Panasonic

FPWIN Pro 導入

研修講義【下】

第 9 章～第 15 章

台灣松下環境方案股份有限公司

地址：台北市中山北路二段 44 號 15 樓　電話：+886 2 2581 6020

目次

第 7 章　使用變數撰寫程式

第 8 章　製作 Function(FUN)/Function Block(FB)

第 9 章　用 Fuction diagram 製作程式

第 10 章　用 Instruction list 製作程式

第 11 章　用 Structured Text 製作程式

第 12 章　用 Sequential function chart 製作程式

第 13 章　監控功能

第 14 章　User library

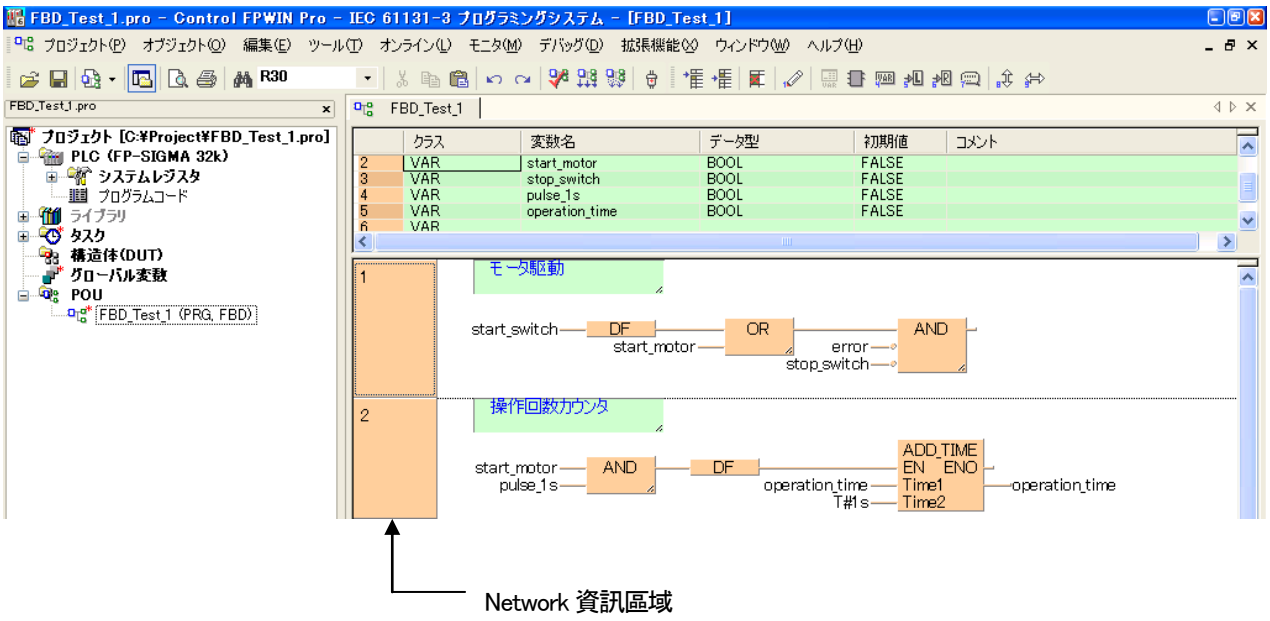
第 15 章　安全性

第9章

用 Function Block 圖製作程式

9-1 概要

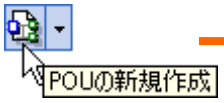
所謂功能區塊圖(FBD)就是將程式定義在圖示裡。
FBD 為「Function」「Jump」「輸出入變數」所構成。
下圖的 2 個 Network 是用 FBD 語言所作成的程式。



