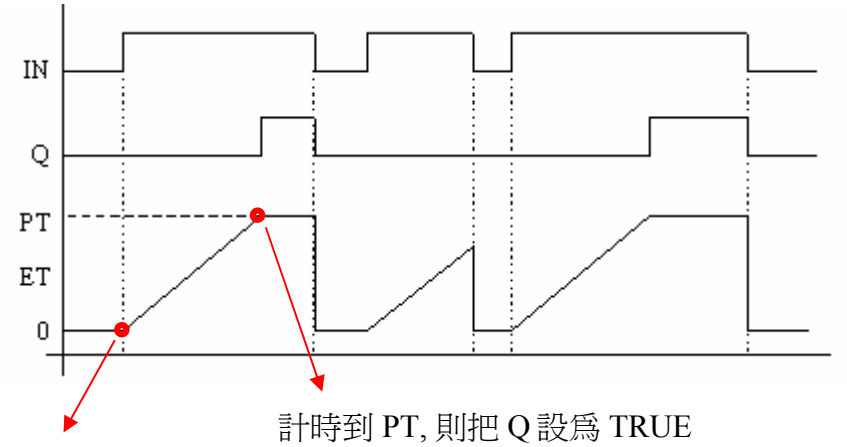
輸入參數 :

IN :	Boolean	控制 on-delay 的輸入, 若爲上升, 則啓動 ET 的 on-delay 開始計時 若爲下降, 則停止計時並重設 Q 爲 False
PT :	Timer	設定的 delay (延遲) 時間

傳回值 :

Q :	Boolean	輸出
ET :	Timer	TON 開始動作時 經過的時間

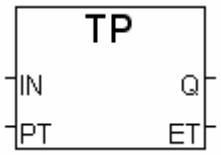
時序圖: (延遲啓動)



計時到 PT, 則把 Q 設爲 TRUE

IN 爲上昇開始計時

TP



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : Standard Function

輸出指定的一段時間爲 ON (Pulse timer).

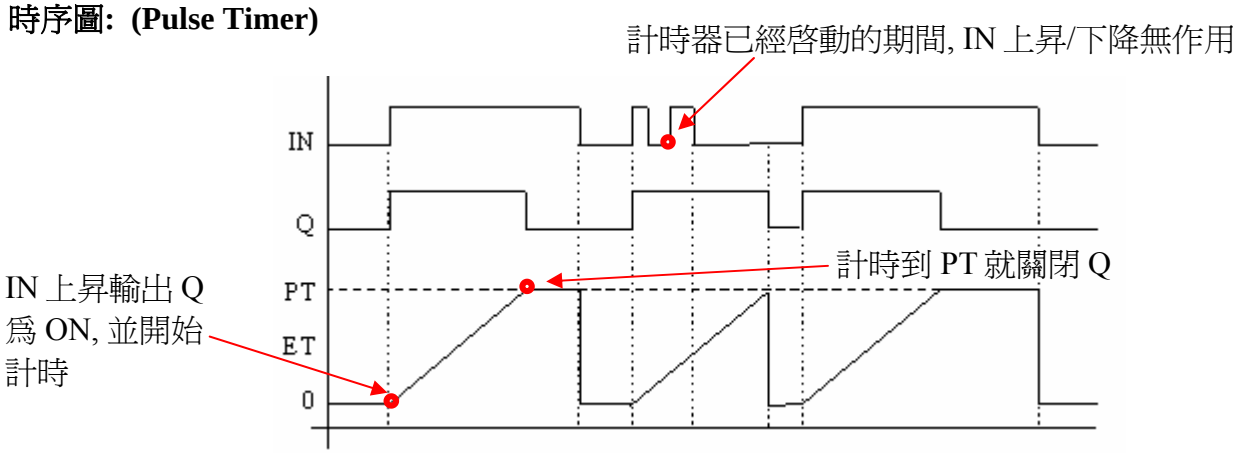
輸入參數 :

IN :	Boolean	控制 TP 的輸入, 當 IN 上升, 則 Q 輸出爲 ON, 並啓動 ET 計時, 若爲下降, 則停止計時並重設計時器 (IN 的上升下降只在 Q 爲 off 時有效, 且計時期間 IN 的改變無效)
PT :	Timer	設定 Q 要 ON 的時間長度

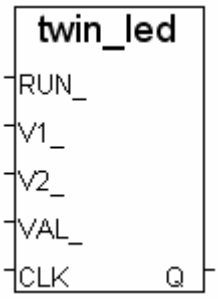
傳回值 :

Q :	Boolean	輸出
ET :	Timer	TP 開始動作時 經過的時間

時序圖: (Pulse Timer)



TWIN_LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

在 S-MMI 上顯示 2 個數值畫面.

輸入參數 :

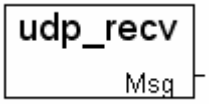
RUN_	Boolean	設爲 TRUE 才有動作
V1_	Integer	要顯示在第 1 個畫面左側 2 個位置的值, 0 ~ 99
V2_	Integer	要顯示在第 1 個畫面右側 2 個位置的值, 0 ~ 99
VAL_	Integer	要顯示在第 2 個畫面的值, -99999 ~ 99999
CLK_	Timer	畫面切換的週期時間

傳回值 :

Q_	Boolean	只傳回 TRUE

範例: 請參考 demo_10.

UDP_RECV



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

從遠端的 UDP/IP 連結 接收訊息(透過乙太網路). (請參考 19.2 節)

傳回值 :

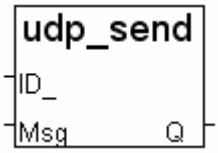
Msg_:	Message	接收的訊息. 若 Msg_ = '' (空訊息), 表示沒有訊息進來

注意 :

1. W-8xx7 的 UDP 功能的接收的 buffer 大小爲 8192 byte , 包含每個字串的結尾符號 1 byte
   而 I-7188EG 與 I-8437/8837 的接收的 buffer 大小則爲 2048 byte.
2. 如果一次進來太多個字串 導致 接收的 buffer 無法及時消化掉, 新進來的字串會取代掉 最早進來的那筆 字串.

範例 : Wdmo_19 和 Wdmo_19a (Wincon CD_ROM:\napdos\isagraf\wincon\demo\)

UDP_SEND



□ I-8417/8817 □ I-8437/8837 □ I-7188EG/7186EG □ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

傳送訊息到 UDP/IP 連結的遠端 PC 或控制器 (透過乙太網路). (請參考 19.2 節)

輸入參數 :

ID_ :	Integer	對應的連結, 可爲 1 到 4. 對應的 "IP 位址" 和 "埠號" 請在 "udp_ip" 裡設定
Msg_ :	Message	要傳送的訊息

傳回值 :

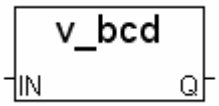
Q_ : :	Boolean	True: 傳送 OK, False: 傳送 buffer 已滿或參數錯誤 (如, ID_ = 8).

注意 :

1. W-8xx7 的 UDP 功能的傳送的 buffer 大小爲 2048 byte , 包含每個字串的結尾符號 1 byte
   表示 每個 ISaGRAF PLC Scan 最多只能傳送出 2048 個 byte .
2. 請不要很頻繁的 傳送大量資料出去, 跟 "udp_ip" 內的 "Send_Time_Gap" 參數有關, 若資料太多了, 消化不掉, 會積在傳送的 buffer 內, Buffer 若滿了, 使用 udp_send( ) 會傳回 False, 表示滿了, 無法再塞入更多的 Message. 每個 PLC Scan 對每個 UDP 連線, 只能消化 1 筆 Message (傳出 1 筆)

範例 : Wdmo_19 和 Wdmo_19a (Wincon CD_ROM:\napdos\isagraf\wincon\demo\)

V_BCD



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■W-8xx7/8xx6

型態 : C_Function

轉換十進位值爲 BCD 值.

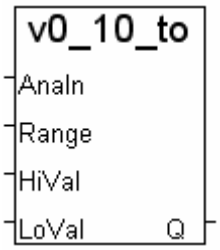
輸入參數 :

IN_ :	Integer	要轉換的十進位值, 有效範圍 : 0 ~ 99999999

傳回值 :

Q :	Integer	BCD 值 ex: 12345 → 16#12345 16 → 22 (16#16)

V0_10_TO



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■W-8xx7/8xx6

型態 : C_Function

將類比輸入訊號的值從 0 - 10V 單位轉換成使用者工程單位("實數" 格式).
如: 將 I-8017H 的類比輸入值轉換成 0 - 100 psi. 或 0 - 3000 rpm

輸入參數 :

AnaIn_ :	Integer	整數變數, 與類比輸入卡或模組有關. 在 -32768 ~ +32767 之間, 依據 IO 卡的 range 設定而異.
Range_ :	Integer	類比輸入 IO 卡或模組的 range 設定. 16#0 : -15mV ~ +15 mV 16#1 : -50mV ~ +50 mV 16#2 : -100mV ~ +100 mV 16#3 : -500mV ~ +500 mV 16#4 : -1 ~ +1 V 16#5 : -2.5 ~ +2.5 V 16#7 : -1.25 ~ +1.25 V 16#8 : -10 ~ +10 V 16#9 : -5 ~ +5 V 16#A : -1 ~ +1 V 16#B : -500mV ~ +500 mV 16#C : -150mV ~ +150 mV
HiVal_ :	Real	當類比輸入訊號爲 10 V 時, 使用者工程單位相對的高對應值
LoVal_ :	Real	當類比輸入訊號爲 0 V 時, 使用者工程單位相對的低對應值

Ex: 將 I-8017H 的輸入訊號由 0 - 10 V 轉換成 0 - 100 psi, 請設定 HiVal_ = 100.0 , LoVal_ = 0.0 及 Range_ = 16#5 或 16#7 或 16#8 或 16#9
(請依據相關 IO 卡的 range 設定)

傳回值 :

Q_ :	Real	轉換後使用者工程單位的值. 若輸入不正確的 Range_ , 則回傳 1.23E-2

注意: 1. 類比輸入卡或模組的相關 range 需設定爲 Voltage.
例如, (-10 , +10V) , (-5 , +5V) , (-1 , 1 V), 等 Voltage range.

2. 使用 A4_20_to, To_A4_20, To_V0_10, V0_10_to 等 function, 需將 driver 更新爲 i-7188EG: 2.16 版, i-7188XG:2.14 版, i-8xx7:3.18 版, 或更高的版本, 程式才不會有問題(較舊的 driver 會發生程式 run 一段時間後會停止的現象).

VAL_HEX

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

轉換整數爲 16 進位字串.

輸入參數 :

VAL_	Integer	要被轉換的值
DIGIT_	Integer	要轉換成幾個位數, 1 ~ 8 , 超出範圍傳回 ' ' (空字串)

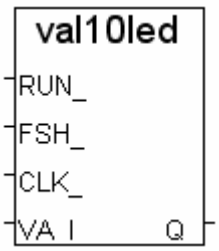
傳回值 :

HEX_	Message	轉換後的字串

範例:

val_hex(100,3)	--->	'064'
val_hex(192,4)	--->	'00C0'
val_hex(4589,2)	--->	'ED' ('11ED', 位數爲 2, 造成 '11' 被刪掉)
val_hex(4589,9)	--->	' ' (位數 > 8, 傳回空字串)
val_hex(-2,8)	--->	'FFFFFFFE'

VAL10LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

在 S-MMI 上顯示 1 個 10 進位數值

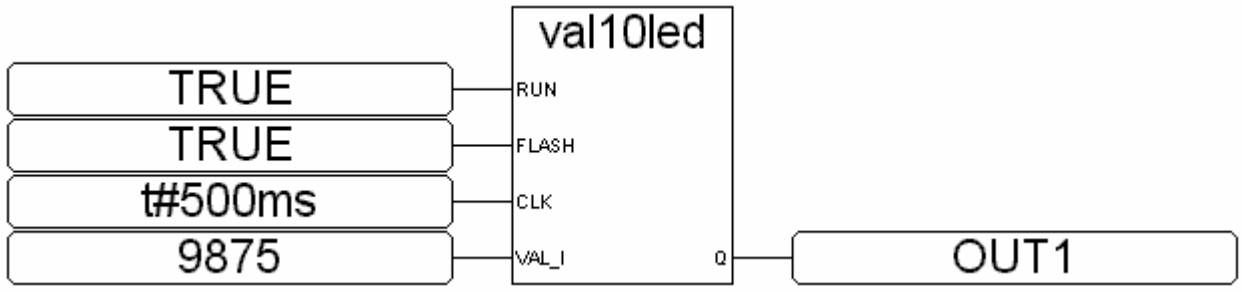
輸入參數 :

RUN_	Boolean	設爲 TRUE 才動作
FLASH_	Boolean	設爲 TRUE 則閃爍
CLK_	Timer	閃爍的週期時間
VAL_I_	Integer	要顯示的 integer, -9999 ~ +99999

傳回值 :

Q_	Boolean	只傳回 TRUE

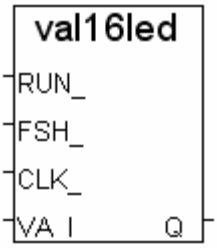
範例: 請參考 demo_07 及 demo_11b



ST 相等式:

```
TMP := VAL10LED(TRUE,TRUE,t#500ms,9875);
(* TMP 宣告爲 boolean *)
```

VAL16LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG □ W-8xx7/8xx6

型態 : C_Function

在 S-MMI 上顯示 1 個 16 進位數值

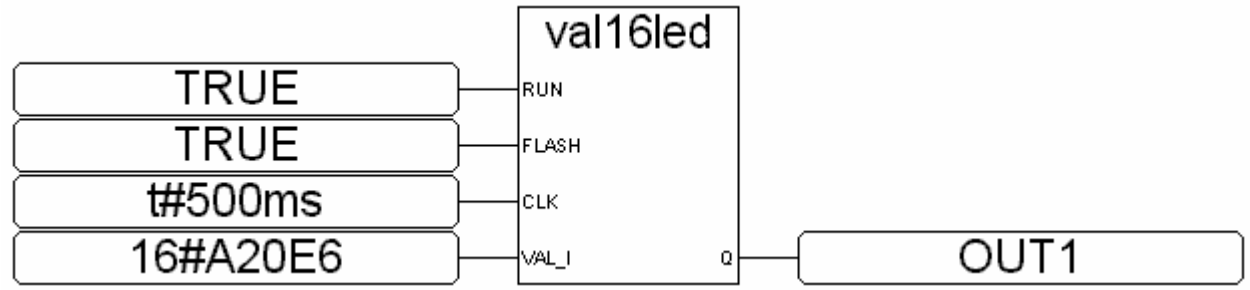
輸入參數 :

RUN_	Boolean	設爲 TRUE 才動作
FLASH_	Boolean	設爲 TRUE 則閃爍
CLK_	Timer	閃爍的週期時間
VAL_I_	Integer	要顯示的值, 16#0 ~ 16#FFFFF

傳回值 :

Q_	Boolean	只傳回 TRUE

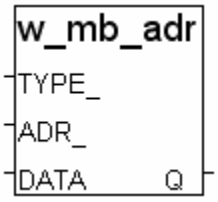
範例:



ST 相等式:

```
TMP := VAL10LED(TRUE,FALSE,t#500ms,16#A20E6);
(*TMP 宣告爲 boolean *)
```

W_MB_ADR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8ss7/8ss6

型態 : C_Function

使用 Modbus 位址, 寫入值到布林或整數變數

輸入參數 :

TYPE_ :	Integer	0: 布林變數, 1: 整數變數
ADR_ :	Integer	要寫入的 Modbus 位址, 有效範圍 Wincon: 1~8191, 其餘: 1~4095
DATA_ :	Integer	要寫入的整數 (或 布林, 0:False, 1:True)

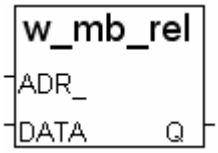
傳回值 :

Q_ :	Boolean	TRUE : 成功. , FALSE : 失敗

注意 :

1. “實數” 變數請使用 W_MB_REL function 來寫入.
2. 若指定的 Modbus 位址沒有定義對應的變數, 則不做寫入動作.
3. 若 TYPE_ 爲整數, 而對應的變數是 "實數" 型態, 則寫入相對應的 32-bit. 最好使用 "W_MB_REL" 來寫入實數變數.
4. 若 TYPE_ 爲整數, 而對應的變數是 "布林" 型態, 則不做寫入動作.
5. 若 TYPE_ 爲布林, 而對應的變數不是 "布林" 型態, 則不做寫入動作.
6. 若長整數 (32-bit 整數) 要藉由 Modbus 通訊協定傳遞到 HMI, 需佔用 2 個 Modbus 位址編號. 詳細資料請參考 ISaGRAF 使用手冊第 4.2 章.

W_MB_REL



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

使用 Modbus 位址寫入值到實數變數

輸入參數 :

ADR_ :	Integer	要寫入的 Modbus 位址, 有效範圍 Wincon: 1~8191, 其餘: 1~4095
DATA_ :	Real	要寫入的實數

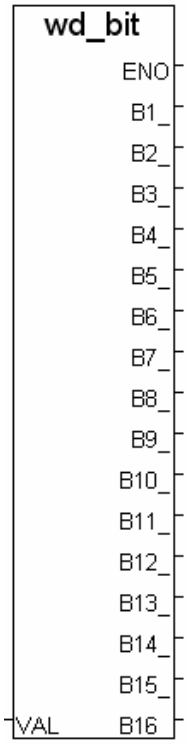
傳回值 :

Q_ :	Boolean	TRUE : 成功. , FALSE : 失敗

注意 :

1. 請確認對應變數的型態爲 "實數". 若爲 "整數", 請使用 "W_MB_ADR" function.
2. 若對應的變數型態不是 "類比 Analog" (即實數或整數), 則不做寫入動作.
3. 若指定的 Modbus 位址沒有定義對應的變數, 則不做寫入動作.
4. 若實數藉由 Modbus 通訊協定傳遞到 HMI, 需佔用 2 個 Modbus 位址編號. 詳細資料請參考 ISaGRAF 使用手冊第 4.2 章.

WD_BIT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function Block

轉換 1 個 word (signed 16-bit) 值成 16 boolean 值

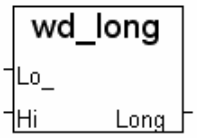
輸入參數 :

VAL_	Integer	要被轉換的 word (只有最低的 16-bit 有用)

傳回值 :

ENO_	Boolean	保留, 無作用.
B1_ ~ B16_	Boolean	轉換後的 16 個 boolean 值, 例如 VAL_ 爲 4, 則 B3_ 爲 TRUE, 其它爲 FALSE.

WD_LONG



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG/7186EG ■ I-7188XG ■ W-8xx7/8xx6

型態 : C_Function

合併 2 個 word (signed 16-bit) 爲 1 個長整數 (signed 32-bit)

輸入參數 :

Lo_	Integer	要被轉換的 Low word (只有最低的 16-bit 有用)
Hi_	Integer	要被轉換的 High word (只有最低的 16-bit 有用)

傳回值 :

Long_	Integer	轉換後的 integer

範例:

Lo_	Hi_	--->	Long_
-32768 (8000)	-1 (FFFF)	--->	-32768 (FFFF 8000)
-1 (FFFF)	-1 (FFFF)	--->	-1 (FFFF FFFF)
-32768 (8000)	0 (0000)	--->	+32768 (0000 8000)
100 (0064)	4103 (1007)	--->	+ 268 894 308 (1007 0064)

附錄 B ： 設定 I-8437/8837, I-7188EG & µPAC-7186EG 的 IP, Mask, Gateway

在這個章節裡,我們描述如何設定 I-8437 /8837 & I-7188EG & µPAC-7186EG控制器的 IP，Mask 及Gateway位址. 有關WinCon-8xx7/8xx6 控制器的設定方式，請參考其 “快速上手手冊”, 放置在 http://www.icpdas.com/products/PAC/i-8000/getting_started_manual.htm .

****

每一台 I-8437/8837, I-7188EG & µPAC-7186EG 控制器可以使用通訊埠號碼 502 來與人機界面程式和 ISaGRAF Workbench 溝通。且最多有 4 台 PC 可以透過 Modbus TCP/IP 通訊協定與 I-8437 /8837, I-7188EG & µPAC-7186EG 控制器溝通。

****

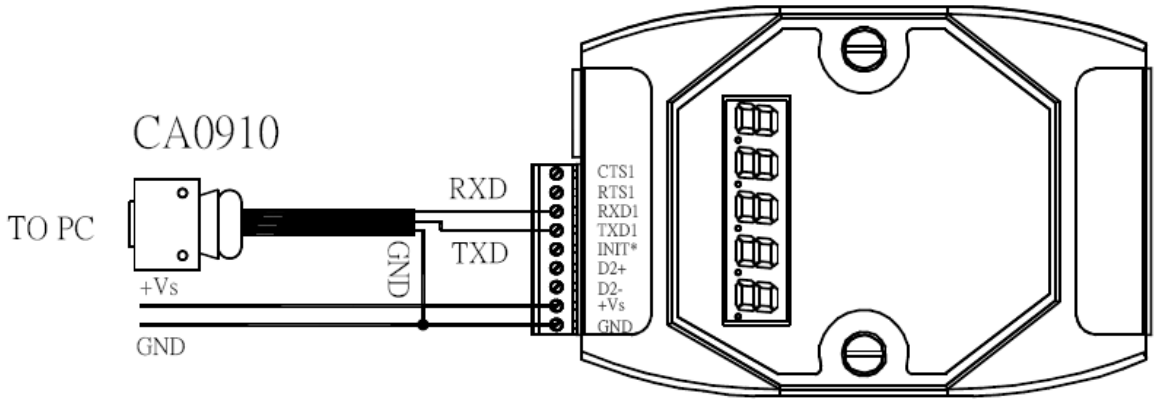
在 I-7188EG & µPAC-7186EG 控制器的設定步驟:

1. 在您的 PC 硬碟建立一個名爲 "7188" 的資料夾. 例如: "c:\7188".

2. 複製 CD 中的\Napdos\ISaGRAF\7188EG\Driver\2.xx\7188xw.exe, 7188xw.ini, 檔案到您建立的 “7188” 資料夾:

3. 執行 "\7188\7188xw.exe" (Windows NT, Windows 2000 & Windows XP) 或執行

4. 藉由 RS232 傳輸線 (CA0910), 連接您 PC 的 COM1 或 COM2 到控制器的 COM1



若您的電腦沒有 COM1/COM2 或您要使用其他 COM 埠 (如 COM5)來連接 I-7188, 可以變更 “7188xw.ini” 檔案第一行的 “C 編號”.

例如: 使用電腦的 COM5 連接 I-7188, 則更改 C1 爲 C5, 如下

```
C1 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```

⇒

```
C5 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```

5. 關閉 I-7188 EG/XG 電源, 連接 "INIT" 和 "GND", 重新開啓電源.

6. 假如連接成功，畫面上將會出現 “i7188E >” 的訊息, 如圖.

7. 輸入 "ip" 可查看目前的 IP 位址設定值

8. 輸入 "ip xxx.xxx.xxx.xxx" 可設定新的 IP 位址.

   Ex: i7188E> ip 192.168.1.200

9. 輸入 "mask" 可查看目前位址的遮罩.

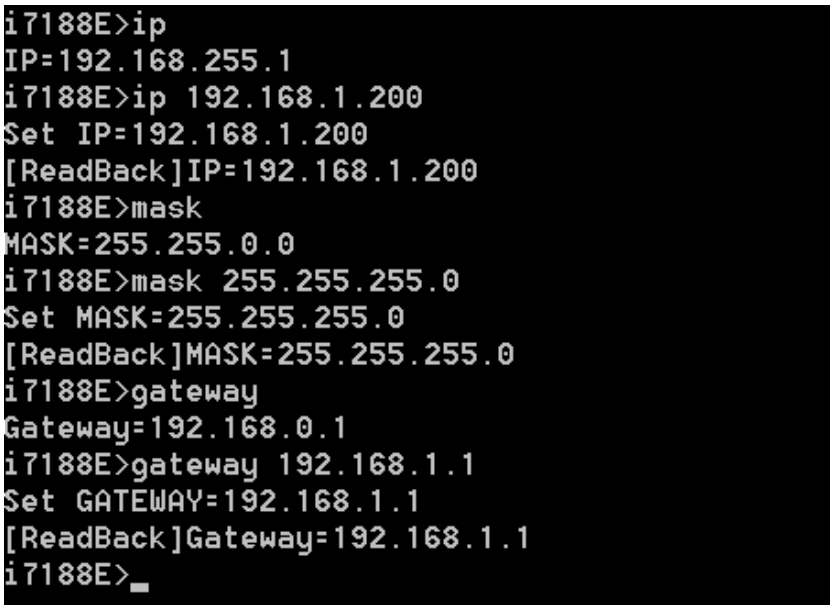
10. 輸入 "mask xxx.xxx.xxx.xxx" 可設定新的位址遮罩(mask).
    Ex: i7188E> mask 255.255.255.0

11. 輸入 "gateway" 可查看目前的 gateway 位址.
    i7188E> gateway

12. 輸入 "gateway xxx.xxx.xxx.xxx" 可設定新的 gateway 位址.
    i7188E> gateway 192.168.1.1

13. 按 ALT_X 來離開 exit "7188xw" 視窗, 否則 COM1/COM2 會一直被佔用住.

14. 將 “INIT” –“GND” 的連接線移開, 重新啓動控制器.

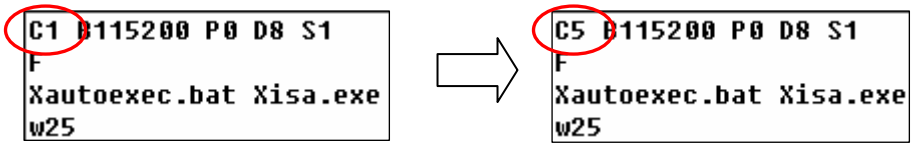


在 I-8437/8837 控制器的設定步驟:

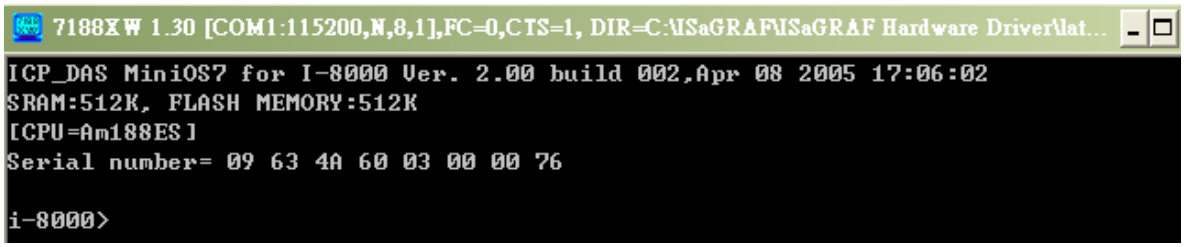
1. 在您的硬碟裡建立一個 “8000”的檔案目錄. 例如: "c:\8000".
2. 從 CD_ROM 複製 \Napdos\ISaGRAF\8000\Driver\....\7188xw.exe, 7188xw.ini 到 “8000”的目錄裡。
3. 執行\8000\7188xw.exe. 將會出現"7188xw " 視窗畫面。
4. 藉由 RS232 傳輸線，連接電腦上的 COM1 或 COM2 到 I-8437/8837 控制器的 COM1。

   如果您想使用其他的 COM port(ex.COM5),請修正”7188xw.ini” 檔案第一行的 “C 編號”。

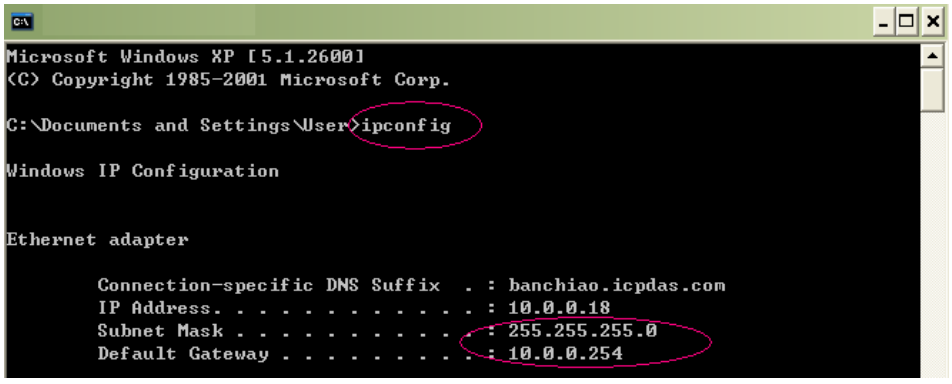
   例如: 使用電腦的 COM5 連接, 則更改”7188xw.ini” C1 爲 C5, 如下



5. 將 I-8437 / 8837 控制器的電源關掉，連接 “INIT”和”INIT COM”，然後開起電源。
6. 假如連接成功，將會出現 7188xw 的畫面，如下圖所示。



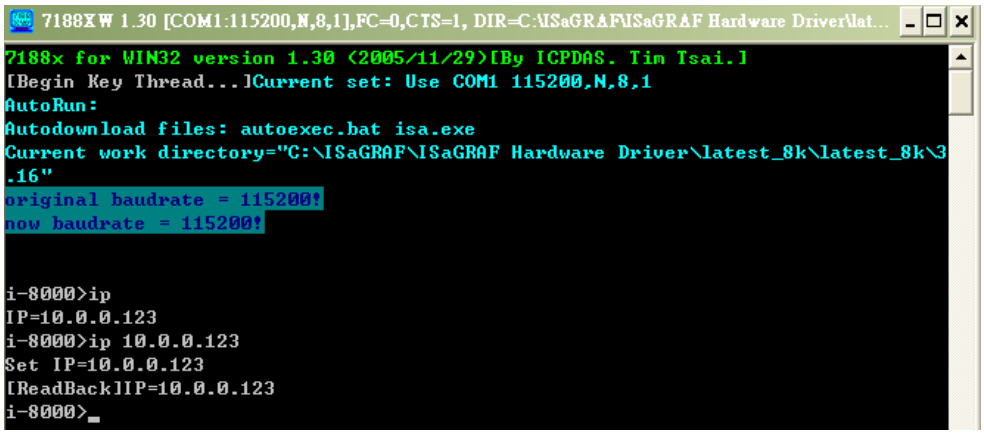
7. 先使用電腦上的”命令提示字元”找出電腦的網路設定值. 。



   ★ 請根據電腦中的網路設定值,進行後續的設定. (IP/MASK/GATEWAY)

8. 在命令列輸入 “ip”，可查看此 I-8437/8837 的 IP 位址。

   輸入 “ ip 10.0.0.xxx “，可設定新 IP 位址。



8000> ip
IP=192.168.255.255
8000>

9. 輸入"mask" 可以查出目前 I-8437/8837 的網路遮罩(address mask)。
輸入"mask 255.255.255.0" 設定新的網路遮罩。

```
i-8000>mask
MASK=255.255.255.0
i-8000>mask 255.255.255.0
Set MASK=255.255.255.0
[ReadBack]MASK=255.255.255.0
i-8000>
```

10. 輸入"gateway"可以查出目前的預設閘道(gateway)。
輸入"**gateway 10.0.0.254**"設定新的預設閘道。

```
i-8000>gateway
Gateway=10.0.0.254
i-8000>gateway 10.0.0.254
Set GATEWAY=10.0.0.254
[ReadBack]Gateway=10.0.0.254
i-8000>
```

11. 按 **ALT_X** 來離開 "7188x" 視窗, 否則 COM1 或 COM2 將會一直被佔用住。

12. 將 "INIT" –"INIT COM"的連接線移開，重新開機 I-8437 /8837 控制器。

附錄 C ： 更新 I-8417/8817/8437/8837 的驅動程式

ISaGRAF 的驅動程式屬於韌体, 燒錄在 I-8417/8817/8437/8837, I-7188EG/XG, µPAC-7186EG 及 Wincon-8xx7/8xx6 的 Flash 記憶体內。使用者可以很容易的自行更新驅動程式。

若要更新 Wincon-8xx7/8ss6, I-7188EG/XG & µPAC-7186EG的驅動程式, 更新方法請參考他們各自的 “快速上手手冊”, 放置在 CD_ROM : Napdos\ISaGRAF\ 下 各自的子資料夾內 或 http://www.icpdas.com/products/PAC/i-8000/getting_started_manual.htm .

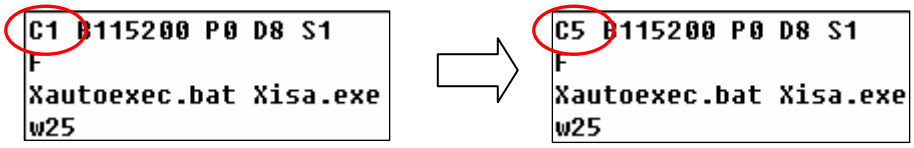
我們最新版本的驅動程式放置在 (**注意 :** 不同版本的檔案會是不同檔名的 img 檔)
http://www.icpdas.com/products/PAC/i-8000/isagraf.htm 或
http://www.icpdas.com/products/PAC/i-8000/isagraf-link.htm 下載後, 請解壓縮 (zip檔)

警告: 這個 ISaGRAF 驅動程式的智慧財產權是屬於泓格科技股份有限公司。目前只有 I-8417,8817,8437,8837, I-7188EG/XG, µPAC-7186EG 及 Wincon-8xx7/8xx6 已經註冊爲合法的 ISaGRAF Target license，若您燒錄此 ISAGRAF 驅動程式至其他的 I-8000 模組或其他相容產品是違法的。

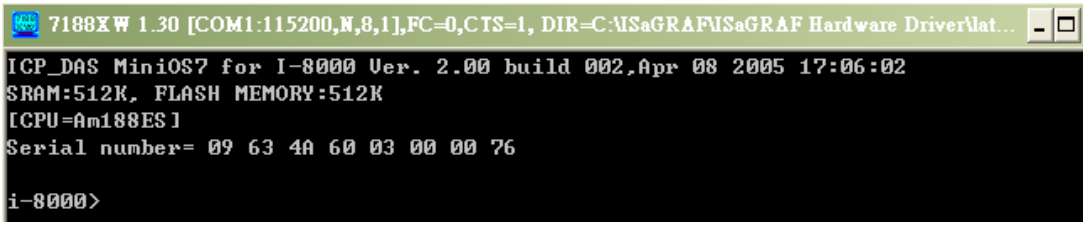
注意: 請先確定您目前的版本再更新驅動程式。

查看驅動程式版本 (此例,我們使用 3.16 版.)

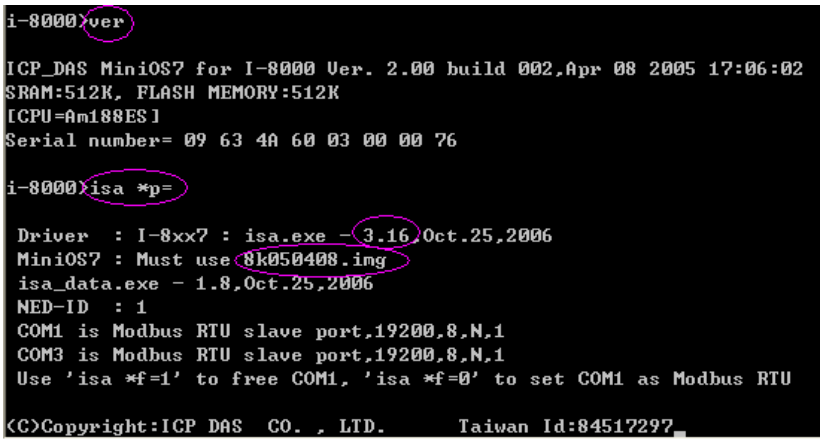
1. 在您的硬碟裡建立一個 “8000”的檔案目錄 (例如, "**c:\8000**")
2. 從 CD_ROM 複製 Napdos\ISaGRAF\8000\Driver\40m\3.16\
   ①.**7188xw.exe,** ②**7188xw.f4,** ③**7188xw.ini ,** ④**8k050408.img ,** ⑤**autoexec.bat ,** ⑥**isa.exe**, ⑦**isa_data.exe** 到 “8000” 的目錄裡。
   (注意 : 不同版本的檔案會是不同檔名的 img 檔)
3. 執行"c:\8000\7188xw.exe",開啓 " 7188xw " 視窗 (可按 F1 取得 Help).
4. 藉由 RS232 傳輸線，連接電腦上的 COM1 或 COM2 到 I-8xx7 控制器的 COM1。
   如果您想使用其他的 COM port(ex.COM5), 請修正 ”7188xw.ini”.的第 1 行。



5. 將 I-8xx7 控制器的電源關掉，連接 “INIT” 和 ” INIT COM”，然後開起電源。
6. 假如連接成功，將會顯示”i-8000>”訊息於 7188xw 的畫面。

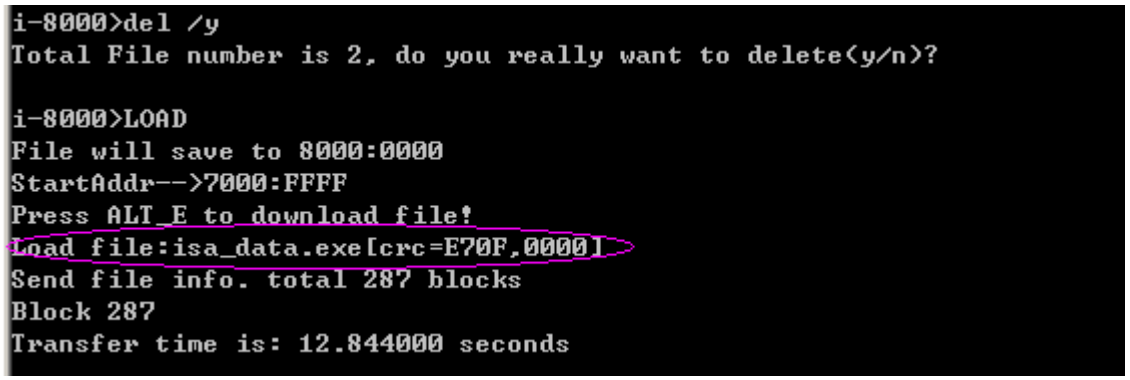


7. 輸入 "ver" 以查看控制器內的 OS 版本及日期.
8. 輸入"isa *p=", 將會顯示目前安裝的驅動程式版本編號和日期

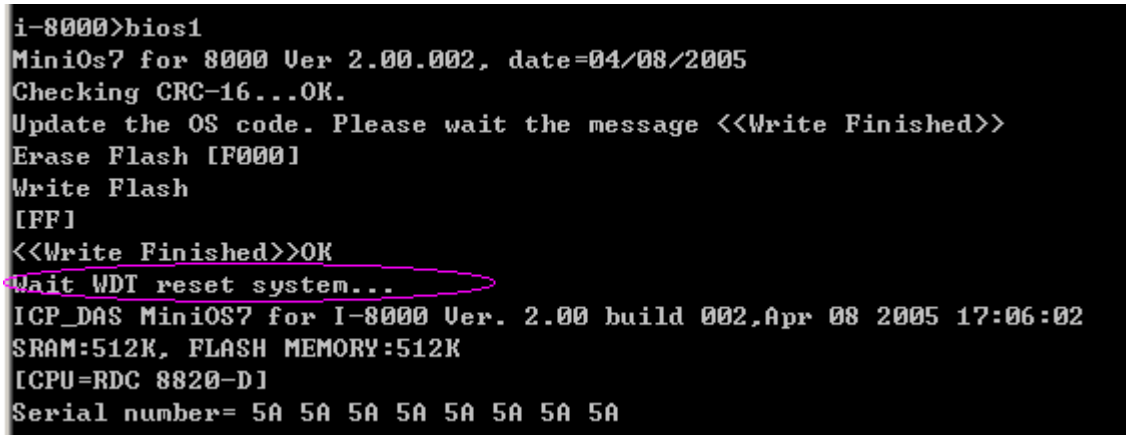


更新 ISaGRAF 內嵌式驅動程式

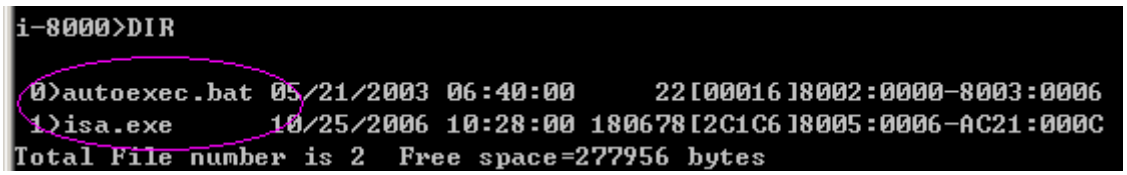
9. 將 I-8xx7 控制器的電源關掉，連接 “INIT”和” INIT COM”，然後開起電源。
10. 按 "**F4**" 將自動下載以下檔案並重新啓動系統. (約 60sec)
    (isa_data.exe,autoexec.bat,isa.exe, 8k050408.img)



🕐 **請等待約 60 秒讓 ISaGRAF 自動更新程式 ,期間請勿關閉電源！**



11. 輸入" dir " 來確定" autoexec.bat " 和 " isa.exe " 是否已下載成功



12. 按 Alt_X 來離開 " 7188xw "視窗

13. 將 “INIT” –“INIT COM”的連接線移開，重新開機 I-8xx7 控制器。

附錄 C.1: 設定 I-8xx7 & I-7188EG 的 COM1 爲非 Modbus-Slave port

I-8417/8817/8437/8837, I-7188EG 及µPAC-7186EG 的 COM1 預設爲支持 Modbus RTU Slave 通訊協議. 但是使用者可視需求關閉此功能, 將 COM1 變更爲非 Modbus-Slave port, 如此便可利用 COM1 來作其他用途. 比如可用 "COMxxx" 等函式撰寫自訂的通訊協議. 或使用 COM1 來當 Modbus Master port.

注意： 若是 7188XG, COM1 固定支援 Modbus RTU Slave 協議, 不可更改.

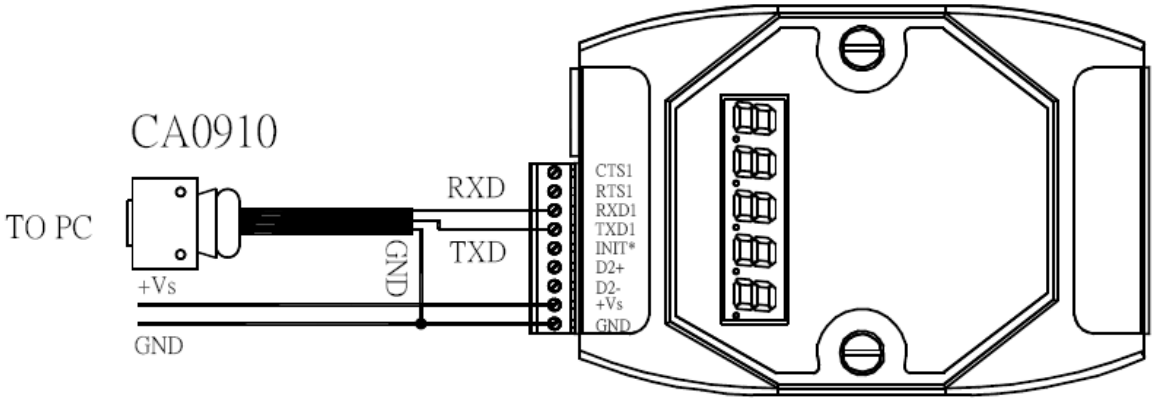
** 以下使用的控制器以 I-7188EG 爲例 : (I-8000 請參考括弧內說明)

步驟： 1. 在您的 PC 硬碟建立一個名爲 "7188" 的資料夾. 例如: "**c:\7188**". (I-8000 可用 "**c:\8000**")

2. 複製 CD 中的\Napdos\ISaGRAF\7188EG\Driver\2.xx\7188xw.exe, 7188xw.ini,檔案到您建立的 **"7188"** 資料夾. (I-8000 請複製到 "**8000**" 資料夾)

3. 執行該資料夾的 "7188xw.exe" (Windows NT, Windows 2000 & Windows XP 系統用此)

4. 藉由 RS232 傳輸線 (如 CA0910), 連接您 PC 的 COM1 或 COM2 到控制器的 COM1



若您的電腦沒有 COM1/COM2 或您要使用其他 COM 埠 (如 COM5)來連接控制器, 可以變更 "7188xw.ini" 檔案第一行的 "C 編號".

例如: 使用電腦的 COM5 連接控制器, 則更改 C1 爲 C5, 如下

C1 B115200 P0 D8 S1 F Xautoexec.bat Xisa7188e.exe w25	**C5** B115200 P0 D8 S1 F Xautoexec.bat Xisa7188e.exe w25

5. 關閉 I-7188 EG 電源, 連接 "INIT" 和 "GND", 重新開啓電源.
(I-8000 則連接 "INIT"和" INIT COM")

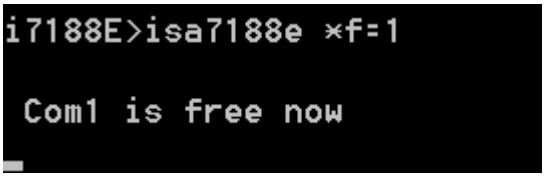
6. 假如連接成功，畫面上將會出現 "i7188E >" 的訊息, 如圖. (I-8000 則出現 "i-8000>")

7. 輸入"isa7188e *f=1" 來釋放控制器的 COM1 (即 將 COM1 設定爲 非 Modbus-Slave 埠)

i7188E> isa7188e *f=1 (I-7188EG 使用)
i7188E> isa7186e *f=1 (µPAC-7186EG 使用)
i-8000> isa *f=1 (I-8000 使用此命令)

8. 按 ALT_X 來離開 "7188xw" 視窗, 否則 PC 的 COM1/COM2 會一直被佔用住.