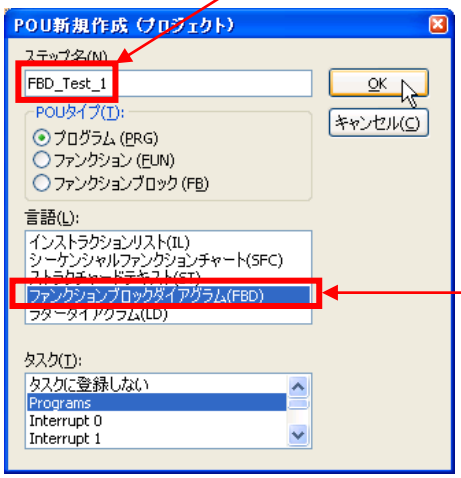
Network 的標籤顯示於 Block 資訊區域。在此顯示中斷點和錯誤訊息等的程式狀態資訊。

9-2 作成新 Project

製作新 POU。



取名成叫做“FBD_Test_1”。



選擇 Function Block 圖(FBD)。

如上圖、

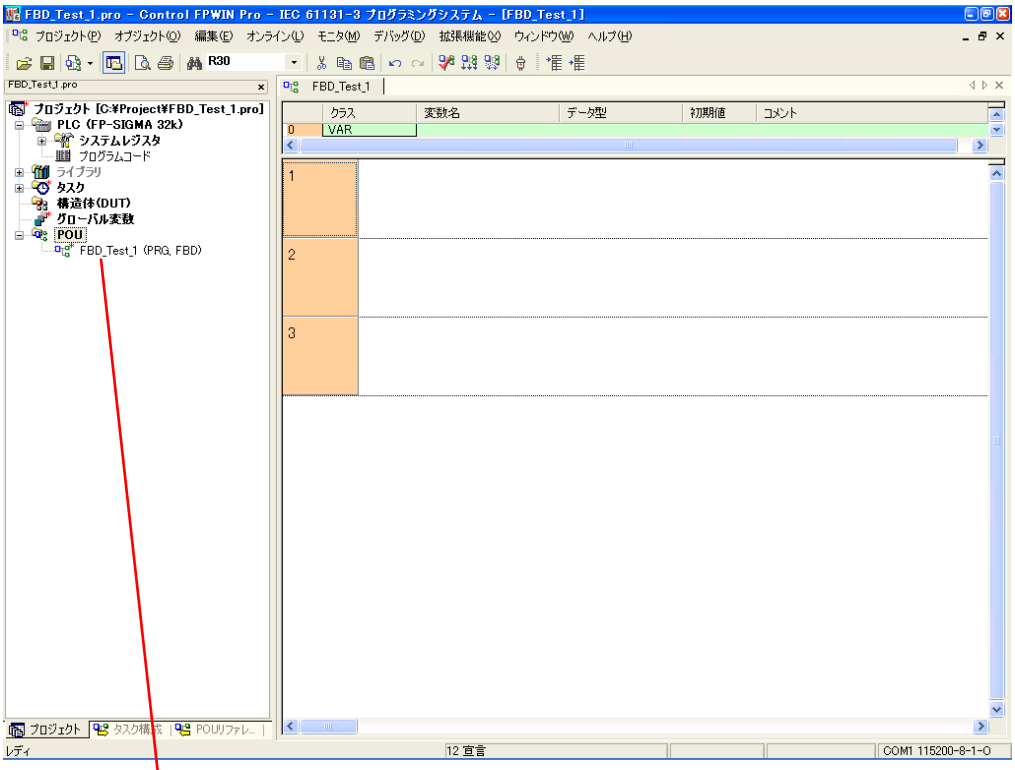
POU 類型：程式(PRG)

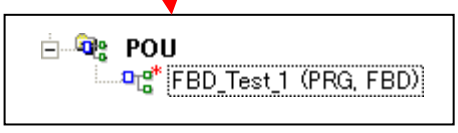
程式語言：Function Block 圖(FBD)

Task：Programs

選擇之後按下「OK」按鍵。

開啟如同下圖畫面。





POU 名稱顯示為「FBD_Test_1」。

雖然跟階梯圖的畫面很雷同、但是有許多不同的差異。

階梯圖(LD)時的描畫小圖示:

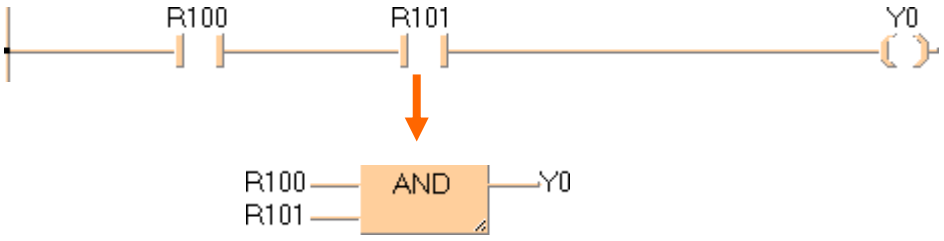
Function Block 圖(FBD)時的描畫小圖示:

FBD 沒有像有 LD 的  (接點)、  (OUT)的小圖示。
這裡為 LD 和 FBD 的最大差異。

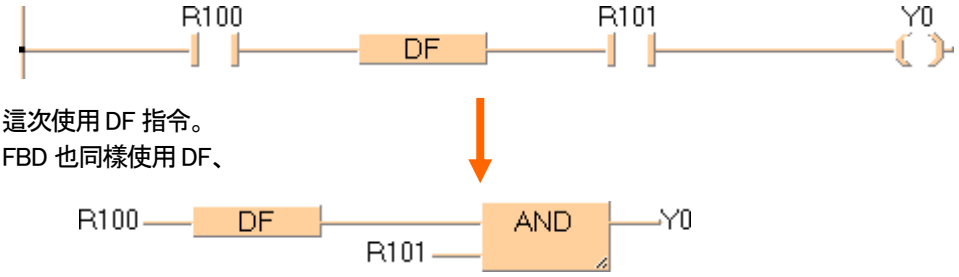
那麼到底要如何製作階梯圖程式・・・

9-3 製作 FBD 程式

“Function Block” 如其名、用來連接 Function 並作成回路圖。
例如階梯圖程式在下圖的情況、

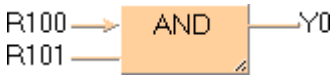


會變成這樣。
那麼下個範例會如何？



這次使用 DF 指令。
FBD 也同樣使用 DF、

雖然可以記述成

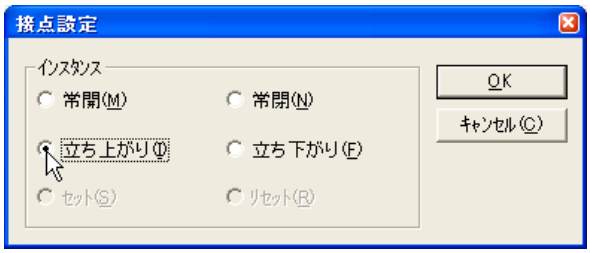


如上圖般不使用(DF)也可以畫出相同回路。

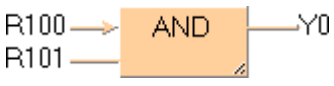
<u>畫圖方法</u>

①將游標放置到想變更輸入的位置。

②點選兩次。
打開接點設定對話框、選擇「啓動」。



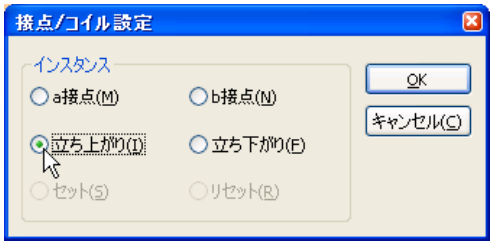
③按下「OK」按鍵即完成。

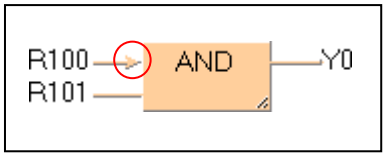


●備註

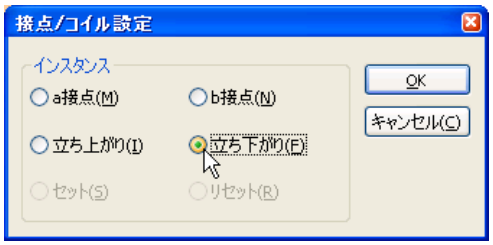
實際的畫圖範例