9. 將 “INIT” –“GND” 的連接線移開, 重新啓動控制器.
(I-8000 則將 “INIT”–“INIT COM”的連接線移開)



重要注意事項：

如果要恢復 COM1 爲 Modbus RTU Slave port, 請使用 "isa7188e *f=0" 指令 , 如下:

Ex1: i7188E> isa7188e *f=0 (I-7188EG 使用)
Ex2: i7188E> isa7186e *f=0 (µPAC-7186EG 使用)
Ex3: i-8000> isa *f=0 (I-8000 使用此)

附錄 D ： 類比 I/O 數值對照表

I-87013, I-7013, I-7033, I-7015, M-7015, M-7033, I-87015

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
20 (Default)	Platinum 100 a = 0.00385 -100 ~ 100 °C	Temperature (攝氏度)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+138.50	+060.60
21	Platinum 100 a = 0.00385 0 ~ 100 °C	Temperature (攝氏度)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+138.50	+100.00
22	Platinum 100 a = 0.00385 0 ~ 200 °C	Temperature (攝氏度)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+175.84	+100.00
23	Platinum 100 a = 0.00385 0 ~ 600 °C	Temperature (攝氏度)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+313.59	+100.00
24	Platinum 100 a = 0.003916 -100 ~ 100 °C	Temperature (攝氏度)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+139.16	+060.60
25	Platinum 100 a = 0.003916 0 ~ 100 °C	Temperature (攝氏度)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+139.16	+100.00
26	Platinum 100 a = 0.003916 0 ~ 200 °C	Temperature (攝氏度)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+177.14	+100.00

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
27	Platinum 100 a = 0.003916 0 ~ 600 °C	Temperature (攝氏度)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+317.28	+100.00
28	Nickel 120 -80 ~ 100 °C	Temperature (攝氏度)	+100.0	-80.0
		Decimal Value	+32767	-26214
		2's complement HEX	7FFF	999A
		Ohms	+200.64	+066.60
29	Nickel 120 0 ~ 100 °C	Temperature (攝氏度)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+200.64	+120.60
2A	Platinum 1000 a = 0.00385 -200 ~ 600 °C	Temperature (攝氏度)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+3137.1	+0185.2
2B*	Cu 100 a = 0.00421 -20 ~ +150 °C	Temperature (攝氏度)	+150.0	-20.0
		Decimal Value	+32767	-4369
		2's complement HEX	7FFF	EEEF
		Ohms	+163.17	+091.56
2C*	Cu 100 a = 0.00427 0 ~ 200 °C	Temperature (攝氏度)	+200.0	0.0
		Decimal Value	+32767	0
		2's complement HEX	7FFF	0
		Ohms	+167.75	+090.34
2D*	Cu 1000 a = 0.00421 -20 ~ 150 °C	Temperature (攝氏度)	+150.0	-20.0
		Decimal Value	+32767	-4369
		2's complement HEX	7FFF	EEEF
		Ohms	+1631.7	+0915.6
2E	Platinum 100 a = 0.00385 -200 ~ 200 °C	Temperature (攝氏度)	+200.0	-200.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+175.84	+018.49
2F	Platinum 100 a = 0.003916 -200 ~ 200 °C	Temperature (攝氏度)	+200.0	-200.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+177.14	+017.14

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
80	Platinum 100 a = 0.00385 -200 ~ 600 °C	Temperature (攝氏度)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+313.59	+018.49
81	Platinum 100 a = 0.003916 -200 ~ 600 °C	Temperature (攝氏度)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+317.28	+017.14

* Range Type Code 2B, 2C 與 2D 只有 I-7015, M-7015 及 I-87015 才有支援.
* I-87015, I-7015 與 M-7015 每點可設成不同的 Range Code.

I-8017H(8-ch), I-8017HS(16-ch)

Range Type Code (Hex)	Data Format	Max value	Min value
05	Input Range	+2.5 V	-2.5 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
06*	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
07	Input Range	+1.25 V	-1.25 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* I-8017H 每點可設成不同的 Range Code.
* 使用 Code 06 需外接 125Ω 電阻

Range Type Code (Hex)	Data Format	Max value	Min value
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0A	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0B	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0C	Input Range	+150.0 mV	-150.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0D*	Input Range (with 125 ohms resistor)	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* 使用 Code 0D 需外接 125Ω 電阻

Range Type Code (Hex)	Data Format	Max value	Min value
7	Input Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0
D	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
1A	Input Range	+20.0 mA	0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0

* 使用 I-7017RC, M-7017RC, I-87017RC 量電流不需外接電阻

I-87018Z, I-87018R, I-87018, I-87019R, I-7018Z, I-7018R, I-7018, M-7018, M-7018R, I-7019R, M-7019R (1)

Range Type Code (Hex)	Data Format	Max value	Min value
00	Input Range	-15.0 mV	-15.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
01	Input Range	+50.0 mV	-50.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
02	Input Range	+100.0 mV	-100.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
03	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
04	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
05 (Default)	Input Range	+2.5V	-2.5V
	Decimal Value	+100.00	-100.00
	2's Complement HEX	7FFF	8000
06*	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* 除了使用 I-7019, I-7019R, M-7019, M-7019R & I-87019R 量 Code 6 電流可用 jumper 來調整外, 其他 Module 量電流要外接 125Ω 電阻.

* 每塊 I-87018Z, I-7018Z 共有 10 個 Channel, 且每個 channel 可設爲不同的 Type Code 值

- I-87018Z, I-7018Z 另外有支援

Range Type Code (Hex)	Data Format	Max value	Min value
7	Input Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0
1A	Input Range	+20.0 mA	0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0

I-87018Z, I-87018R, I-87018, I-87019R, I-7018Z, I-7018R, I-7018, M-7018, M-7018R, I-7019R, M-7019R (2)

Range Type Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
0E	J Type -210 ~ 760 °C	Temperature (攝氏度)	+760.0	-210.0
		Decimal Value	+32767	-9054
		2's Complement HEX	7FFF	DCA2
0F	K Type -270 ~ 1372 °C	Temperature (攝氏度)	+1372.0	-270.0
		Decimal Value	+32767	-6448
		2's Complement HEX	7FFF	E6D0
10	T Type -270 ~ 400 °C	Temperature (攝氏度)	+400.0	-270.0
		Decimal Value	+32767	-22118
		2's Complement HEX	7FFF	A99A
11	E Type -270 ~ 1000 °C	Temperature (攝氏度)	+1000.0	-270.0
		Decimal Value	+32767	-8847
		2's Complement HEX	7FFF	DD71
12	R Type 0 ~ 1768 °C	Temperature (攝氏度)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
13	S Type 0 ~ 1768 °C	Temperature (攝氏度)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
14	B Type 0 ~ 1820 °C	Temperature (攝氏度)	+1820.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
15	N Type -270 ~ 1300 °C	Temperature (攝氏度)	+1300.0	-270.0
		Decimal Value	+32767	-6805
		2's Complement HEX	7FFF	E56B
16	C Type 0 ~ 2320 °C	Temperature (攝氏度)	+2320.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
17	L Type -200 ~ 800 °C	Temperature (攝氏度)	+800.0	-200.0
		Decimal Value	+32767	-8192
		2's Complement HEX	7FFF	E000

Range Type Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
18	M Type -200 ~ 100 °C	Temperature (攝氏度)	+100.0	-200.0
		Decimal Value	+16384	-32768
		2's Complement HEX	4000	8000

- I-87018Z, I-7018Z 另外有支援

Range Type Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
19	L Type DIN43710 -200 ~ 900°C	Temperature (攝氏度)	+900.0	-200.0
		Decimal Value	+32767	-7281
		2's Complement HEX	7FFF	E38F

- I-7019, I-7019R, M-7019, M-7019R & I-87019R 另外支持 :

Range Type Code (Hex)	Data Format		Max value	Min value
08 (Default)	Input Range		+10.0 V	-10.0 V
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
09	Input Range		+5.0 V	-5.0 V
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
0A	Input Range		+1.0 V	-1.0 V
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
0B	Input Range		+500.0 mV	-500.0 mV
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
0C	Input Range		+150.0 mV	-150.0 mV
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
0D*	Input Range (with 125 ohms resistor)		+20.0 mA	-20.0 mA
	Decimal Value		+32767	-32768
	2's Complement HEX		7FFF	8000
19	L Type DIN43710 -200 ~ 900°C	Temperature (攝氏度)	+900.0	-200.0
		Decimal Value	+32767	-7281
		2's Complement HEX	7FFF	E38F

* I-7019, I-7019R, M-7019, M-7019R & I-87019R 的 Range Code 可每 Channel 規劃爲不同值
* I-7019, I-7019R, M-7019, M-7019R & I-87019R 使用 Code 0D 不需外接 125ohm 電阻, 只需用 Jumper 調整即可.

I-7021

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
32 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

I-7022

Range Type Code (Hex)	Data Format	Max Value	Min Value
0	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
1	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
2 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

Range Type Code (Hex)	Themistor Type	Data Format	Max Value	Min Value
60	PreCon Type III 10K @ 25 °C -35 ~ 115 °C	Temperature(華氏度)	+240.00 °F	-030.00 °F
		2's complement HEX	7FFF	E000
		Ohms	+000539.4	+173600.0
61	Fenwell U 2K @ 25 °C -50 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000037.2	+134020.0
62	Fenwell U 2K @ 25 °C 0 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	+000.00 °C
		2's complement HEX	7FFF	0000
		Ohms	+000037.2	+006530.0
63	YSI L Mix 100 @ 25 °C -80 ~ 100 °C	Temperature(攝氏度)	+100.00 °C	-080.00 °C
		2's complement HEX	7FFF	999A
		Ohms	+000014.3	+014470.0
64	YSI L Mix 300 @ 25 °C -80 ~ 100 °C	Temperature(攝氏度)	+100.00 °C	-080.00 °C
		2's complement HEX	7FFF	999A
		Ohms	+000035.8	+067660.0
65	YSI L Mix 1000 @ 25 °C -70 ~ 100 °C	Temperature(攝氏度)	+100.00 °C	-070.00 °C
		2's complement HEX	7FFF	A667
		Ohms	+000106.4	+132600.0
66	YSI B Mix 2252 @ 25 °C -50 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000041.8	+151000.0
67	YSI B Mix 3000 @ 25 °C -40 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-040.00 °C
		2's complement HEX	7FFF	DDDE
		Ohms	+000055.6	+101000.0
68	YSI B Mix 5000 @ 25 °C -40 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-040.00 °C
		2's complement HEX	7FFF	DDDE
		Ohms	+000092.7	+168300.0
69	YSI B Mix 6000 @ 25 °C -30 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000111.5	+106200.0
6A	YSI B Mix 10K @ 25 °C -30 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000185.9	+177000.0

Range Type Code (Hex)	Themistor Type	Data Format	Max Value	Min Value
6B	YSI H Mix 10K @ 25 °C -30 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000237.0	+135200.0
6C	YSI H Mix 30K @ 25 °C -10 ~ 200 °C	Temperature(攝氏度)	+200.00 °C	-010.00 °C
		2's complement HEX	7FFF	F99A
		Ohms	+000186.7	+158000.0
70 ~ 77	User-defined -50 ~ 150 °C	Temperature(攝氏度)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000000.0	+000000.0

* 使用者定義的 Type, 如果電阻大於 180000 ohms, 則被當作低於標準範圍.
* 請參閱 I-7005/M-7005 User's Manual 第 1.11 節.

I-8024

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
33	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768

* 每個 channel 可規劃成不同的 Range 值

I-87024, I-7024

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
32	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
33 (Default)	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
34	Output Range	+5.0 V	+0.0 V
	Decimal Value	+32767	+0
35	Output Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768

Range Type Code (Hex)	Data Format	Max Value	Min Value
0	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
1	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
2	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	0

* I-87022, I-87026 的每個 channel 可規劃成不同的 Range 值

附錄 E ： ISaGRAF 語法參考

ISaGRAF

Version 3.46

LANGUAGE REFERENCE

AlterSys Inc.

E.1 Project architecture

An ISaGRAF project is divided into several programming units called **programs**. The programs of the project are linked together in a tree-like architecture. Programs can be described using any of **SFC, FC (Flow Chart)**, **FBD**, **LD, ST** or **IL** graphic or literal languages.

E.1.1 Programs

A **program** is a logical programming unit, which describes operations between **variables** of the process. Programs describe either **sequential** or **cyclic** operations. Cyclic programs are executed at each target system cycle. The execution of sequential programs follows the dynamic rules of either the **SFC** language or the **FC** language.

Programs are linked together in a hierarchy tree. Programs placed on the top of the hierarchy are activated by the system. Sub-programs (lower level of the hierarchy) are activated by their father. A program can be described with any of the available graphic or literal following languages:

Sequential Function Chart (SFC) for high level programming
Flow Chart (FC) for high level programming
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
Instruction List (IL) for low level operations

The same program cannot mix several languages, except LD and FBD can be combined in one diagram.

E.1.2 Cyclic and sequential operations

The hierarchy of programs is divided into four main **sections** or groups:

Begin	programs executed at the beginning of each target cycle
Sequential	programs following SFC or FC dynamic rules
End	programs executed at the end of each target cycle
Functions	set of non-dedicated sub-programs

Programs of the **'Begin'** or **'End'** sections describe cyclic operations, and are not time dependent. Programs of the '**Sequential**' section describe sequential operations, where the time variable explicitly synchronises basic operations. Main programs of the **'Begin'** section are systematically executed at the beginning of each run time cycle. Main programs of the **'End'** section are systematically executed at the end of each run time cycle. Main programs of the **'Sequential'** section are executed according to either the **SFC** or the **FC** dynamic rules.

Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. A program of the "**Function**" section can call another program of this section.
Main and child programs of the sequential section must be described with **SFC** or **FC** language. Programs of cyclic sections (**begin** and **end**) cannot be described with **SFC** or **FC** language. Any program of any section may own one or more sub-programs. Any program of the sequential section

may own one or more **SFC** or **FC** child programs (according to its own programming language). Sub-programs cannot be described with **SFC** or **FC** language.

Programs of the **Begin** section are typically used to describe preliminary operations on input devices to build high level filtered variables. Such variables are frequently used by the programs of the **Sequential** section. Programs of the **End** section are typically used to describe security operations on the variables operated on by the **Sequential** section, before sending values to output devices.

E.1.3 Child SFC and FC programs

Any **SFC** program of the sequential section may control other **SFC** programs. Such low-level programs are called **child SFC programs**. A **child SFC program** is a parallel program that can be started, killed, frozen or restarted by its parent program. The parent program and child program must both be described with the **SFC** language. A child SFC program may have local variables and defined words.

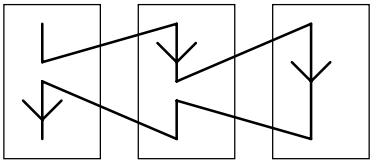
When a parent program starts a child **SFC** program, it puts an SFC **token** (activates) into each initial step of the child program. This command is described with the **GSTART** statement. When a parent program kills a child **SFC** program, it clears all the tokens existing in the **steps** of the child. Such a command is described with the **GKILL** statement.

When a parent program freezes a child **SFC** program, it suspends its execution. The suspended program can then be restarted using the **GRST** statement.

Any **FC** program of the sequential section may control other **FC** sub-programs. An **FC** father program is blocked (waits) during execution of an FC sub-program. It is not possible that simultaneous operations are done in father FC program and one of its FC sub-programs.

E.1.4 Functions and sub-programs

A sub-program or a function execution is driven by its parent program. The execution of the parent program is suspended until the sub-program or the function ends:



main sub-programs

Any program of any section may have one or more sub-programs. A sub-program is owned by only one father program. A sub-program may have local variables and defines. Any language but **SFC** or **FC** can be used to describe a sub-program. Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. Unlike other sub-programs, they are not dedicated to one father program. A program of the "**Function**" section can call another program of this section. A function can be located in the Library.

Warning: The ISaGRAF system does not support **recursive function calls**. A run time error will occur if a program of the "**Functions**" section is called by itself or by one of its called sub-program.

Warning: A function or sub-program does not "store" the local value of its local variables. A function or sub-program is not instantiated and so can not call function blocks.

The interface of a sub-program must be explicitly defined, with a **type** and a **unique name** for each of its calling or return parameter. In order to support the **ST** language convention, the return parameter must have the same name as the sub-program.

The following table shows how to set the value of the return parameter in the body of a sub-program, in the different languages:

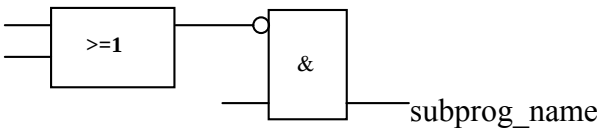
ST: assign the return parameter using its name
(the same name as the sub-program):

```
subprog_name := <expression>;
```

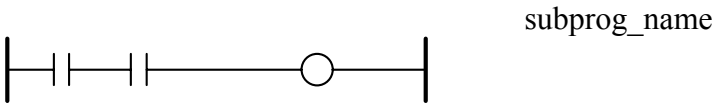
IL: the value of the current result (IL register)
at the end of the sequence is stored in the return parameter:

```
LD    10
ADD   20   (* return parameter value = 30 *)
```

FBD: set the return parameter using its name:

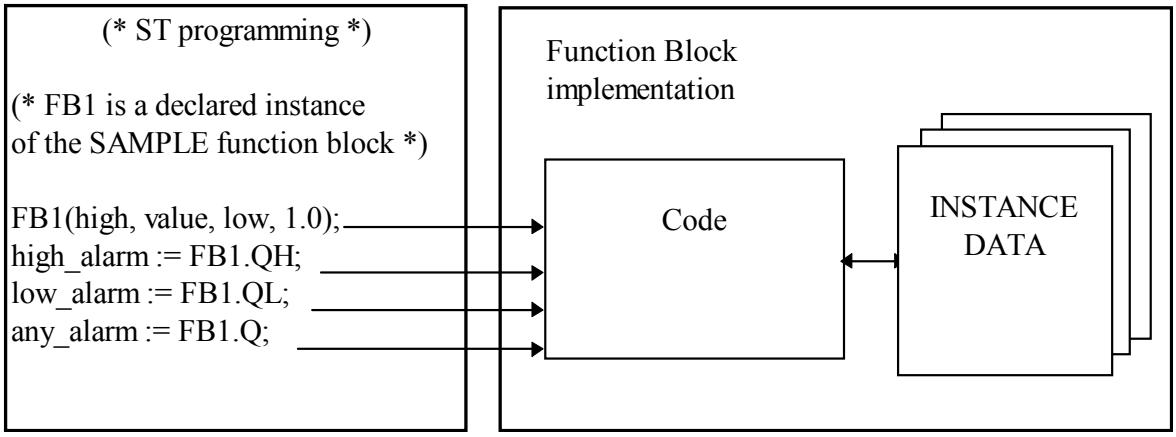


LD: use a coil symbol with the name of the return parameter:



E.1.5 Function blocks

Function blocks can use the languages: LD, FBD, ST or IL. Function blocks are instantiated. It means local variables of a function block are copied for each instance. When calling a block in a program, you actually call the instance of the block: the same code is called but the data used are the one which have been allocated for the instance. Values of the variables of the instance are stored from one cycle to the other.



Warnings:

- A function block written with one of the IEC languages can not call other function blocks: the instantiation mechanism only manages the local variables of the block itself. Here is the list of standard function blocks that you cannot use inside an IEC function block:

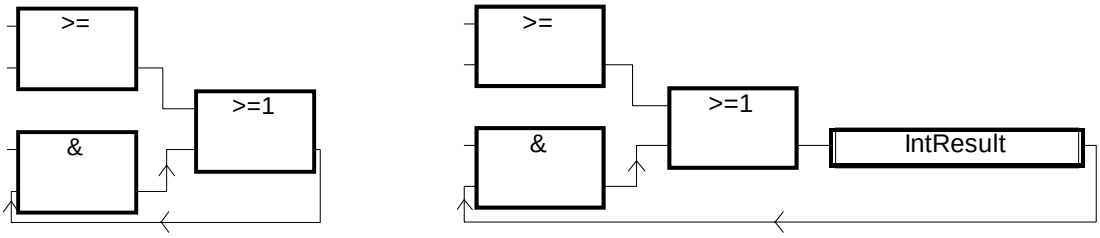
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALRM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

- For the same reason, you can not use Positive or Negative contact or coils, or Set and Reset coils.

- TSTART and TSTOP functions to start and stop timers cannot be used in a function block for 3.0x targets. It works since the 3.20 target.

- When you need loop in your function block, you must use local variable before doing the loop. See the example below:

This will not work: This is OK:



E.1.6 Description language

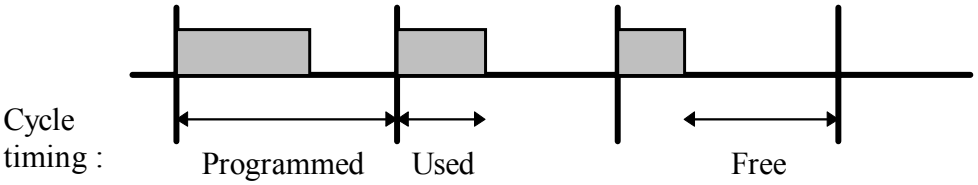
A program can be described with any of the following graphic or literal languages:

Sequential Function Chart (SFC) for high level operations
Flow Chart (FC) for high level operations
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
Instruction List (IL) for low level operations

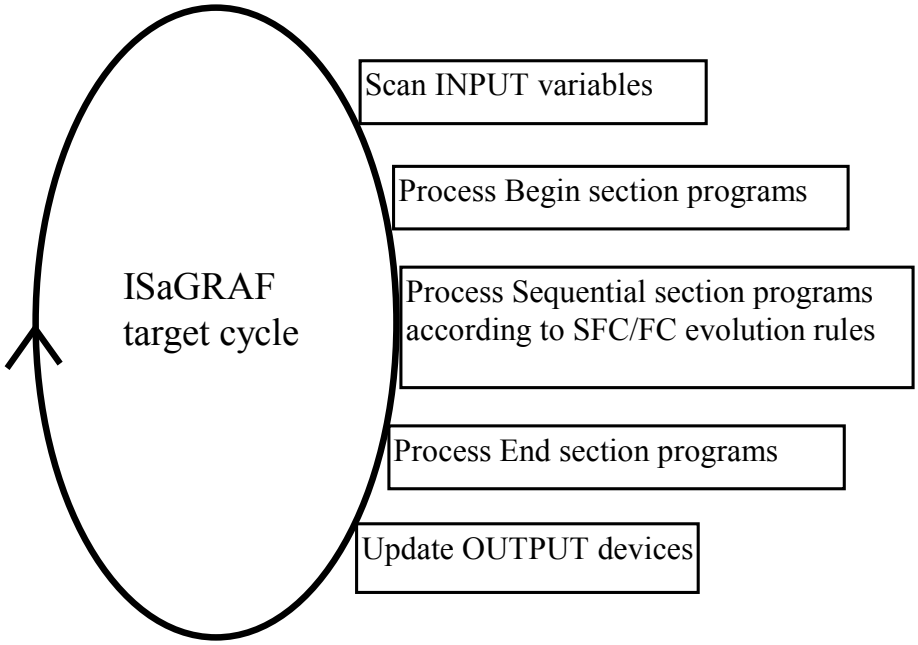
The same program cannot mix several languages. The language used to describe a program is chosen when the program is created, and cannot be changed later on. The exception is that it is possible to combine FBD and LD in a single program.

E.1.7 Execution rules

ISaGRAF is a **synchronous** system. All the operations are triggered by a clock. The basic duration of the clock is called the cycle timing:



Basic operations processed during a target cycle are:



This system makes it possible to:

- guarantee that an input variable keeps the same value within a cycle,
- guarantee that an output device is not updated more than once in a cycle,
- work safely on the same global variable from different programs,
- estimate and control the response time of the complete application.

E.2 Common objects

These are main features and common **objects** of the ISaGRAF programming database. Such objects can be used in any program written with any of the **SFC**, **FC**, **FBD**, **LD**, **ST** or **IL** languages.

E.2.1 Basic types

Any constant, expression or variable used in a program (written in any language) must be characterised by a type. Type coherence must be followed in graphic operations and literal statements. These are the available basic types for programming objects:

BOOLEAN: logic (true or false) value
ANALOG: integer or real (floating) continuous value
TIMER: time value
MESSAGE: character string

Note: Timers contain values less than one day and cannot be used to store dates.

E.2.2 Constant expressions

Constant expressions are relative to one type. The same notation cannot be used to represent constant expressions of different types.

E.2.2.1 Boolean constant expressions

There are only two boolean constant expressions:

TRUE is equivalent to the integer value 1
FALSE is equivalent to the integer value 0

"True" and "False" keywords are case insensitive.

E.2.2.2 Integer analog constant expressions

Integer constant expressions represent signed long integer (32 bit) values: from **-2147483647** to **+2147483647**. Integer analog constants may be expressed with one of the following **bases**. Integer constants must begin with a **prefix** that identifies the bases used:

Base	Prefix	Example
DECIMAL	**(none)**	**-908**
HEXADECIMAL	**"16#"**	**16#1A2B3C4D**
OCTAL	**"8#"**	**8#1756402**
BINARY	**"2#"**	**2#1101_0001_0101_1101**

The underscore character ('_') may be used to separate groups of digits. It has no particular significance, and is used to increase constant expression readability.

E.2.2.3 Real analog constant expressions

Real analog constant expressions can be written with either **decimal** or **scientific** representation. The **decimal point** ('**.**') separates the integer and decimal parts. The decimal point must be used to differentiate a real constant expression from an integer one. The scientific representation uses the '**E**' or '**F**' letter to separate the **mantissa** part and the **exponent**. Exponent part of a real scientific expression must be a signed integer value from **-37** to **+37**. Below are examples of real analog constant expressions:

3.14159	**-1.0E+12**
+1.0	**1.0F-15**
-789.56	**+1.0E-37**

The expression "**123**" does not represent a real constant expression. Its correct real representation is "**123.0**".

E.2.2.4 Timer constant expressions

Timer constant expressions represent time values from **0 second** to **23h59m59s999ms**. The lowest allowed unit is a millisecond. Standard time units used in constant expressions are:

Hour	The "**h**" letter must follow the number of hours
Minute	The "**m**" letter must follow the number of minutes
Second	The "**s**" letter must follow the number of seconds
Millisecond	The "**ms**" letters must follow the number of milliseconds

The time constant expression must begin with "**T#**" or "**TIME#**" prefix. Prefixes and unit letters are case insensitive. Some units may not appear. These are examples of timer constant expressions:

T#1H450MS	1 hour, 450 milliseconds
time#1H3M	1 hour, 3 minutes

The expression "0" does not represent a time value, but an analog constant.

E.2.2.5 Message string constant expressions

String or message constant expressions represent character strings. Characters must be preceded by a quote and followed by an apostrophe. For example:

'THIS IS A MESSAGE'

Warning: The apostrophe "'" character cannot be used within a string constant expression. A string constant expression must be expressed on one line of the program source code. Its length cannot exceed 255 characters, including spaces.

Empty string constant expression is represented by two apostrophes, with no space or tab character between them:

'' (* this is an empty string *)

The special character dollar ('**$**'), followed by other special characters, can be used in a string constant expression to represent a non-printable character:

Sequence	Meaning	ASCII (hexa)	Example
$$	'$' character	16#24	'I paid $$5 for this'
$'	apostrophe	16#27	'Enter $'Y$' for YES'
$L	line feed	16#0a	'next $L line'
$R	carriage return	16#0d	' llo $R He'
$N	new line	16#0d 0a	'This is a line$N'
$P	new page	16#0c	'lastline $P first line'
$T	tabulation	16#09	'name$Tsize$Tdate'
$hh (*)	any character	16#hh	'ABCD = $41$42$43$44'

(*) "**hh**" is the hexadecimal value of the ASCII code for the expressed character.

E.2.3 Variables

Variables can be **LOCAL** to one program, or **GLOBAL**. Local variables can be used by one program only. Global variables can be used in any program of the project. Variable names must conform to the following rules:

name cannot exceed **16** characters
first character must be a **letter**
following characters can be **letters**, **digits** or the underscore character

E.2.3.1 Reserved keywords

A list of the reserved keywords is shown below. Such identifiers cannot be used to name a program, a variable or a "C" function or function block:

A ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,

C CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
E ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F FALSE, FEDGE, FIND, FOR, FUNCTION,
G GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
I IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
J JMP, JMPC, JMPCN, JMPN, JMPNC,
L LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
M MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
N NE, NOT,
O OF, ON, OPERATE, OR, OR_MASK, ORN,
P PROGRAM
R R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR,
S S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,
T TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE,
U UDINT, UINT, ULINT, UNTIL, USINT,
V VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, ,VAR_OUTPUT,
W WHILE, WITH, WORD,
X XOR, XOR_MASK, XORN

All keywords beginning with an underscore ('_') character are internal keywords and must not be used in textual instructions.

E.2.3.2 Directly represented variables

ISaGRAF enables the use of **directly represented variables** in the source of the programs to represent a free channel. Free channels are the ones which are not linked to a declared I/O variable. The identifier of a directly represented variable always begins with "**%**" character.

Below are the naming conventions of a directly represented variable for a channel of a single board. "***s***" is the slot number of the board. "***c***" is the number of the channel.

%IX*s.c*	free channel of a boolean input board
%ID*s.c*	free channel of an integer input board
%IS*s.c*	free channel of a message input board
%QX*s.c*	free channel of a boolean output board
%QD*s.c*	free channel of an integer output board
%QS*s.c*	free channel of a message output board

Below are the naming conventions of a directly represented variable for a channel of a complex equipment. "***s***" is the slot number of the equipment. "***b***" is the index of the single board within the complex equipment. "***c***" is the number of the channel.

%IX*s.b.c*	free channel of a boolean input board
%ID*s.b.c*	free channel of an integer input board
%IS*s.b.c*	free channel of a message input board
%QX*s.b.c*	free channel of a boolean output board
%QD*s.b.c*	free channel of an integer output board
%QS*s.b.c*	free channel of a message output board

Below are examples:

%QX1.6	6th channel of the board #1 (boolean output)
%ID2.1.7	7th channel of the board #1 in the equipment #2 (integer input)

A directly represented variable cannot have the "**real**" data type.

E.2.3.3 Boolean variables

Boolean means **logic**. Such variables can take one of the boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in boolean expressions. Boolean variables can have one of the following **attributes**:

Internal: memory variable updated by the program
Constant: read-only memory variable with an initial value
Input: variable connected to an input device (refreshed by the system)
Output: variable connected to an output device

Warning: When declaring a boolean variable, strings can be defined to replace 'true' and 'false' values during debug. Those strings cannot be used in the programs unless entered as '**defined words**' for the language.

E.2.3.4 Analog variables

Analog means **continuous**. Such variables have signed integer or real (floating) values. Available formats for an analog variable are:

Integer 32 bit signed integer: from **-2147483647** to **+2147483647**
Real standard IEEE 32 bit floating value (single precision)
1 sign bit + 23 mantissa bits + 8 exponent bits

REAL analog exponent value cannot be less than **-37** or greater than **+37**. Analog variables can have one of the following **attributes**:

Internal memory variable updated by the program
Constant: read-only memory variable with an initial value
Input variable connected to an input device (refreshed by the system)

Output variable connected to an output device

Note: When a real variable is connected to an I/O device, the corresponding I/O driver operates the equivalent integer value.

Warning: Integer and real analog variables or constant expressions cannot be mixed in the same analog expression.

E.2.3.5 Timer variables

Timer means **clock** or **counter**. Such variables have time values and are typically used in time expressions. A timer value cannot exceed **23h59m59s999ms** and cannot be negative. Timer variables are stored in 32 bit words. The internal representation is a positive number of milliseconds.
Timer variables can have one of the following **attributes**:

Internal memory variable managed by the program, refreshed by ISaGRAF system
Constant: read-only memory variable with an initial value

Warning: Timer variables cannot have the INPUT or OUTPUT attributes.

Timer variables can be automatically refreshed by the ISaGRAF system. When a timer is **active**, its value is automatically increased according to the target system real time clock. The following statements of the **ST** language can be used to control a timer:

TSTART starts automatic refresh of a timer
TSTOP stops automatic refresh of a timer

E.2.3.6 Message string variables

Message or string variables contain character strings. The length of the string can change during process operations. The length of a message variable cannot exceed the capacity (maximum length) specified when the variable is declared. Message capacity is limited to 255 characters. Message variables can have one of the following **attributes**:

Internal memory variable updated by the program
Constant: read-only memory variable with an initial value
Input variable connected to an input device (refreshed by the system)
Output variable connected to an output device

String variables can contain any character of the standard ASCII table (ASCII code from **0** to **255**). The null character can exist in a character string. Some "C" functions of the standard ISaGRAF library will not correctly operate messages which contain null (**0**) characters.

E.2.4 Comments

Comments may be freely inserted in literal languages such as **ST** and **IL**. A comment must begin with the special characters "**(***" and terminate with the characters "***)**". Comments can be inserted anywhere in a **ST** program, and can be written on more than one line.

These are examples of comments:

```
counter := ivalue;   (* assigns the main counter *)
(* this is a comment expressed
on two lines *)
c := counter (* you can put comments anywhere *) + base_value + 1;
```

Interleave comments cannot be used. This means that the "**(***" characters cannot be used within a comment.

Warning: The IL language only accepts comments as the last component of an instruction line.

E.2.5 Defined words

The ISaGRAF system allows the re-definition of constant expressions, true and false boolean expressions, keywords or complex **ST** expressions. To achieve this, an **identifier** name has to be given to the corresponding expression. For example:

```
YES     is   TRUE
PI      is   3.14159
OK      is   (auto_mode AND NOT (alarm))
```

When such equivalence is defined, its **identifier** can be used anywhere in an **ST** program to replace the attached expression. This is an example of **ST** programming using defines:

```
If OK Then
  angle := PI / 2.0;
  isdone := YES;
End_if;
```

Defined words can be **LOCAL** to one program, **GLOBAL**, or **COMMON**.
Local defined words can be used by only one program.
Global defined words can be used in any program of the project.
Common defined words can be used in any program of any project.
Note that common defined can be stored separately with the Archive manager.

Warning: When the same identifier is defined twice with different **ST** equivalencies, the last defined expression is used. For example:

```
Define:     OPEN     is     FALSE
    OPEN    is     TRUE

means:      OPEN     is     TRUE
```

Naming defined words must conform to following rules:
- name cannot exceed **16** characters
- first character must be a **letter**

- following characters can be **letters**, **digits** or underscore ('_') character

Warning: A defined word can not use a defined word in its definition, for example, you can not have:

PI is **3.14159**
PI2 is **PI*2**

write the complete equivalence using constants or variables and operations:

PI2 is **6.28318**

E.3 SFC language

Sequential Function Chart (SFC) is a **graphic** language used to describe **sequential operations**. The process is represented as a set of well-defined **steps**, linked by **transitions**. A **boolean condition** is attached to each transition. **Actions** within the steps are detailed by using other languages (**ST**, **IL**, **LD** and **FDB**).

E.3.1 SFC chart main format

An SFC program is a graphic set of **steps** and **transitions**, linked together by **oriented links**. Multiple connection links are used to represent divergences and convergences. Some parts of the complete program may be separated and represented in the main chart by a single symbol, called **macro steps**. The basic **graphic rules** of the SFC are:

- A step cannot be followed by another step

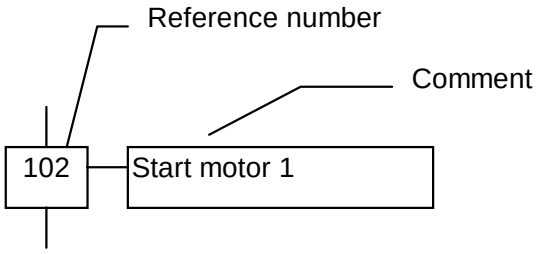
A transition cannot be followed by another transition

E.3.2 SFC basic components

The basic components (graphic symbols) of the SFC language are: steps and initial steps, transitions, oriented links, and jumps to a step.

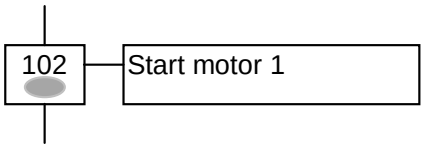
E.3.2.1 Steps and initial steps

A step is represented by a single **square**. Each step is **referenced** by a number, written in the step square symbol. A main description of the step is written in a rectangle linked to the step symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the step:

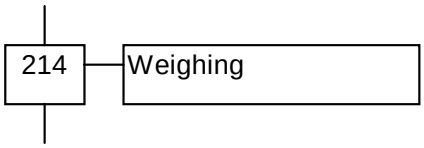


At run time, a **token** indicates that the step is **active**:

Active step:

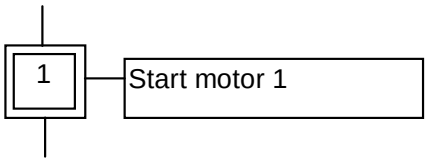


Inactive step:



The **initial situation** of an SFC program is expressed with **initial steps**. An initial step has a **double-bordered** graphic symbol. A token is automatically placed in each initial step when the program is started.

Initial step:

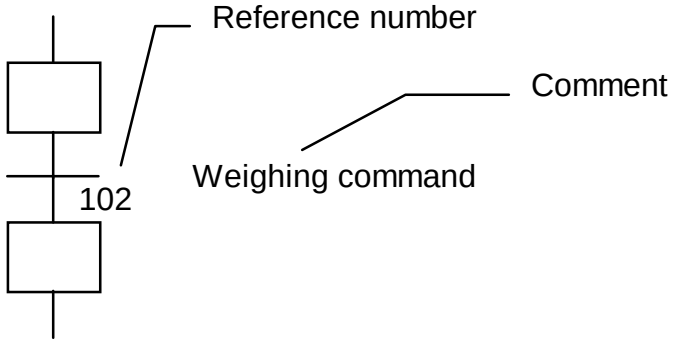


An SFC program must contain **at least one** initial step.

These are the attributes of a step. Such fields may be used in any of the other languages:
GSnnn.x ..... activity of the step (boolean value)
GSnnn.t ..... activation duration of the step (time value)
(where **nnn** is the reference number of the step)

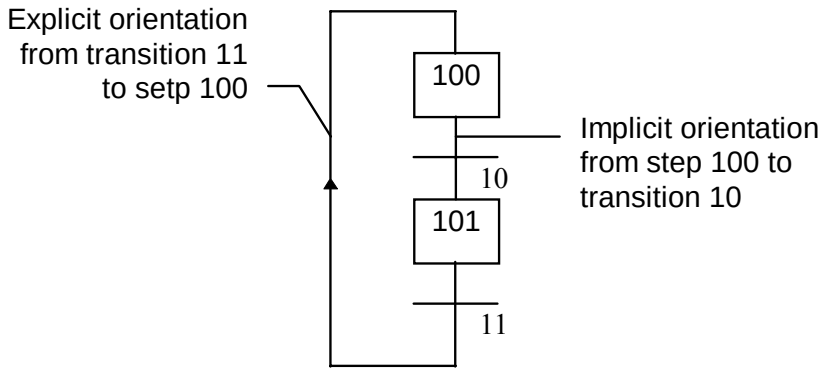
E.3.2.2 Transitions

Transitions are represented by a small horizontal bar that crosses the connection link. Each transition is **referenced** by a number, written next to the transition symbol. A main description of the transition is written on the right side of the transition symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the transition:



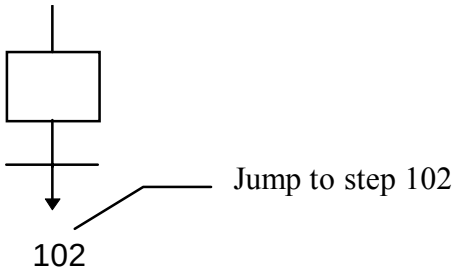
E.3.2.3 Oriented links

Single lines are used to link steps and transitions. These are oriented links. When the orientation is not explicitly given, the link is oriented from the top to the bottom.

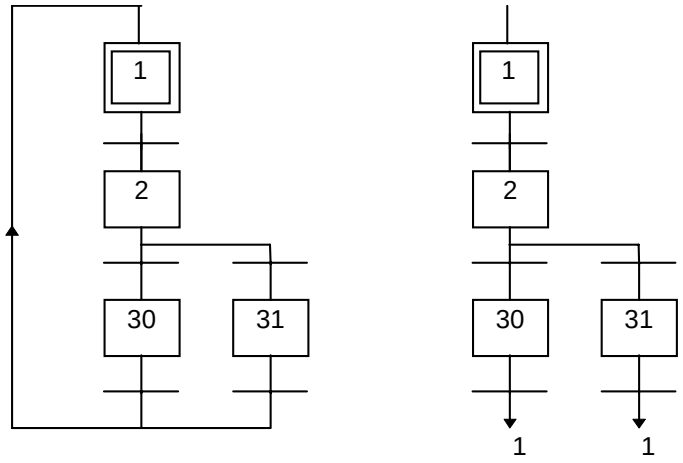


E.3.2.4 Jump to a step

Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the number of the destination step:



A jump symbol cannot be used to represent a link from a step to a transition. Example of jumps - the following charts are equivalent:

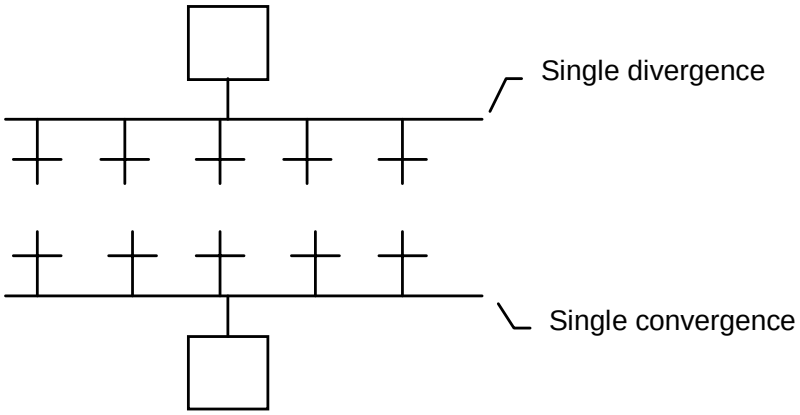


E.3.3 Divergences and convergences

Divergences are **multiple connection links** from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

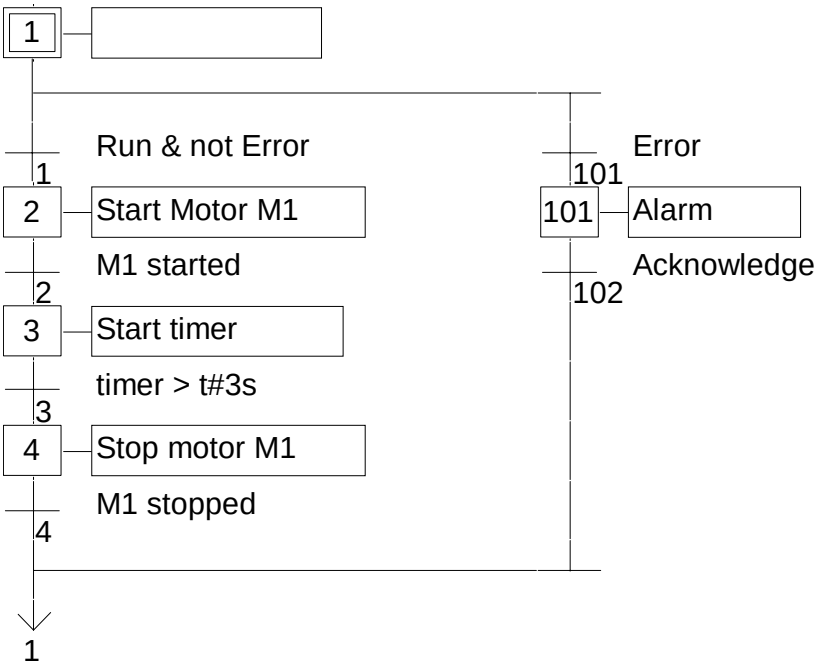
E.3.3.1 single divergences

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence. Single divergences and convergences are represented by single horizontal lines.