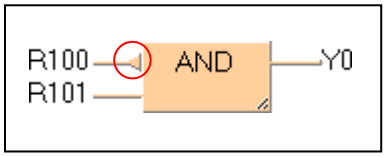
啓動



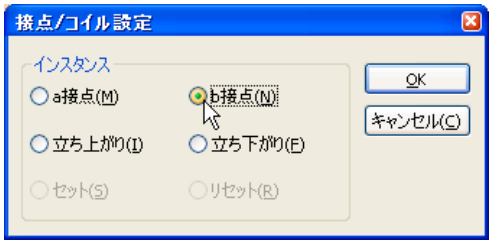


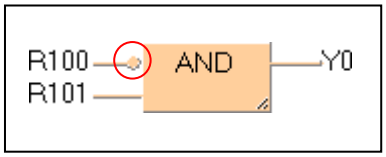
關閉





常閉





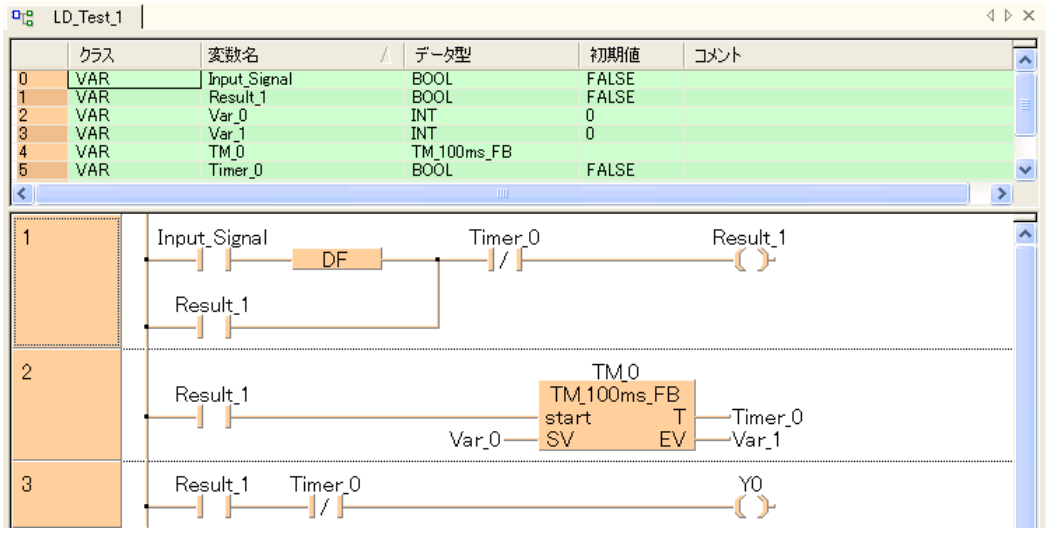
LD(階梯圖)也可以作相同的輸入。

9-4 將 LD 程式置換為 FBD 程式

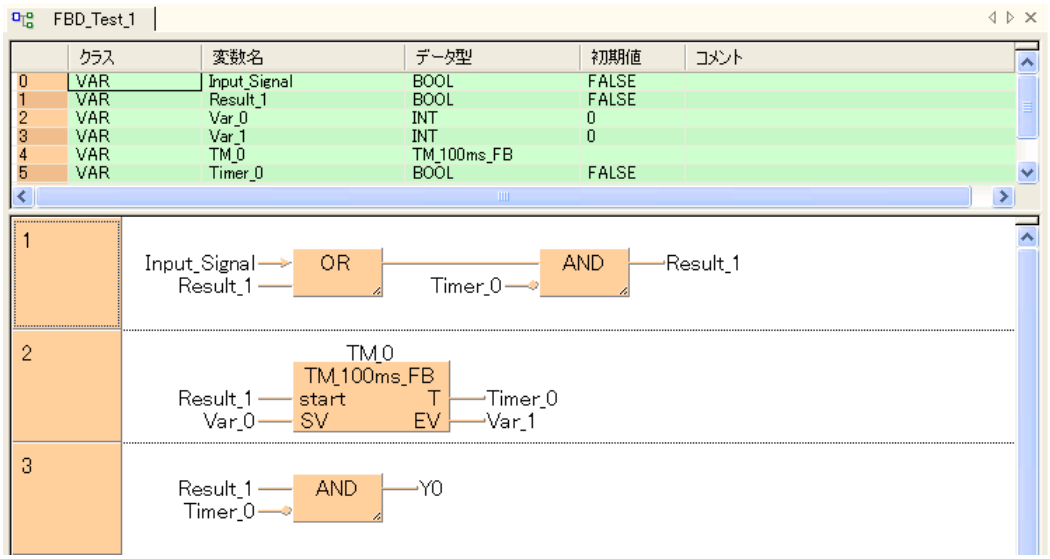
將上一章作好的 LD 程式轉換成 FBD 程式看看。

■LD程式

上一章作好的程式。



■FBD 程式