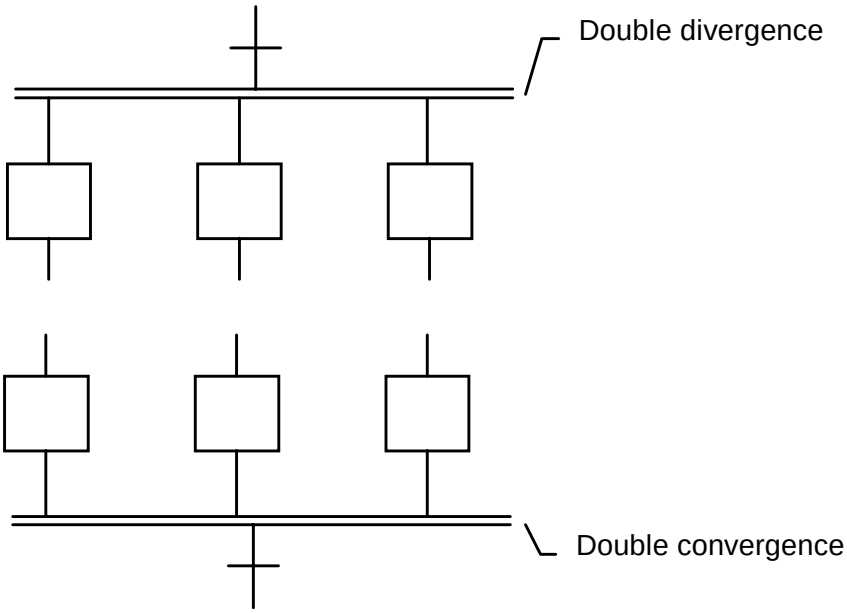
Warning: The conditions attached to the different transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of the transitions to ensure that only one token progresses in one branch of the divergence at run time. Below is an example of single divergence and convergence:

(* SFC program with single divergence and convergence *)



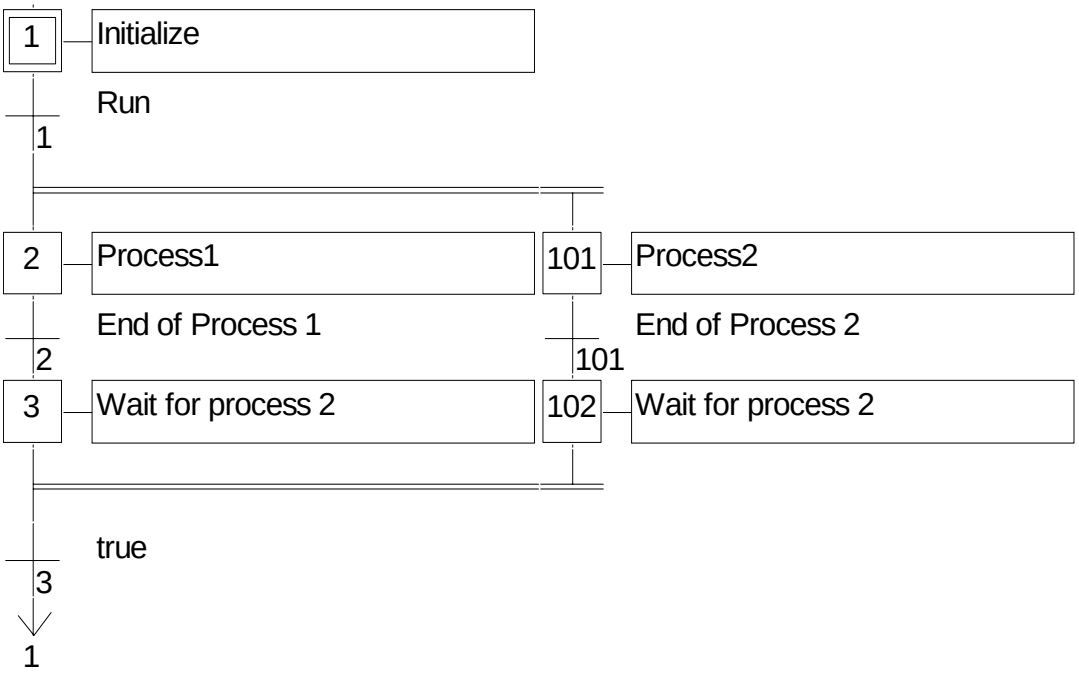
E.3.3.2 Double divergences

A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence. Double divergences and convergences are represented by double horizontal lines.



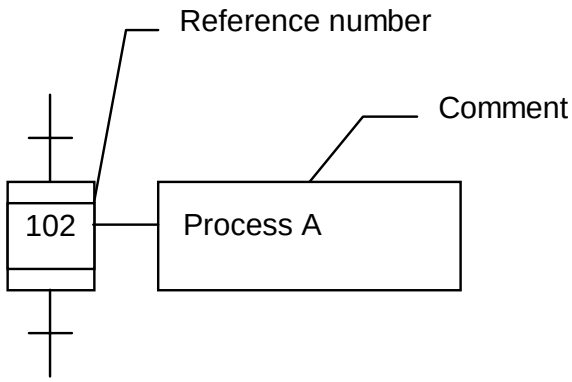
Example of double divergence and convergence:

(* SFC program with double divergence and convergence *)



E.3.4 Macro steps

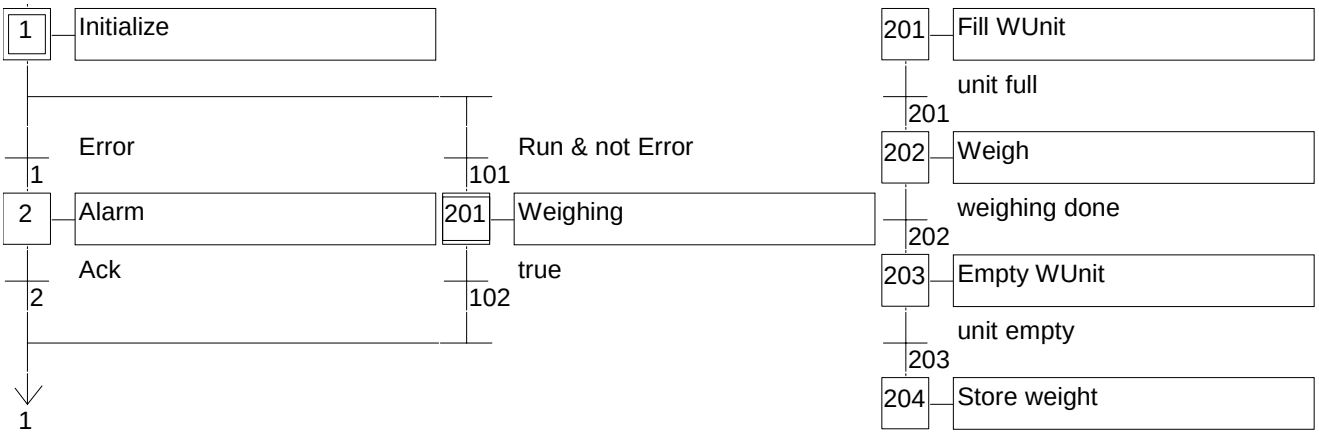
A macro step is a unique representation of a unique group of steps and transitions. The body of the macro step is described separately, elsewhere in the same SFC program. It appears as a single symbol in the main SFC chart. This is the symbol used for a macro step:



The reference number written in the macro step symbol is the reference number of the first step in the body of the macro step. The macro step body must begin with a **beginning step** and terminate with an **ending step**. The chart must be self-contained. A beginning step has no upper link (no backward transition). An ending step has no lower link (no forward transition). A macro step symbol may be put in the body of another macro step.

Warning: Because macro step is a **unique** set of steps and transitions, the same macro step cannot be used more than once in an SFC program.

Example of macro step:
(* SFC program with macro step *)
(* Main chart *) (* Body of the macro step *)



E.3.5 Actions within the steps

The **level 2** of an SFC step is the detailed description of the **actions** executed **during the step activity**. This description is made by using **SFC literal features**, and other languages such as Structured Text (**ST**). The basic types of actions are:
- Boolean actions
- Pulse actions programmed in ST
- Non-stored actions programmed in ST
- SFC actions

Several actions (with same or different types) can be described in the same step. The special features that enable the use of any of the other languages are:
- Calling sub-programs
- Instruction List (IL) language convention

E.3.5.1 Boolean actions

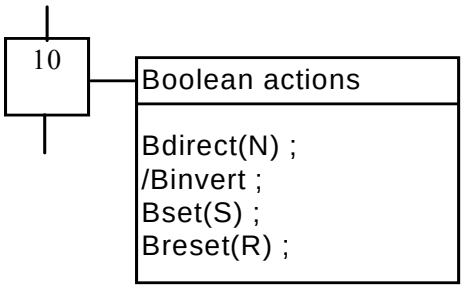
Boolean actions assign a boolean variable with the activity of the step. The boolean variable can be an output or an internal. It is assigned each time the step activity starts or stops. This is the syntax of the basic boolean actions:

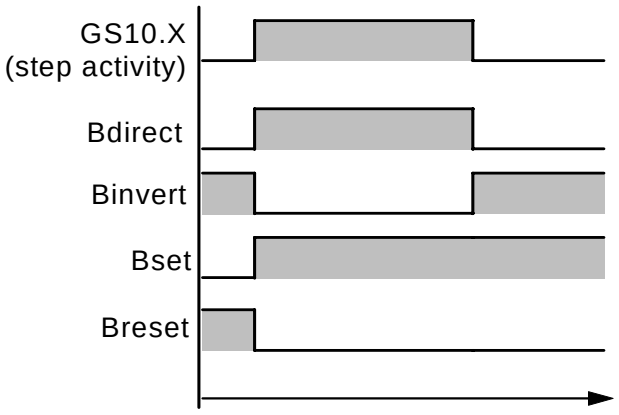
<boolean_variable> (N) ;	assigns the step activity signal to the variable
<boolean_variable> ;	same effect (N attribute is optional)
/ <boolean_variable> ;	assigns the negation of the step activity signal to the variable

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

<boolean_variable> (S) ;	sets the variable to TRUE when the step activity signal becomes TRUE
<boolean_variable> (R) ;	resets the variable to FALSE when the step activity signal becomes TRUE

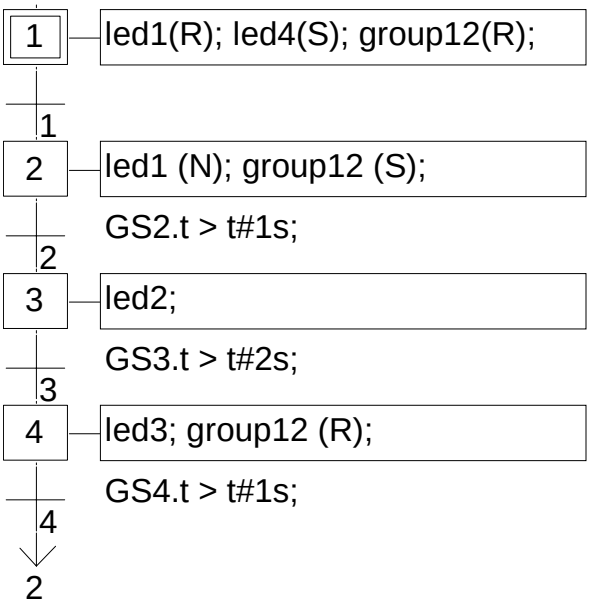
The boolean variable must be an OUTPUT or an INTERNAL. The following SFC programming leads to the following behaviour:





Example of boolean actions:

(* SFC program using BOOLEAN actions *)

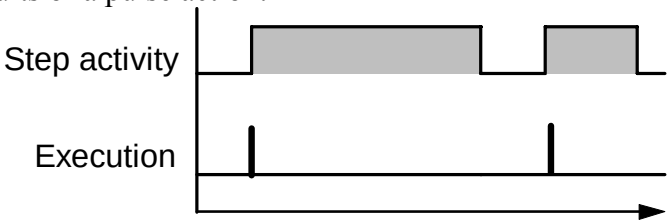


E.3.5.2 Pulse actions

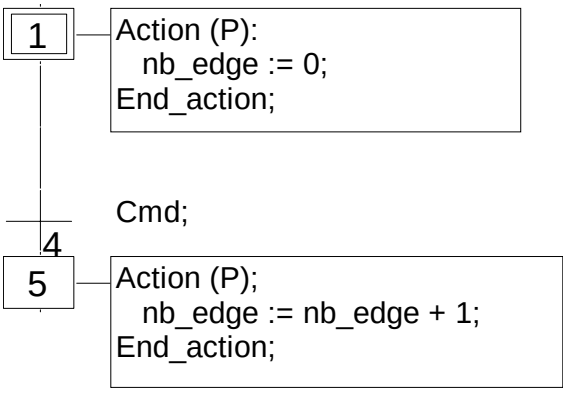
A pulse action is a list of ST or IL instructions, which are executed only **once** at the **activation** of the step. Instructions are written according to the following SFC syntax:

ACTION (P) :
(* ST statements *)
END_ACTION ;

The following shows the results of a pulse action:



Example of pulse action:

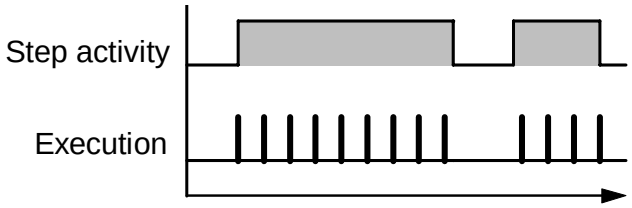


E.3.5.3 Non-stored actions

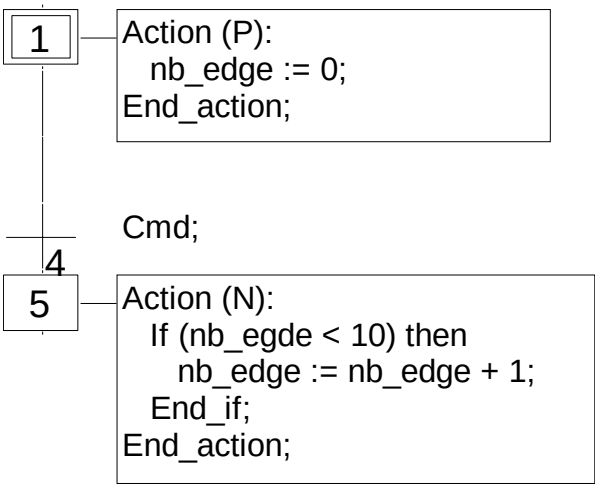
A non-stored (normal) action is a list of ST or IL instructions which are executed **at each cycle** during the whole **active** period of the step. Instructions are written according to the following SFC syntax:

```
ACTION (N) :
  (* ST statements *)
END_ACTION ;
```

The following is the results of a non-stored action:



Example of non-stored action:



E.3.5.4 SFC actions

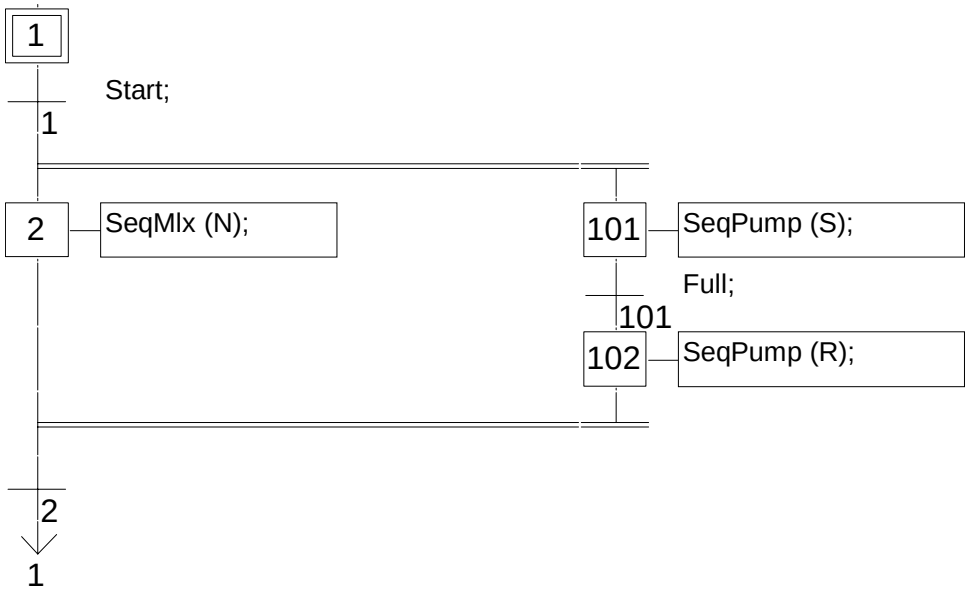
An SFC action is a child SFC sequence, started or killed according to the change of the step activity signal. An SFC action can have the **N** (Non stored), **S** (Set), or **R** (Reset) qualifier. This is the syntax of the basic SFC actions:

<child_prog> (N); starts the child sequence when the step becomes active, and kills the child sequence when the step becomes inactive
<child_prog> ; same effect (N attribute is optional)
<child_prog> (S); starts the child sequence when the step becomes active. Nothing is done when the step becomes inactive
<child_prog> (R); kills the child sequence when the step becomes active. Nothing is done when the step becomes inactive

The SFC sequence specified as an action must be a **child SFC program** of the program currently being edited. Note that using the **S** (Set) or **R** (Reset) qualifiers for an SFC action has exactly the same effect as the **GSTART** and **GKILL** statements, programmed in an **ST** pulse action.

Below is an example of an SFC action. The main SFC program is named **Father**. It has two SFC children, called **SeqMlx** and **SeqPump**. The SFC programming of the father SFC program is:

(* SFC program using SFC actions *)



E.3.5.5 Calling function and function blocks from an action

Sub-programs, functions or function blocks (written in ST, IL, LD or FBD language) or "C" functions and "C" function blocks, can be directly called from an SFC action block, based on the following syntax:

For sub-programs, functions and "C" functions:

```
ACTION (P) :
  result := sub_program ( ) ;
END_ACTION;
```

or

```
ACTION (N) :
  result := sub_program ( ) ;
END_ACTION;
```

For function blocks in "C" or in ST, IL, LD, FBD:

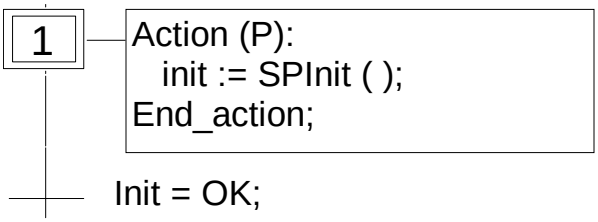
```
ACTION (P) :
  Fbinst(in1, in2);
  result1 := Fbinst.out1;
  result2 := Fbinst.out2;
END_ACTION;
```

or

```
ACTION (N) :
  Fbinst(in1, in2);
  result1 := Fbinst.out1;
  result2 := Fbinst.out2;
END_ACTION;
```

Detailed syntax can be found in the ST language section.
Example of a sub-program call in action blocks:

(* SFC program with a sub-program call in an action block *)



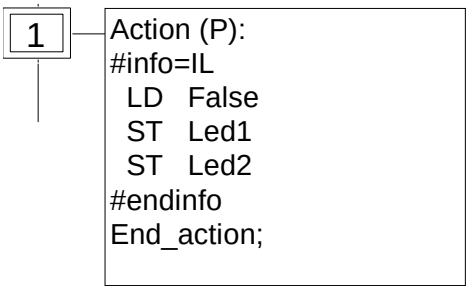
E.3.5.6 IL convention

Instruction List (IL) programming may be directly entered in an SFC action block, based on the following syntax:

```
ACTION (P) :  (* or N *)
#info=IL
    <instruction>
    <instruction>
    ....
#endinfo
END_ACTION;
```

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords. Below is an example of an IL program in an action block:

(* SFC program with an IL sequence in an action block *)



E.3.6 Conditions attached to transitions

At each transition, a **boolean expression** is attached that conditions the clearing of the transition. The condition is usually expressed with ST language or using the LD language (Quick LD editor). This is the **Level 2** of the transition. Other structures may, however, be used:

- ST language convention
- LD language convention
- IL language convention
- Calling function from a transition

Warning: When no expression is attached to the transition, the default condition is **TRUE**.

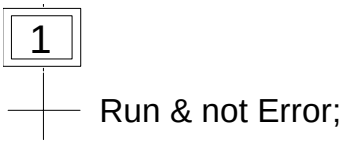
E.3.6.1 ST convention

The **Structured Text** (ST) language can be used to describe the **condition** attached to a transition. The complete expression must have **boolean** type and must be terminated by a **semicolon**, according to the following syntax:

< boolean_expression > ;

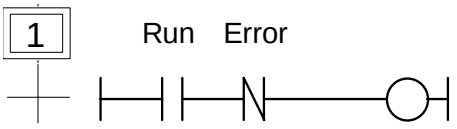
The expression may be a TRUE or FALSE constant expression, a single input or an internal boolean variable, or a combination of variables that leads to a boolean value. Below is an example of ST programming for transitions:

(* SFC program with ST programming for transitions *)



E.3.6.2 LD convention

The **Ladder Diagram** (LD) language can be used to describe the **condition** attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transition value. Below is an example of LD programming for transitions:



E.3.6.3 IL convention

Instruction List (IL) programming may be directly used to describe an SFC transition, according to the following syntax:

#info=IL

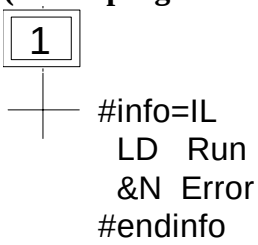
<instruction>
<instruction>
....
#endinfo

The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the transition:

current result = 0 ➔ condition is **FALSE**
current result <> 0 ➔ condition is **TRUE**

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords. Below is an example of IL programming for transitions:

(* SFC program with an IL program for transitions *)



E.3.6.4 Calling functions from a transition

Any sub-program or a function (written in FBD, LD, ST or IL language), or a "C" function can be called to evaluate the condition attached to a transition, according to the following syntax:

< sub_program > ( ) ;

The value returned by the sub-program or the function must be boolean and yields the resulting condition:

return value = FALSE ➔ condition is **FALSE**
return value = TRUE ➔ condition is **TRUE**

Example of a sub-program called in a transition:

(* SFC program with sub-program call for transitions *)

E.3.7 SFC dynamic rules

The **five** dynamic rules of the SFC language are:

Initial situation

The initial situation is characterised by the **initial steps** which are, by definition, in the active state at the beginning of the operation. **At least one** initial step must be present in each SFC program.

Clearing of a transition

A transition is either **enabled** or **disabled**. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are **active**, otherwise it is disabled. A transition cannot be **cleared** unless:

- it is enabled, and
- the associated transition condition is true.

Changing of state of active steps

The clearing of a transition simultaneously leads to the active state of the immediately following steps and to the inactive state of the immediately preceding steps.

Simultaneous clearing of transitions

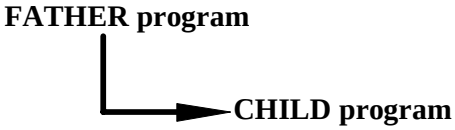
Double lines may be used to indicate transitions which have to be cleared simultaneously. If such transitions are shown separately, the activity state of preceding steps (GSnnn.x) can be used to express their conditions.

Simultaneous activation and deactivation of a step

If, during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

E.3.8 SFC program hierarchy

The ISaGRAF system enables the description of the vertical structure of SFC programs. SFC programs are organised in a **hierarchy tree**. Each SFC program can control (start, kill...) other SFC programs. Such programs are called **children** of the SFC program which controls them. SFC programs are linked together into a main **hierarchy tree**, using a "**father - child**" relation:



The basic rules implied by the hierarchy structure are:
- SFC programs which have no father are called "**main**" SFC programs
- Main SFC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can only be controlled by its father
- A program cannot control the children of one of its own children

The basic actions that a father SFC program can take to control its child program are:

Start (**GSTART**) Starts the child program: activates each of its initial steps. Children of this child program are not automatically started.

Kill (**GKILL**) Kills the child program by deactivating each of its active steps. All the children of the child program are also killed.

Freeze (**GFREEZE**) Suspends the execution of the program (deactivates actions of each of the active steps and suspend transition calculation), and memorises the status of the program steps so the program can be restarted. All the children of the child program are also frozen.

Restart (**GRST**) Restarts a frozen SFC program by reactivating all the suspended steps. Children of the program are not automatically restarted.

Get status (**GSTATUS**) Gets the current status (active, inactive or frozen) of a child program.

E.4 Flow Chart language

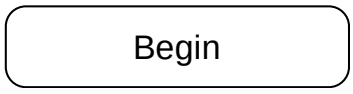
Flow Chart (**FC**) is a graphic language used to describe **sequential operations**. A Flow Chart diagram is composed of **Actions** and **Tests**. Between Actions and test are **oriented links** representing data flow. Multiple connection links are used to represents divergences and convergences. Actions and Tests can be described with ST, LD or IL languages. Functions and Function blocks of any language (except SFC) can be called from actions and tests. A Flow Chart program can call another Flow Chart program. The called FC program is a **sub-program** of the calling FC program.