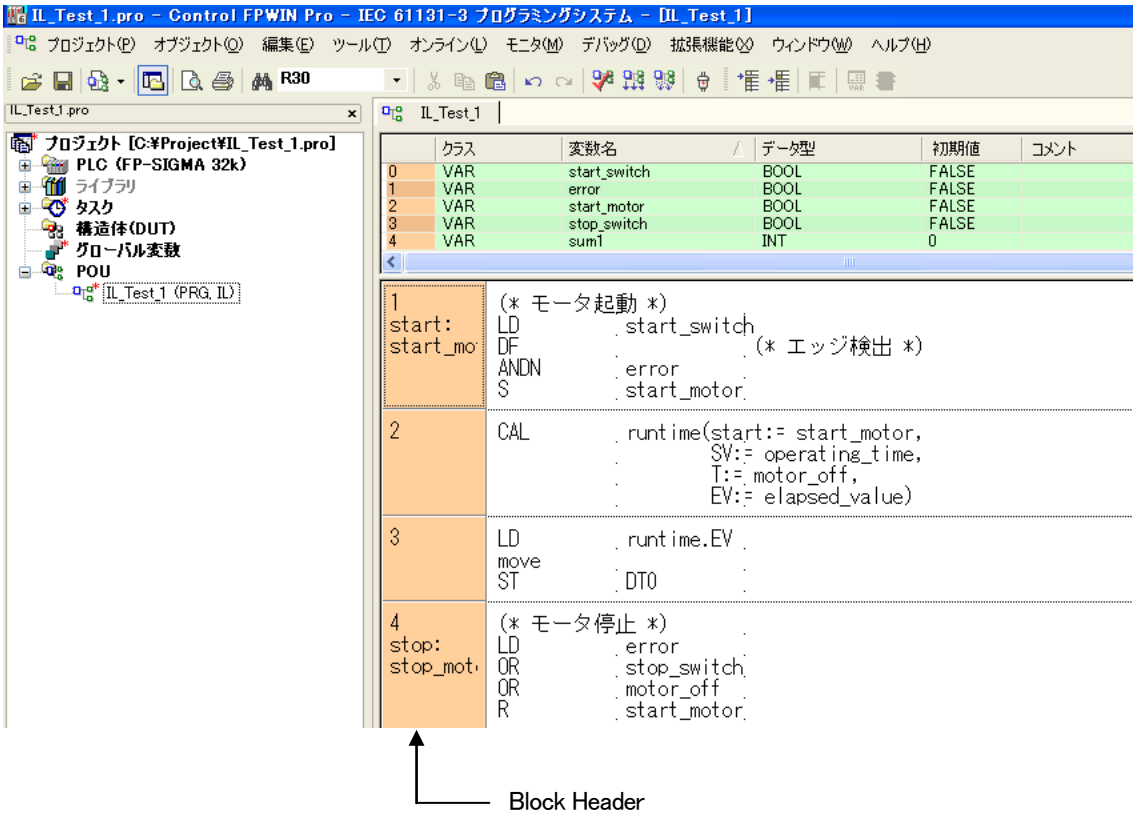
請輸入程式、確認動作看看。

第10章

用 Instruction list 製作程式

10-1 概要

Insturuction list Edit 是用文字編輯的自由語言編輯。Insturuction list 的指令符合 61131-3 標準規格。

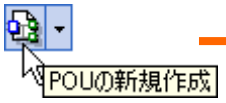


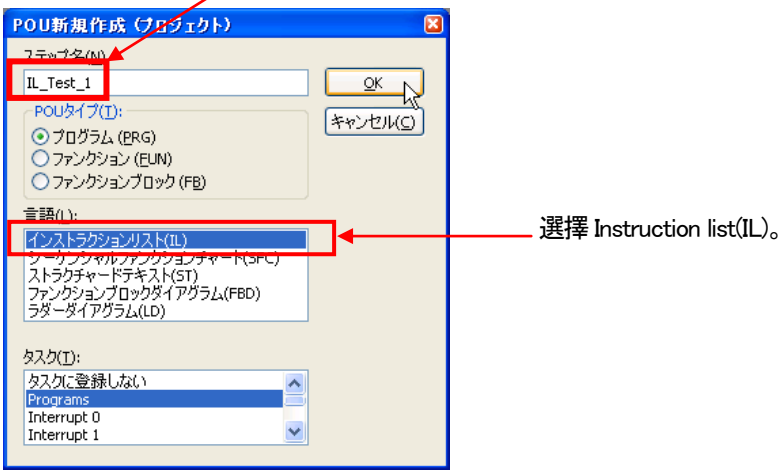
POU 程式區用 Block 來分別。Block 名稱顯示在 Edit 右邊的 Block Header。
中斷點、Block 的選擇、錯誤訊息等的状態情報會在此顯示。