E.4.1 FC components

Below are graphic components of the Flow Chart language:

- ***Beginning of FC chart***

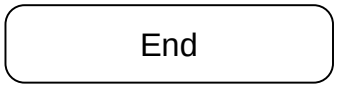
A "**begin**" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated. Below is the drawing of a "begin" symbol:



The "Begin" symbol always has a connection (on the bottom) to the other objects of the chart. A flow chart is not valid if no connection is drawn from "Begin" to another object.

- ***Ending of FC chart***

An "**end**" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It is possible that no connection is drawn to the "End" symbol (always looping chart), but "End" symbol is still drawn anyway at the bottom of the chart. It represents the final state of the chart, when its execution has been completed. Below is the drawing of an "end" symbol:



The "End" symbol generally has a connection (on the top) to the other objects of the chart. A flow chart may have no connection to the "End" object (always looping chart). The "End" object is still visible at the bottom of the chart in this case.

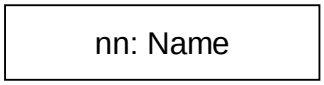
- ***FC flow links***

A flow **link** is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Below is the drawing of a flow link:

Two links cannot start from the same source connection point.

- ***FC actions***

An **action** symbol represents actions to be performed. An action is identified by a number and a name. Below is the drawing of an "action" symbol:



Two different objects of the same chart cannot have the same name or logical number. Programming language for an action can be ST, LD or IL. An action is always connected with links, one arriving to it, one starting from it.

- ***FC conditions***

A **condition** represents a boolean **test**. A condition is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. Below are the possible drawings for a condition symbol:

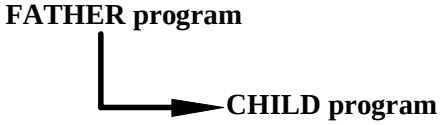
Two different objects of the same chart cannot have the same name or logical number. The programming of a test is either
- an expression in ST, or
- a single rung in LD, with no symbol attached to the unique coil, or
- several instructions in IL. The IL register (or current result) is used to evaluate the condition.

When programmed in ST text, the expression may optionally be followed by a semicolon. When programmed in LD, the unique coil represents the condition value. A condition equal to:
- 0 or FALSE directs the flow to NO
- 1 or TRUE directs the flow to YES

A test is always connected with an arriving link, and both forward connections must be defined.

- ***FC sub-program***

The system enables the description of the vertical structure of FC programs. FC programs are organised in a **hierarchy tree**. Each FC program can call other FC programs. Such a program is called a **child program** of the FC program which calls them. FC programs which call FC sub-programs are called **father program**. FC programs are linked together into a main hierarchy tree, using a "father - child" relation:



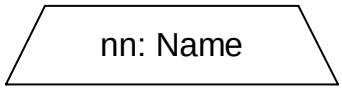
A **sub-program** symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete. A Flow Chart sub-program is identified by a number and a name, as other programs, functions or function blocks. Below is the drawing of a "sub-program call" symbol:

Two different objects of the same chart cannot have the same logical number. The basic rules implied by the FC hierarchy structure are:
- FC programs which have no father are called main FC programs.
- Main FC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can be called only by its father
- A program cannot call the children of one of its own children

The same sub-program may appear several times in the father chart. A Flow Chart sub-program call represents the complete execution of the sub chart. The father chart execution is suspended during the child chart is performed. The sub-program calling blocks must follow the same connection rules as the ones defined for action.

- ***FC I/O specific action***

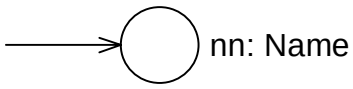
An **I/O specific action** symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. Below is the drawing of an "I/O specific action" symbol:



I/O specific blocks have exactly the same behaviour as standard actions. This covers their properties, ST, LD or IL programming, and connection rules.

- ***FC connectors***

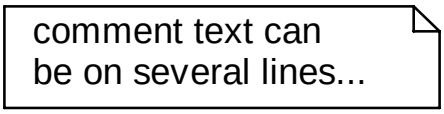
Connectors are used to represent a link between two points of the diagram without drawing it. A connector is represented as a circle and is connected to the source of the flow. The drawing of the connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). Below is the standard drawing of a connector:



A connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

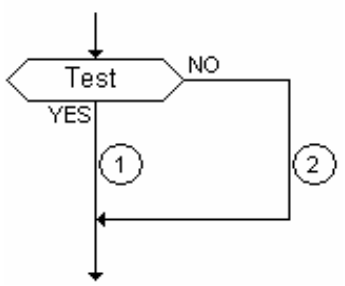
- ***FC comments***

A **comment** block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on an unused space of the Flow Chart document window, and is used to document the program. Below is the drawing of a "comment" symbol:



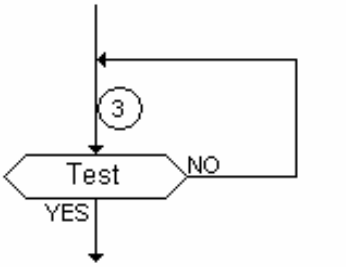
E.4.2 FC complex structures

This section shows **complex structure** examples that can be defined in a Flow Chart diagram. Such structures are combinations of basic objects linked together.



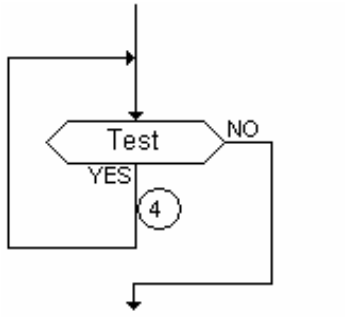
IF / THEN / ELSE

(1) place for "THEN" actions to be inserted
(2) place for "ELSE" actions to be inserted



REPEAT / UNTIL

(3) place for repeated actions to be inserted



WHILE / DO

(3) place for repeated actions to be inserted

E.4.3 FC dynamic behaviour

The **execution** of a Flow Chart diagram can be explained as follows:

- The Begin symbol takes one target cycle
- The End symbol takes one target cycle and ends the execution of the chart. After this symbol is reached, no more actions of the chart are executed.

- The flow is broken each time an item (action, decision) is encountered that has already been reached in the same cycle. In such a case the flow will continue on the next cycle.

Note: Contrary to SFC, an action is not a stable state. There is no repetition of instructions while the action symbol is highlighted.

E.4.4 FC checking

Apart of attached ST, LD or IL programming, some other **syntactic rules** apply to flow chart itself. Below is the list of main rules:

- All "connection" points of all symbols must be wired. (connection to "End" symbol may be omitted)
- All symbols must be linked together (no isolated part should appear)
- All connectors should have valid destination

Other minor syntax errors can be reported:

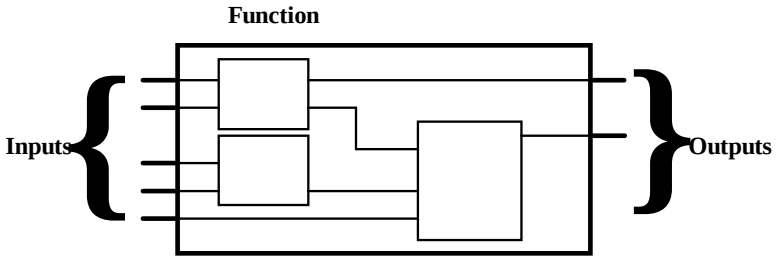
- Empty actions (no programming) are considered as steps during run time scheduling
- Empty tests (no programming) are considered as "always true"

E.5 FBD language

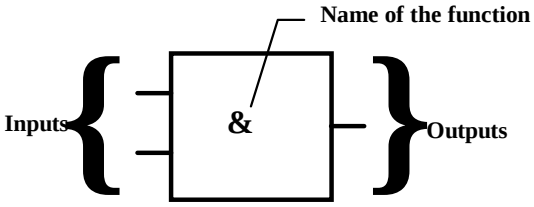
The **Functional Block Diagram** (FBD) is a graphic language. It allows the programmer to build complex procedures by taking existing **functions** from the ISaGRAF library and **wiring** them together in the graphic diagram area.

E.5.1 FBD diagram main format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of **elementary function blocks**. Input and output variables are connected to blocks by **connection lines**. An output of a function block may also be connected to an input of another block.



An entire function operated by an FBD program is built with standard **elementary** function blocks from the ISaGRAF library. Each function block has a fixed number of **input connection points** and a fixed number of **output connection points**. A function block is represented by a single **rectangle**. The inputs are connected on its **left** border. The outputs are connected on its **right** border. An elementary function block performs a single **function** between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well-defined **type**.



Input variables of an FBD program must be connected to input connection points of function blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a **constant** expression, any **internal** or **input** variable, or an **output** variable.

Output variables of an FBD program must be connected to output connection points of function blocks. The type of each variable must be the same as the type expected for the associated block output. An Output for FBD diagram can be any **internal** or **output** variable, or the name of the program (for **sub-programs** only). When an output is the name of the currently edited sub-program, it represents the assignment of the return value for the sub-program (returned to the calling program).

Input and output variables, inputs and outputs of the function blocks are wired together with **connection lines**. Single lines may be used to **connect** two logical points of the diagram:

- An input variable and an input of a function block
- An output of a function block and an input of another block
- An output of a function block and an output variable

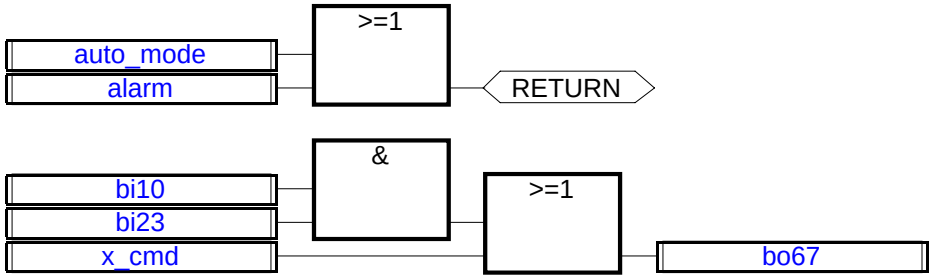
The connection is **oriented**, meaning that the line carries associated data from the left extremity to the right extremity. The left and right extremities of the connection line must be of the **same type**.

Multiple right connection can be used to broadcast an information from its left extremity to each of its right extremities. All the extremities of the connection must be of the same type.

E.5.2 RETURN statement

The "**<RETURN>**" keyword may occur as a diagram output. It must be connected to a boolean output connection point of a function block. The RETURN statement represents a **conditional end** of the program: if the output of the box connected to the statement has the boolean value **TRUE**, the end (remaining part) of the diagram is not executed.

(* Example of an FBD program using RETURN statement *)



(* ST equivalence: *)
If auto_mode OR alarm Then
    Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;

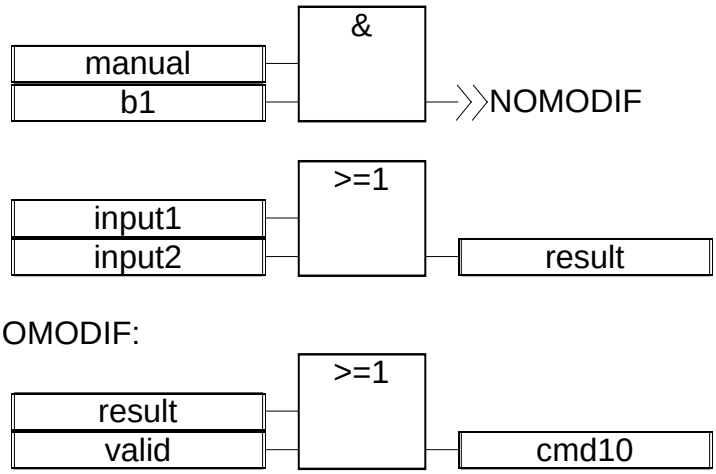
E.5.3 Jumps and labels

Labels and jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notations are used:

>>LAB.....jump to a label (label name is "LAB")
LAB:.....definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the boolean state **TRUE**, the execution of the program directly jumps after the corresponding label symbol.

(* Example of an FBD program using labels and jumps *)

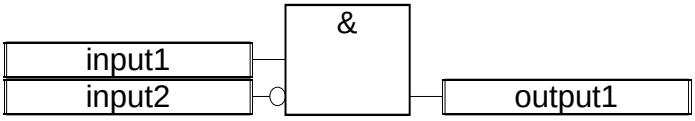



```
(* IL Equivalence: *)
      ld          manual
      and         b1
      jmpc        NOMODIF
      ld          input1
      or          input2
      st          result
NOMODIF:  ld      result
      or          valid
      st          cmd10
```

E.5.4 Boolean negation

A single connection line with its right extremity connected to an input of a function block can be terminated by a **boolean negation**. The negation is represented by a small circle. When a boolean negation is used, the left and right extremities of the connection line must have the **BOOLEAN** type.

(* Example of an FBD program using a boolean negation *)



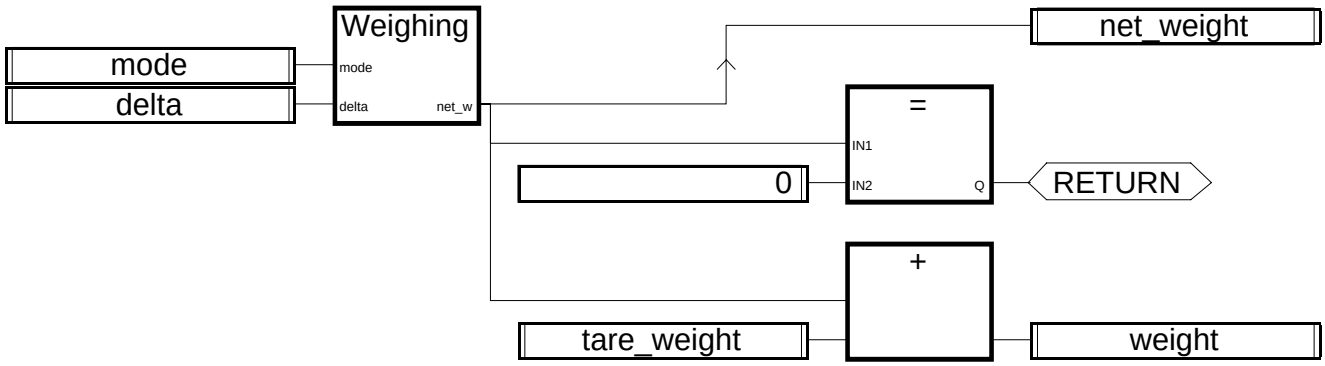

```
(* ST equivalence: *)
output1 := input1 AND NOT (input2);
```

E.5.5 Calling function or function blocks from the FBD

The FBD language enables the calling of sub-programs, functions or function blocks. A sub-program, or function or function block is represented by a function box. The name written in the box is the name of the sub-program or function or function blocks.
In case of a sub-program or a function, the return value is the only output of the function box.

A function block can have more than one output.

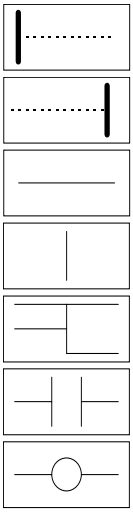
(* Example of an FBD program using SUB PROGRAM block *)



(* ST Equivalence *)
net_weight := Weighing (mode, delta); (* call sub-program *)
If (net_weight = 0) Then Return; End_if;
weight := net_weight + tare_weight;

E.6 LD language

Ladder Diagram (LD) is a graphic representation of boolean equations, combining **contacts** (input arguments) with **coils** (output results). The LD language enables the description of tests and modifications of **boolean** data by placing **graphic symbols** into the program chart. LD graphic symbols are organized within the chart exactly as an electric contact diagram. LD diagrams are connected on the left side and on the right side to vertical **power rails**. These are basic graphic components of an LD diagram:



Left vertical power rail

Right vertical power rail

Horizontal connection line

Vertical connection line

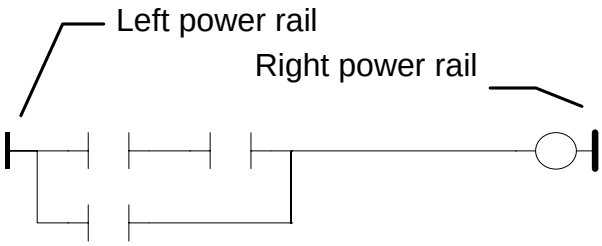
Multiple connection lines (all connected together)

Contact associated with a variable

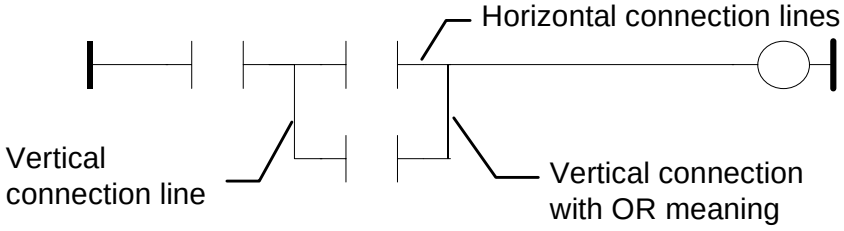
Coil associated to an output or to an internal variable

E.6.1 Power rails and connection lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.



LD diagram graphic symbols are connected to power rails or to other symbols by **connection lines**. Connection lines are horizontal or vertical.



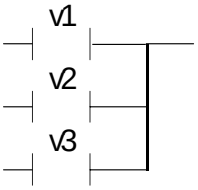
Each line segment has a boolean state **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

E.6.2 Multiple connection

The boolean state given to a single horizontal connection line is the same on the left and on the right extremities of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The boolean state of the extremities of a multiple connection follows logic rules.

A **multiple connection on the left** combines **more than one** horizontal lines connected on the **left** side of a vertical line, and **one** line connected on its **right** side. The boolean state of the right extremity is the **LOGICAL OR** between all the left extremities.

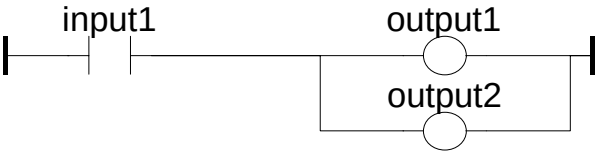
(* Example of multiple LEFT connection *)



(* right extremity state is (v1 OR v2 OR v3) *)

A **multiple connection on the right** combines **one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of the left extremity is propagated into each of the right extremities.

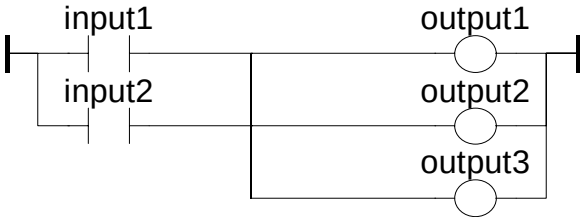
(* Example of multiple RIGHT connection *)



(* ST equivalence: *)
output1 := input1;
output2 := input1;

A **multiple connection on the left and on the right** combines **more than one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of each of the right extremities is the **LOGICAL OR** between all the left extremities

(* Example of multiple LEFT and RIGHT connection *)



(* ST Equivalence: *)
output1 := input1 OR input2;
output2 := input1 OR input2;
output3 := input1 OR input2;

E.6.3 Basic LD contacts and coils

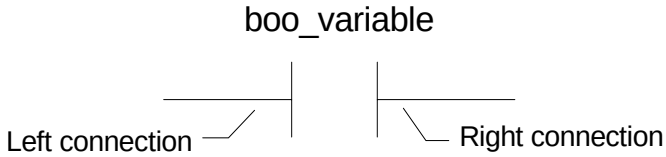
There are several symbols available for input contacts:
- Direct contact
- Inverted contact
- Contacts with edge detection

There are several symbols available for output coils:
- Direct coil
- Inverted coil
- SET coil
- RESET coil
- Coils with edge detection

The name of the variable is written above any of these graphic symbols:

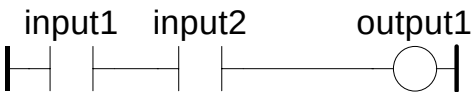
- ***Direct contact***

A direct contact enables a **boolean operation** between a **connection line** state and a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the value of the variable associated with the contact.

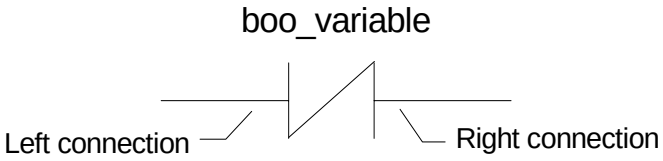
(* Example using DIRECT contacts *)



(* ST Equivalence: *)
output1 := input1 AND input2;

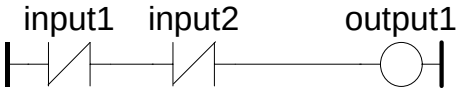
- ***Inverted contact***

An inverted contact enables a **boolean operation** between a **connection line** state and the boolean negation of a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the **boolean negation** of the value of the variable associated with the contact.

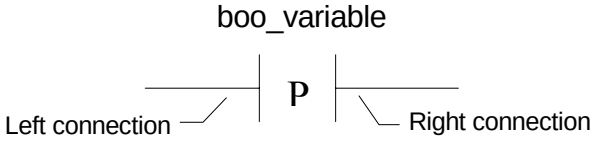
(* Example using INVERTED contacts *)



(* ST Equivalence: *)
output1 := NOT (input1) AND NOT (input2);

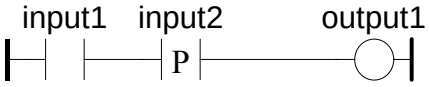
- ***Contact with rising edge detection***

This contact (positive) enables a **boolean operation** between a **connection line** state and the rising edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **rises** from FALSE to TRUE. It is reset to FALSE in all other cases.

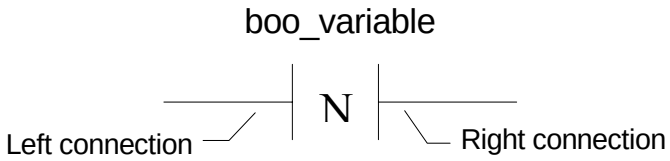
(* Example using RISING EDGE contacts *)



(* ST Equivalence: *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)

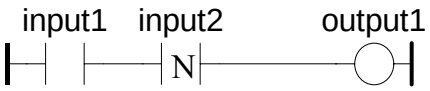
- ***Contact with falling edge detection***

This contact (negative) enables a **boolean operation** between a **connection line** state and the falling edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **falls** from TRUE to FALSE. It is reset to FALSE in all other cases.

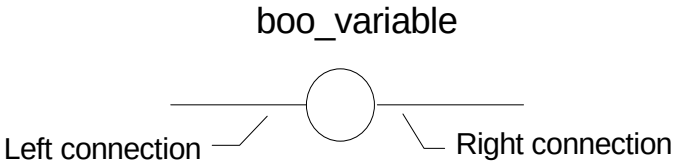
(* Example using FALLING EDGE contacts *)



(* ST Equivalence: *)
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)

- ***Direct coil***

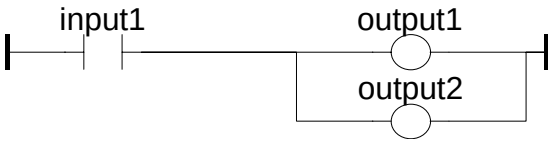
Direct coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is assigned with the boolean **state of the left connection**. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

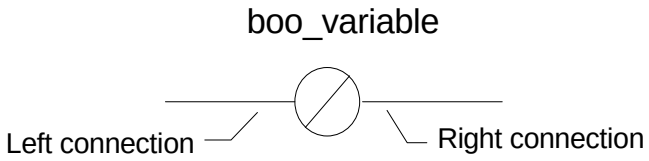
(* Example using DIRECT coils *)



(* ST Equivalence: *)
output1 := input1;
output2 := input1;

- ***Inverted coil***

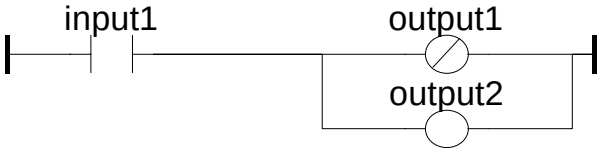
Inverted coils enable a **boolean output** according to the boolean **negation** of a **connection line** state.



The associated variable is assigned with the boolean **negation** of the **state of the left connection**. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

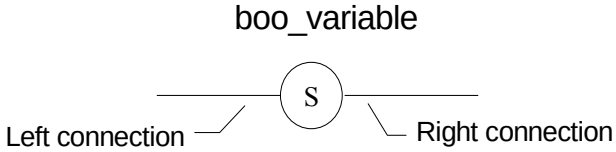
(* Example using INVERTED coils *)



(* ST Equivalence: *)
output1 := NOT (input1);
output2 := input1;

- ***SET coil***

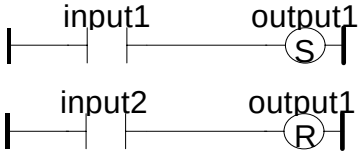
"Set" coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is **SET TO TRUE** when the boolean **state of the left connection** becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)

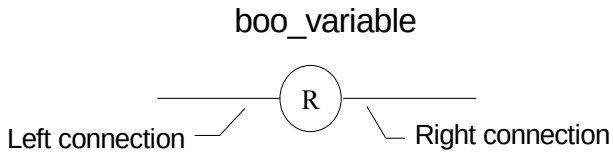


(* ST Equivalence: *)
IF input1 THEN

```
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

- ***RESET coil***

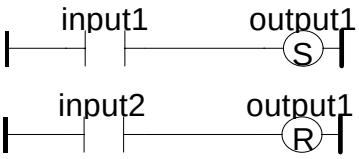
"Reset" coils enable **boolean output** of a **connection line** boolean state.



The associated variable is **RESET TO FALSE** when the boolean **state of the left connection** becomes **TRUE**. The output variable keeps this value until an inverse order is made by a "SET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

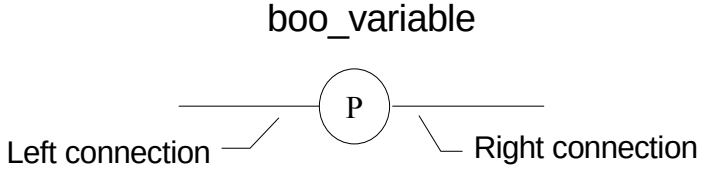
(* Example using "SET" and "RESET" coils *)




```
(* ST Equivalence: *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

- ***Coil with rising edge detection***

"Positive" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is

propagated into the right connection. Right connection may be connected to the right vertical power rail.

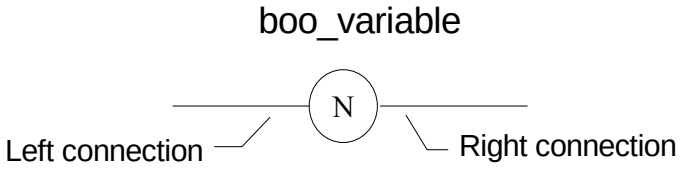
The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)


```
(* ST Equivalence: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)
```

- ***Coil with falling edge detection***

"Negative" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)


```
(* ST Equivalence: *)
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)
```

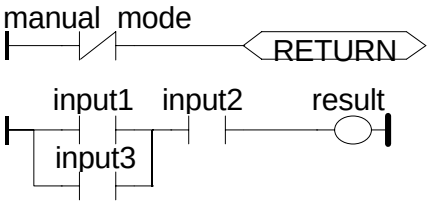
E.6.4 RETURN statement

The **RETURN** label can be used as an output to represent a conditional end of the program. No connection can be put on the right of a RETURN symbol.

If the **left connection** line has the **TRUE** boolean state, the program ends without executing the equations entered on the following lines of the diagram.

Note: When the LD program is a sub-program, its name has to be associated with an output coil to set the return value (returned to the calling program).

(* Example using RETURN symbol *)



(* ST Equivalence: *)
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;

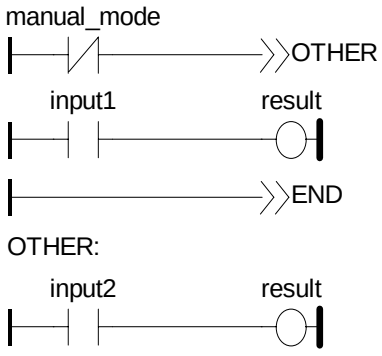
E.6.5 Jumps and labels

Labels, conditional and unconditional JUMPS symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB.....jump to label named "LAB"
LAB:.....definition of the label named "LAB"

If the **connection on the left** of the jump symbol has the **TRUE** boolean state, the program execution is driven after the label symbol.

(* Example using JUMP and LABEL symbols *)



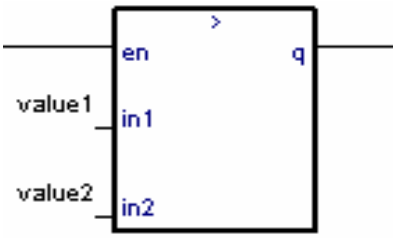
```
(* IL Equivalence: *)
      ldn     manual_mode
      jmpc    other
      ld      input1
      st      result
      jmp     END
OTHER:ld      input2
      st      result
END:  (* end of program *)
```

E.6.6 Blocks in LD

Using the Quick LD editor, you connect function boxes to boolean lines. A function can actually be an operator, a function block or a function. As all blocks do not have always a boolean input and/or a boolean output, inserting blocks in an LD diagram leads to the addition of new parameters EN, ENO to the block interface. The EN, ENO parameters are not added if you use the FBD/LD editor as you can connect the variable with the required type.

- ***The "EN" input***

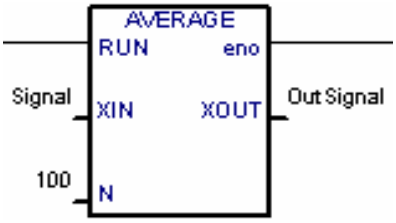
On some operators, functions or function blocks, the first input does not have boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE. Below is the example of a comparison operator, and the equivalent code expressed in ST:




```
IF rung_state THEN
  q := (value1 > value 2);
ELSE
  q := FALSE;
END_IF;
(* continue rung with q state *)
```

- ***The "ENO" output***

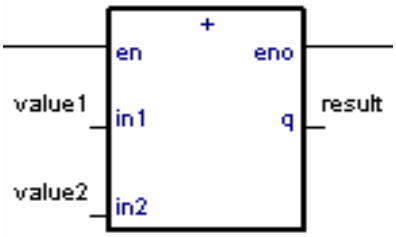
On some operators, functions or function blocks, the first output does not have boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "**ENO**". The **ENO** output always takes the same state as the first input of the block. Below is an example with AVERAGE function block, and the equivalent code expressed in ST:




```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```

- ***The "EN" and "ENO" parameters***

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in ST:




```
IF rung_state THEN
  result := (value1 + value2);
END_IF;
eno := rung_state;
(* continue rung with eno state *)
```

E.7 ST language

ST (**Structured Text**) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST is the default language for the description of the actions within the steps and conditions attached to the transitions of the **SFC** language.

E.7.1 ST main syntax

An ST program is a list of ST **statements**. Each statement ends with a semi-colon (**";"**) separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with **inactive separators** (space character, end of line or tab stops) or by **active separators**, which have a well defined significance (for example, the **">"** separator indicates a "greater than" comparison. Comments may be freely inserted into the text. A comment must begin with **"(*"** and ends with **"*)"**. Each statement terminates with a semi-colon (**";"**) separator. These are basic types of ST statements:

- **assignment** statement (variable := expression;)
- **sub-program** or **function** call
- **function block** call
- **selection** statements (IF, THEN, ELSE, CASE...)
- **iteration** statements (FOR, WHILE, REPEAT...)
- **control** statements (RETURN, EXIT...)
- special statements for links with other languages such as **SFC**

Inactive separators may be freely entered between active separators, constant expressions and identifiers. ST inactive separators are: **Space** (blank) character, **Tabs** and **End of line** character. Unlike line-formatted languages such as IL, end of lines may be entered anywhere in the program. The rules shown below should be followed when using inactive separators to increase ST program readability:

- Do not write more than one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase readability of lines or paragraphs

E.7.1 Expression and parentheses

ST expressions combine ST **operators** and variable or constant **operands**. For each single expression (combining operands with one ST operator), the **type** of the operands must be the same. This single expression has the same type as its operands, and can be used in a more complex expression. For example :

(boo_var1 AND boo_var2) has BOO type
not (boo_var1) has BOO type
(sin (3.14) + 0.72) has REAL ANALOG type
(t#1s23 + 1.78) is an invalid expression

Parentheses are used to isolate sub parts of the expression, and to explicitly order the priority of the operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default **priority** between ST operators. For example:

2 + 3 * 6	equals 2+18=20	because multiplication operator has a higher priority
(2+3) * 6	equals 5*6=30	priority is given by parenthesis

Warning: A maximum number of **8** levels of parentheses can be nested within an expression.

E.7.3 Function or function block calls

Standard ST function calls may be used for each of following objects:
- Sub-programs
- Library functions and function blocks written in IEC languages
- "C" functions and function blocks
- Type conversion functions

Calling sub-programs or functions

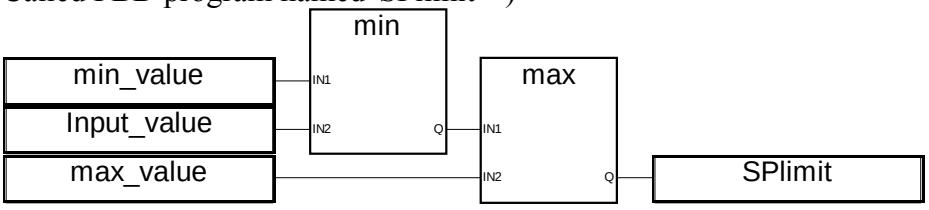
Name: name of the called sub-program
or library function written in IEC language or in "C"
Meaning: calls a ST, IL, LD or FBD sub-program or function or a "C" function
and gets its return value
Syntax: <**variable**> **:=** <**subprog**> **(**<**par1**>**,** ... <**parN**> **);**
Operands: The type of return value and calling parameters must follow
the interface defined for the sub-program.
Return value: value returned by the sub-program

Sub-program calls may be used in any expression. They also may be used in an SFC transition.

Example1: Sub-program call

```
(* Main ST program *)
(* gets an analog value and converts it into a limited time value *)
ana_timeprog := SPlimit ( tprog_cmd );
appl_timer := tmr (ana_timeprog * 100);
```

(* Called FBD program named 'SPlimit' *)



Example2: Function call

```
(* functions used in complex expressions: min, max, right, mlen and left are standard "C" functions *)
limited_value := min (16, max (0, input_value) );
```

rol_msg := right (message, mlen (message) - 1) + left (message, 1);

- ***Calling function blocks***

Name: name of the function block instance
Meaning: calls a function block from the ISaGRAF library or from the user's library and accesses its return parameters
Syntax: **(* call of the function block *)**
<blockname> (<p1>, <p2> ...);
(gets its return parameters *)
<result> := <blockname>. <ret_param1>;
...
<result> := <blockname>. <ret_paramN>;
Operands: parameters are expressions which match the type of the parameters specified for that function block
Return value: See Syntax to get the return parameters.

Consult the ISaGRAF library to find the meaning and type of each function block parameter. The function block instance (name of the copy) must be declared in the dictionary

Example :

(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

E.7.4 ST specific boolean operators

The following boolean operators are specific to the ST language:
- REDGE rising edge detection
- FEDGE falling edge detection

Other standard boolean operators such as:
- NOT boolean negation
- AND (&) logical AND
- OR logical OR
- XOR logical exclusive OR

can be used. Their description is to be found in the section 'Standard operators, function blocks and functions'.

- ***"REDGE" operator***

Name: **REDGE**
Meaning: evaluates the rising edge of a complete boolean expression
Syntax: **<edge> := REDGE (<boo_expression>,<memo_variable>);**

Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store the last state of the expression
Return value: TRUE when the expression changes from FALSE to TRUE
FALSE for all other cases

The rising edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. This operator can be used to describe the condition attached to an SFC transition.

Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the REDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other REDGE evaluations.

Example:

```
(* ST program using REDGE operator *)

(* this program counts the rising edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If REDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of R_TRIG standard block. It has been kept for compatibility reasons.

- ***"FEDGE" operator***

Name: **FEDGE**
Meaning: evaluates the falling edge of a boolean expression
Syntax: **<edge> := FEDGE (<boo_expression>, <memo_variable>);**
Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store
the last state of the expression
Return value: TRUE when the expression changes from TRUE to FALSE
FALSE for all other cases

The falling edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. The operator can be used to describe the condition attached to an SFC transition.
Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the FEDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other FEDGE evaluations.

Example:

```
(* ST program using FEDGE operator *)

(* this program counts the falling edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If FEDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of F_TRIG standard block. It has been kept for compatibility reasons.

E.7.5 ST basic statements

The basic statements of the ST language are:
- Assignment
- RETURN statement
- IF-THEN-ELSIF-ELSE structure
- CASE statement
- WHILE iteration statement
- REPEAT iteration statement
- FOR iteration statement
- EXIT statement

Assignment

Name: **:=**
Meaning: assigns a variable to an expression
Syntax: **<variable> := <any_expression> ;**
Operands: variable must be internal or output
variable and expression must have the same type

The expression can be a call to a sub-program or a function from the ISaGRAF library

Example:

```
(* ST program with assignments *)

(* variable <<= variable *)
bo23 := bo10;

(* variable <<= expression *)
bo56 := bx34 OR alrm100 & (level >= over_value);
```

```
result := (100 * input_value) / scale;

(* assignment with sub-program return value *)
rc := PSelect ( );

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

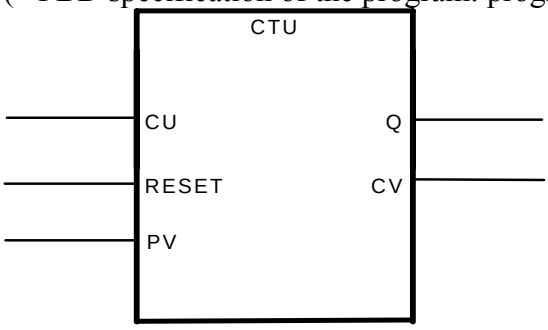
- ***RETURN statement***

Name: **RETURN**
Meaning: terminates the execution of the current program
Syntax: **RETURN ;**
Operands: (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

Example:

(* FBD specification of the program: programmable counter *)




```
(* ST implementation of the program, using RETURN statement *)

If not (CU) then
      Q := false;
      CV := 0;
      RETURN;   (* terminates the program *)
end_if;

if R then
      CV := 0;
else
      if (CV < PV) then
        CV := CV + 1;
      end_if;
end_if;
Q := (CV >= PV);
```

- ***IF-THEN-ELSIF-ELSE statement***

Name: **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**
Meaning: executes one of two lists of ST statements
selection is made according to the value

of a boolean expression

Syntax: **IF <boolean_expression> THEN**
<statement> ;
<statement> ;
...
ELSIF <boolean_expression> THEN
<statement> ;
<statement> ;
...
ELSE
<statement> ;
<statement> ;
...
END_IF;

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE.

Example:

```
(* ST program using IF statement *)

IF manual AND not (alarm) THEN
    level := manual_level;
    bx126 := bi12 OR bi45;
ELSIF over_mode THEN
    level := max_level;
ELSE
    level := (lv16 * 100) / scale;
END_IF;

(* IF structure without ELSE *)
If overflow THEN
        alarm_level := true;
END_IF;
```

- ***CASE statement***

Name: **CASE ... OF ... ELSE ... END_CASE**
Meaning: executes one of several lists of ST statements
selection is made according to an integer expression
Syntax: **CASE <integer_expression> OF**
<value> : <statements> ;
<value> , <value> : <statements> ;
...
ELSE
<statements> ;
END_CASE;

Case values must be integer constant expressions. Several values, separated by comas, can lead to the same list of statements. The ELSE statement is optional.

Example:

```
(* ST program using CASE statement *)

CASE error_code OF
    255:    err_msg := 'Division by zero';
      fatal_error := TRUE;
    1:      err_msg := 'Overflow';
    2, 3:   err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
END_CASE;
```

- ***WHILE statement***

Name: **WHILE ... DO ... END_WHILE**
Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated BEFORE any iteration
Syntax: **WHILE <boolean_expression> DO**
<statement> ;
<statement> ;
...
END_WHILE ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

Example:

```
(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END_WHILE;
```

- ***REPEAT statement***

Name: **REPEAT ... UNTIL ... END_REPEAT**
Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration

Syntax: **REPEAT**
<statement> ;
<statement> ;
...
UNTIL <boolean_condition>
END_REPEAT ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

Example:

```
(* ST program using REPEAT statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
    REPEAT
      string := string + ComGetChar ( );
      nbchar := nbchar + 1;
    UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
    END_REPEAT;
END_IF;
```

- ***FOR statement***

Name: **FOR ... TO ... BY ... DO ... END_FOR**
Meaning: executes a limited number of iterations, using an integer analog index variable
Syntax: **FOR <index> := <mini> TO <maxi> BY <step> DO**
<statement> ;
<statement> ;
END_FOR;
Operands: **index**: internal analog variable increased at any loop
mini: initial value for index (before first loop)
maxi: maximum allowed value for index
step: index increment at each loop

The [BY step] statement is optional. If not specified, the increment step is 1

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during FOR iterations.

This is the "while" equivalent of a FOR statement:

index := mini;

```
    while (index <= maxi) do
      <statement> ;
      <statement> ;
      index := index + step;
    end_while;
```

Example:

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
      target := target + char (code);
    END_IF;
END_FOR;
```

- ***EXIT statement***

Name: **EXIT**
Meaning: exit from a FOR, WHILE or REPEAT iteration statement
Syntax: **EXIT;**
Operands: (none)

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

Example:

```
(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code  = searched_char) THEN
      found := YES;
      EXIT;
    END_IF;
END_FOR;
```

E.7.6 ST extensions

The following functions are extensions of the ST language:
- TSTART - TSTOP: timer control

The following statements and functions are available to control the execution of the SFC child programs. They may be used inside ACTION(): ... END_ACTION; blocks in SFC steps.

- GSTART starts an SFC program
- GKILL kills an SFC program
- GFREEZE freezes an SFC program
- GRST restarts a frozen SFC program
- GSTATUS gets current status of an SFC program

Warning: These functions are not in the IEC 1131-3 norm.
Easy equivalent can be found for GSTART and GKILL using the following syntax in the SFC step:
child_name(S); (* equivalent to GSTART(child_name); *)
child_name(R); (* equivalent to GKILL(child_name); *)

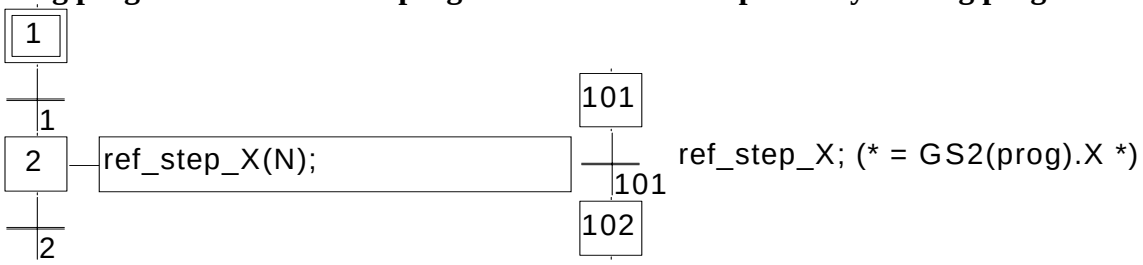
The following fields can be used to access the status of an SFC step:
GSnnn.x boolean value that represents the activity of the step
GSnnn.t time elapsed since the last activation of the step
("**nnn**" is the reference number of the SFC step)

It is also possible to test the activity of a step declared in another SFC program, by using the following syntax:

GSnnn(progname).x

Warning: referencing a step of an other program, using this syntax is not in the IEC 1131-3 norm. An easy way to do the same respecting IEC rules, is to declare a global boolean variable in the dictionary which will represent the step activity to be tested (for example ref_step_X). Then you insert in the step, the variable with the N qualifier (ref_step_X(N);). Then in the program which wants to test the activity of the step, you use the variable.

Prog program **the other program which needs step activity of Prog program**

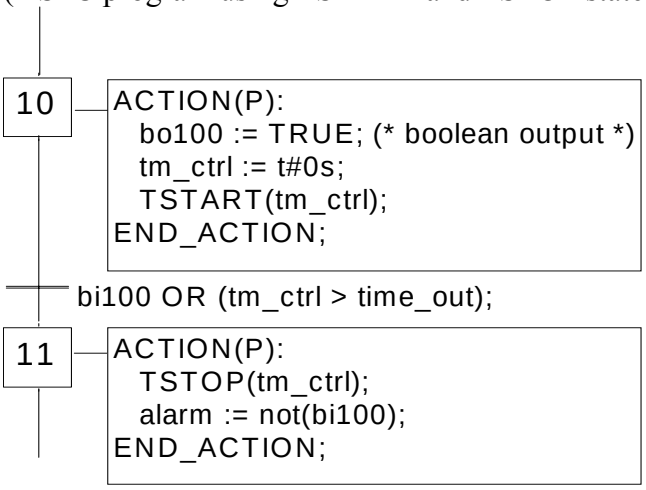


- ***TSTART statement***

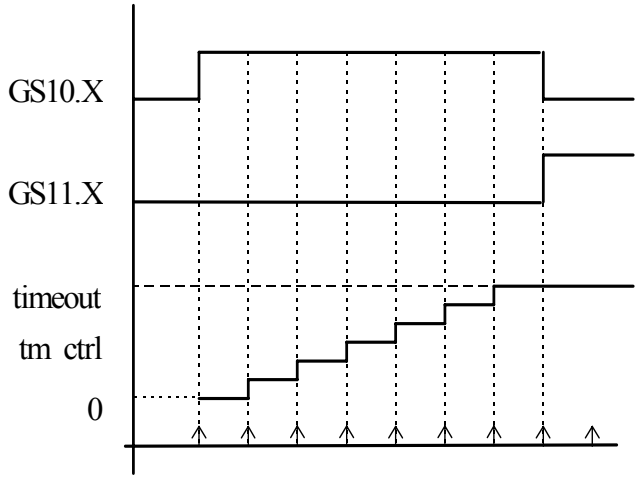
Name: **TSTART**
Meaning: starts the counting of a timer variable
timer value is not modified by the TSTART command, i.e. the counting starts from the current value of the timer.
Syntax: **TSTART (<timer_variable>);**
Operands: any inactive timer variable
Return value: (none)

Example:

(* SFC program using TSTART and TSTOP statements *)



Time diagram if bi100 is always FALSE:



The timer keeps the same value during one cycle.

- ***TSTOP statement***

Name: **TSTOP**
Meaning: stops updating a timer variable
timer value is not modified by the TSTOP command
Syntax: **TSTOP (<timer_variable>);**
Operands: any active timer variable
Return value: (none)

Example: See TSTART (the function is described above)

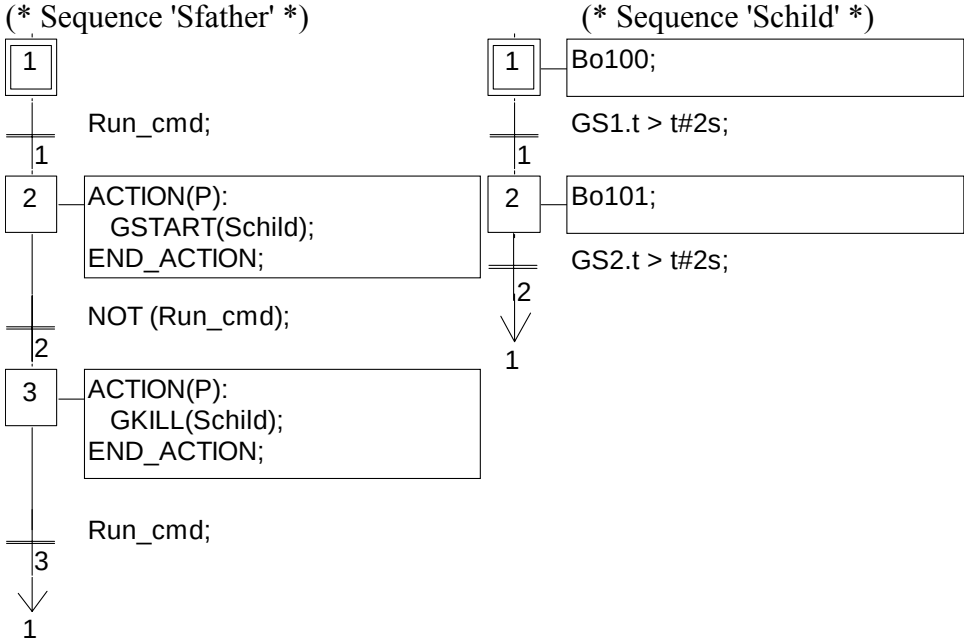
- ***GSTART statement***

Name: **GSTART**
Meaning: starts a child SFC program by putting a token
into each of its initial steps
Syntax: **GSTART (<child_program>);**

Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are not automatically started by the GSTART statement.
Note: As GSTART is not in the IEC 1131-3 norm, prefer the use of the S qualifier, with the following syntax to start a child SFC:
Child_name(S);

Example: Use of GSTART and GKILL



- ***GKILL statement***

Name: **GKILL**
Meaning: kills a child SFC program by removing the tokens currently existing in its steps
Syntax: **GKILL (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are automatically killed with the specified program.
Note: As GKILL is not in the IEC 1131-3 norm, prefer the use of the R qualifier, with the following syntax to kill a child SFC:
Child_name(R);

Example: See GSTART (function described above)

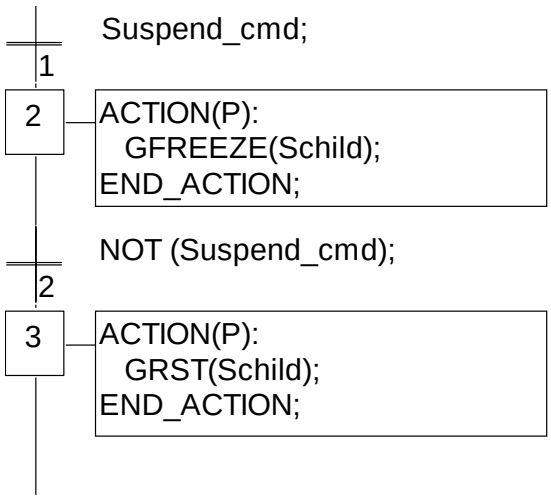
- ***GFREEZE statement***

Name: **GFREEZE**
Meaning: Suspends the execution of a child SFC program.
Frozen program can be restarted by the GRST statement.

Syntax: **GFREEZE (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are automatically frozen along with the specified program.
Note: GFREEZE is not in the IEC 1131-3 norm.

Example:



- ***GRST statement***

Name: **GRST**
Meaning: Restarts a child SFC program frozen by the GFREEZE statement.
Syntax: **GRST (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are automatically restarted by the GRST statement
Note: GRST is not in the IEC 1131-3 norm.

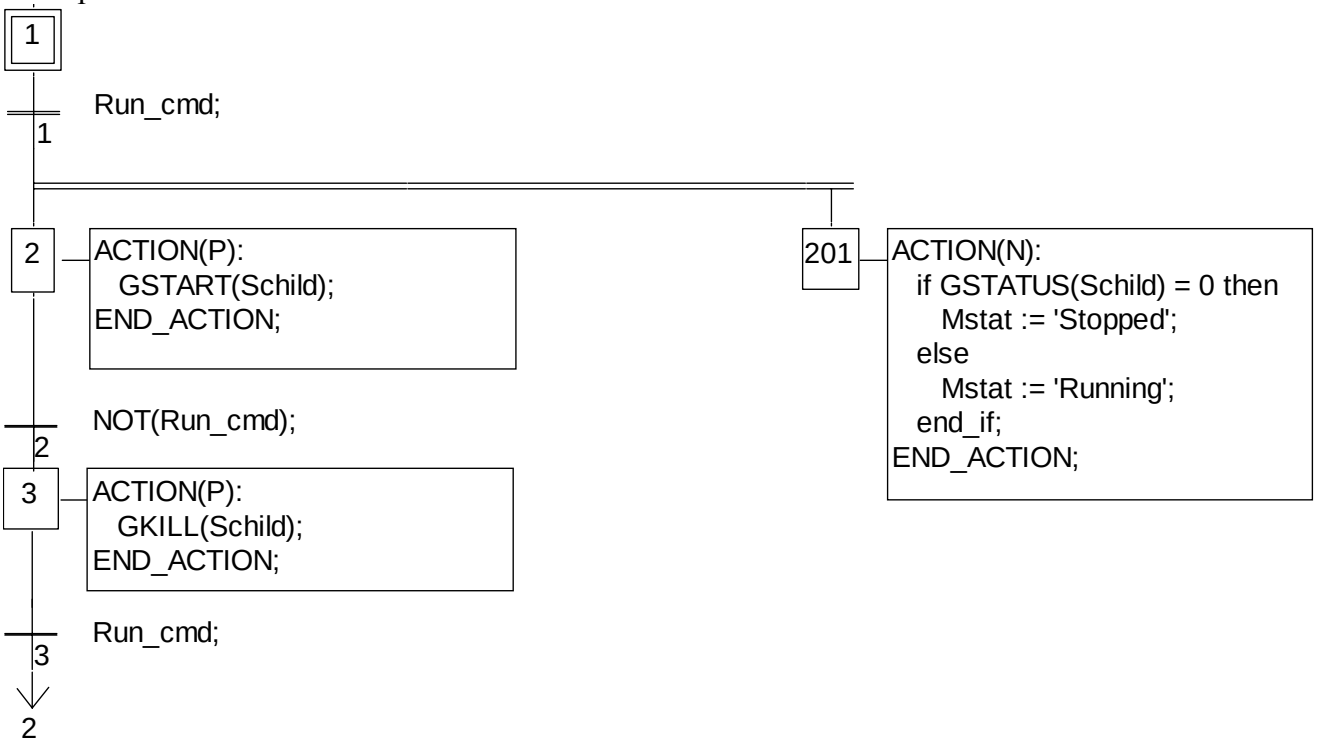
Example: See GFREEZE (function described above)

- ***GSTATUS statement***

Name: **GSTATUS**
Meaning: returns the current status of an SFC program
Syntax: **<ana_var> := GSTATUS (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: 0 = program is inactive (killed)
1 = program is active (started)
2 = program is frozen

Note: GSTATUS is not in the IEC 1131-3 norm.

Example:


1
Run_cmd;
1
2
ACTION(P):
GSTART(Schild);
END_ACTION;
201
ACTION(N):
if GSTATUS(Schild) = 0 then
Mstat := 'Stopped';
else
Mstat := 'Running';
end_if;
END_ACTION;
NOT(Run_cmd);
2
3
ACTION(P):
GKILL(Schild);
END_ACTION;
Run_cmd;
3
2

E.8 IL language

Instruction List, or **IL** is a low level language. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

E.8.1 IL main syntax

An IL program is a list of **instructions**. Each instruction must begin on a new line, and must contain an **operator**, completed with optional **modifiers** and, if necessary, for the specific operation, one or more **operands**, separated with commas (**','**). A **label** followed by a colon (**':'**) may precede the instruction. If a **comment** is attached to the instruction, it must be the last component of the line. Comments always begin with **'(*'** and ends with **'*)'**. Empty lines may be entered between instructions. Comments may be put on empty lines. Below are examples of instruction lines:

Label	*Operator*	*Operand*	*Comments*
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

- ***Labels***

A **label** followed by a colon (**':'**) may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Naming labels must conform to the following rules:

- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters**, **digits** or **'_'** character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a variable.

- ***Operator modifiers***

The available operator modifiers are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

N boolean negation of the operand
(delayed operation
C conditional operation

The **'N'** modifier indicates a boolean negation of the operand. For example, the instruction **ORN IX12** is interpreted as: **result := result OR NOT (IX12)**.

The parenthesis **'('** modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis **')'** operator is encountered.

The **'C'** modifier indicates that the attached instruction must be executed only if the current result has the boolean value TRUE (different than 0 for non-boolean values). The **'C'** modifier can be combined

with the '**N**' modifier to indicate that the instruction must be executed only if the current result has the boolean value FALSE (or 0 for non-boolean values).

- ***Delayed operations***

Because there is only one IL register (current result), some operations may have to be delayed, so that the execution order or the instructions can be changed. Parentheses are used to indicate delayed operations:

'('	is a modifier	indicates the operation to be delayed
')'	is an operator	executes the delayed operation

The opening parenthesis '**(**' modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis '**)**' operator is encountered. For example, following sequence:

```
AND(  IX12
OR    IX35
)
```

is interpreted as:

result := result AND (IX12 OR IX35)

E.8.2 IL operators

The following table summarizes the standard operators of the IL language:

Operator	*Modifiers*	*Operand*	*Description*
LD	N	Variable, constant	Loads operand
ST	N	Variable	Stores current result
S R		BOO variable BOO variable	Sets to TRUE Resets to FALSE
AND & OR XOR	N (N (N (N (	BOO BOO BOO BOO	boolean AND boolean AND boolean OR exclusive OR
ADD SUB MUL DIV	((((	variable, constant variable, constant variable, constant variable, constant	Addition Subtraction Multiplication Division
GT GE EQ LE LT NE	((((((	variable, constant variable, constant variable, constant variable, constant variable, constant variable, constant	Test: > Test: >= Test: = Test <= Test < Test <>

CAL JMP RET	C N C N C N	Function block instance name Label	Calls a function block Jumps to label Returns from sub-program
)			Executes delayed operation

In the next section, only operators which are specific to the IL language are described, other standard operators can be found in the section "standard operators, function blocks and functions".

- ***LD operator***

Operation loads a value in the current result
Allowed modifiers N
Operand constant expression
internal, input or output variable

Example:

```
            (* EXAMPLES OF LD OPERATIONS *)
LDex:   LD      false                   (* result := FALSE boolean constant *)
    LD      true                    (* result := TRUE boolean constant *)
    LD      123                     (* result := integer constant *)
    LD      123.1                   (* result := real constant *)
    LD      t#3ms                   (* result := time constant *)
    LD      boo_var1        (* result := boolean variable *)
    LD      ana_var1        (* result := analog variable *)
    LD      tmr_var1        (* result := timer variable *)
    LDN     boo_var2        (* result := NOT ( boolean variable ) *)
```

- ***ST operator***

Operation stores the current result in a variable
the current result is not modified by this operation
Allowed modifiers N
Operand internal or output variable

Example:

```
            (* EXAMPLES OF ST OPERATIONS *)
STboo:  LD      false
    ST      boo_var1        (* boo_var1 := FALSE *)
    STN     boo_var2        (* boo_var2 := TRUE *)
STana:  LD      123
    ST      ana_var1        (* ana_var1 := 123 *)
STtmr:  LD      t#12s
    ST      tmr_var1        (* tmr_var1 := t#12s *)
```

- ***S operator***

Operation: stores the boolean value TRUE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation

Allowed modifiers: (none)

Operand: output or internal boolean variable

Example:

```
            (* EXAMPLES OF S OPERATIONS *)
SETex:      LD      true                        (* current result := TRUE *)
    S       boo_var1        (* boo_var1 := TRUE *)
                                                (* current result is not modified *)
    LD      false                               (* current result := FALSE *)
    S       boo_var1        (* nothing done - boo_var1 unchanged *)
```

- ***R operator***

Operation stores the boolean value FALSE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation

Allowed modifiers (none)

Operand output or internal boolean variable

Example:

```
            (* EXAMPLES OF R OPERATIONS *)
RESETex:    LD      true                        (* current result := TRUE *)
    R       boo_var1        (* boo_var1 := FALSE *)
                                                (* current result is not modified *)
    ST      boo_var2        (* boo_var2 := TRUE *)
    LD      false                               (* current result := FALSE *)
    R       boo_var1        (* nothing done - boo_var1 unchanged *)
```

- ***JMP operator***

Operation jumps to the specified label

Allowed modifiers C N

Operand label defined in the same IL program

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)          *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with               *)
(* the JMPC operator                 *)

JMPex:      LD      selector                    (* selector is 0 or 1 or 2 *)
    BOO                                 (* conversion to boolean *)
    JMPC    test1                       (* if selector = 0 then *)
    LD      true
    ST      bo0                         (* bo0 := true *)
    JMP     JMPend          (* end of the program *)
```

```
test1:    LD    selector
     SUB  1                          (* decrease selector:  is now 0 or 1 *)
     BOO                             (* conversion to boolean *)
     JMPC test2                      (* if selector = 0 then *)
     LD   true
     ST   bo1                        (* bo1 := true *)
     JMP  JMPend          (* end of the program *)
test2:    LD    true                        (* last possibility *)
     ST   bo2                        (* bo2 := true *)
JMPend:                                      (* end of the IL program *)
```

- ***RET operator***

Operation ends the current instruction list. If the IL sequence is a sub-program, the current result is returned to the calling program

Allowed modifiers C N

Operand (none)

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)          *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with               *)
(* the JMPC operator                          *)

JMPex:    LD    selector                    (* selector is 0 or 1 or 2 *)
     BOO                             (* conversion to boolean *)
     JMPC test1                      (* if selector = 0 then *)
     LD   true
     ST   bo0                        (* bo0 := true *)
     RET                             (* end - return 0 *)
                                     (* decrease selector *)
test1:    LD    selector
     SUB  1                          (* selector is now 0 or 1 *)
     BOO                             (* conversion to boolean *)
     JMPC test2                      (* if selector = 0 then *)
     LD   true
     ST   bo1                        (* bo1 := true *)
     LD   1                          (* load real selector value *)
     RET                             (* end - return 1 *)
                                     (* last possibility *)
test2:    RETNC                              (* returns if the selector has *)
                                     (* an invalid value *)
     LD   true
     ST   bo2                        (* bo2 := true *)
     LD   2                          (* load real selector value *)
                                     (* end - return 2 *)
```

- ***")" operator***

Operation executes a delayed operation. The delayed operation was notified by '('

Allowed modifiers (none)

Operand (none)

Example:

```
(* The following program interleaves delayed operations: *)
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)

Delayed:  LD    a1      (* result := a1; *)
    ADD(  a2    (* delayed ADD - result := a2; *)
    MUL(  a3    (* delayed MUL - result := a3; *)
    SUB   a4    (* result := a3 - a4; *)
    )           (* execute delayed MUL - result := a2 * (a3-a4); *)
    MUL   a5    (* result := a2 * (a3 - a4) * a5; *)
    )           (* execute delayed ADD *)
                (* result := a1 + (a2 * (a3 - a4) * a5); *)
    ADD   a6    (* result := a1 + (a2 * (a3 - a4) * a5) + a6; *)
    ST    res   (* store current result in variable res *)
```

- ***Calling sub-programs or functions***

A sub-program or a function (written in any of the IL, ST, LD, FBD or "C" language) is called from the IL language, using its name as an operator.

Operation executes a sub-program or a function - the value returned by the sub-program or function is stored into the IL current result

Allowed modifiers (none)

Operand The first calling parameter must be stored in the current result before the call. The following ones are expressed in the operand field, separated by comas.

Example:

```
(* Calling program : converts an analog value into a time value *)

Main:     LD      bi0
    SUBPRO        bi1,bi2     (* call sub-program to get analog value *)
    ST    result  (* result := value returned by sub-program *)
    GT    vmax    (* test value overflow *)
    RETC          (* return if overflow *)
    LD    result
    MUL   1000    (* converts seconds in milliseconds *)
    TMR           (* converts to a timer *)
    ST    tmval   (* stores converted value in a timer *)

(* Called sub-program named 'SUBPRO' : evaluates the analog value *)
(* given as a binary value on three boolean inputs: in0, in1, in2 are the three boolean input parameters of
    the sub-program *)

    LD    in2
    ANA                         (* result = ana (in2); *)
    MUL   2                     (* result := 2*ana (in2); *)
```

```
ST      temporary       (* temporary := result *)
LD      in1
ANA
ADD     temporary       (* result := 2*ana (in2) + ana (in1); *)
MUL     2                       (* result := 4*ana (in2) + 2*ana (in1); *)
ST      temporary       (* temporary := result *)
LD      in0
ANA
ADD     temporary       (* result := 4*ana (in2) + 2*ana (in1)+ana (in0); *)
ST      SUBPRO          (* return current result to calling program *)
```

- ***Calling function blocks: CAL operator***

Operation calls a function block
Allowed modifiers C N
Operand Name of the function block instance.

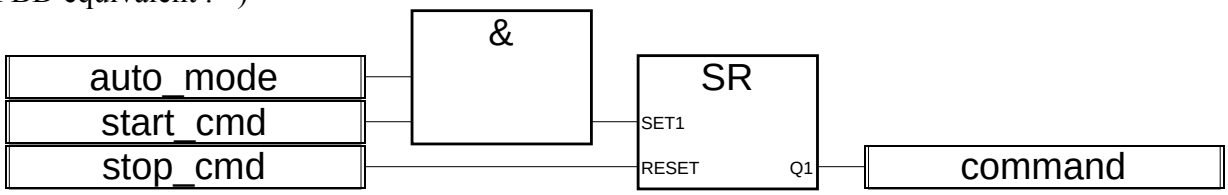
The input parameters of the blocks must be assigned before the call using LD/ST operations sequence.

Output parameters are known if used.

Example1:

```
(* Calling function block SR : SR1 is an instance of SR *)
LD  auto_mode
AND     start_cmd
ST  SR1.set1
LD  stop_cmd
ST  SR1.reset
CAL     SR1
LD  SR1.Q1
ST  command
```

(* FBD equivalent : *)

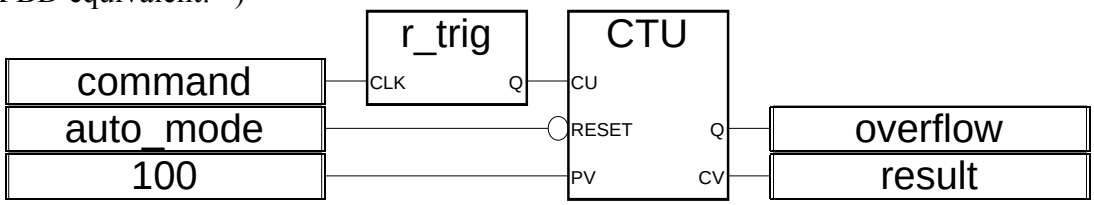


Example 2

```
(*We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)
LD  command
ST  R_TRIG1.clk
CAL     R_TRIG1
LD  R_TRIG1.Q
ST  CTU1.cu
LDN     auto_mode
ST  CTU1.reset
LD  100
ST  CTU1.pv
```

```
CAL         CTU1
LD  CTU1.Q
ST  overflow
LD  CTU1.cv
ST  result

(* FBD equivalent: *)
```



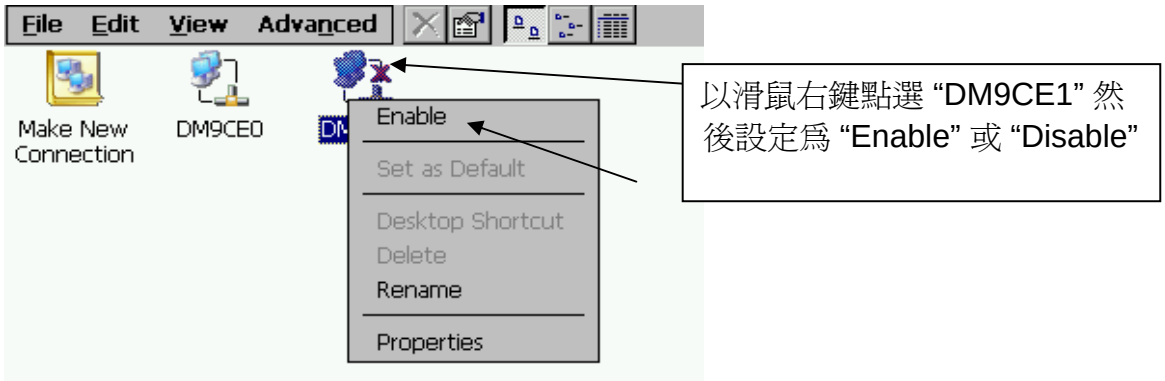
附錄 F ： 如何 Enable/Disable W-8x47 的 LAN2

重要 :

1. 當 LAN2 沒有使用時, 請設在停用狀態 –“Disable”.
2. LAN1 需設定爲固定的 IP (若 LAN2 設爲啓用 –”Enable”, 也要設定固定 IP).

W-8047/8347/8747 及 W-8046/8346/8746 的 LAN2 預設狀態爲 “Disable”, 使用之前須先 “Enable”.

點選 “Start” – “Setting” - “Control Panel”, 開啓 “Network and Dual-up Connections” 設定 LAN2: DM9CE1 的啓用或停用.



然後執行 “Start” – “Programs” – “Wincon Utility”, 點選 “Save and Reboot” 來儲存設定.