而且在 IL Edit 用 Windows 所使用文字編輯器的標準編集功能
(剪下, 複製, 貼上, 尋找, 取代等)都可以使用。

10-2 製作 Project

製作新 POU。





如上圖、

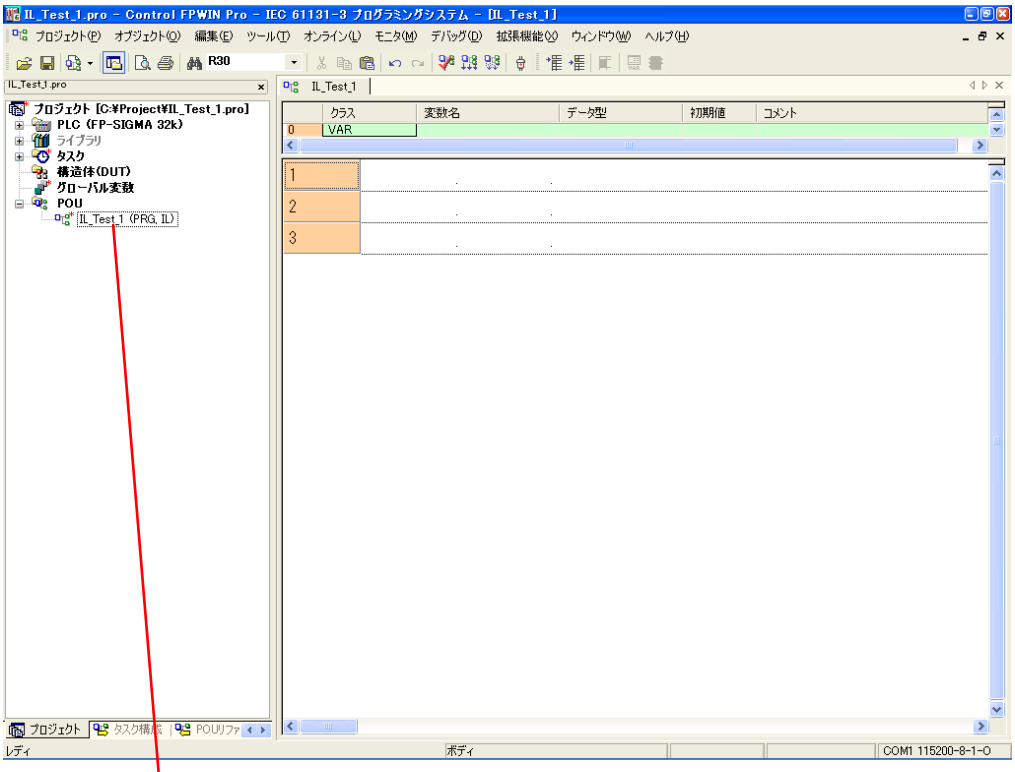
POU 類型：程式(PRG)

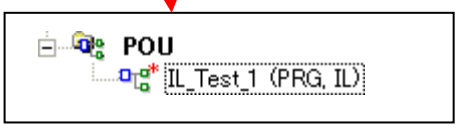
程式語言：Instruction list (IL)

Task：Programs

選擇後按下「OK」鍵。

會顯示下圖畫面。

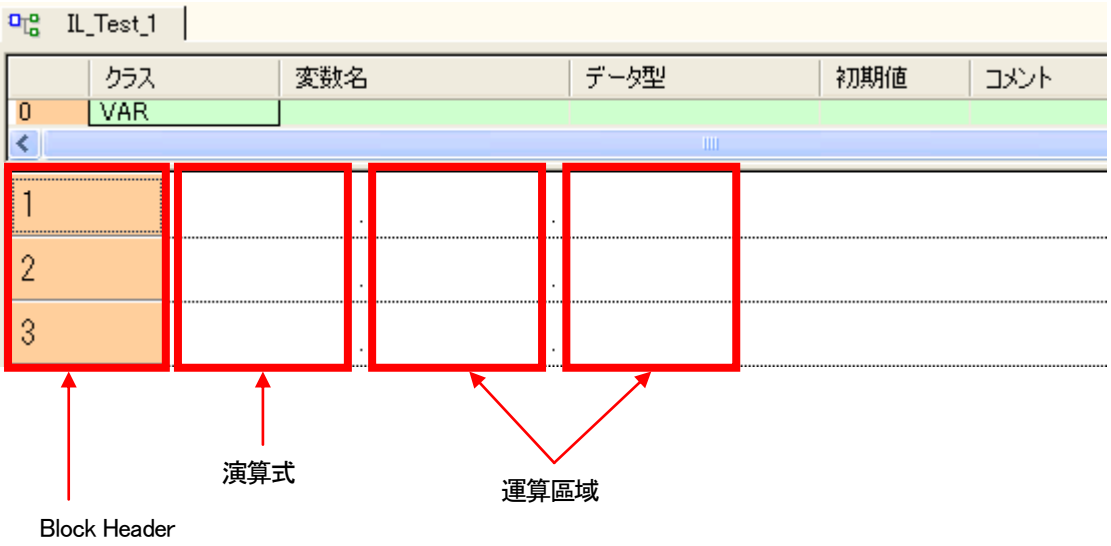




POU 名稱顯示為「IL_Test_1」。

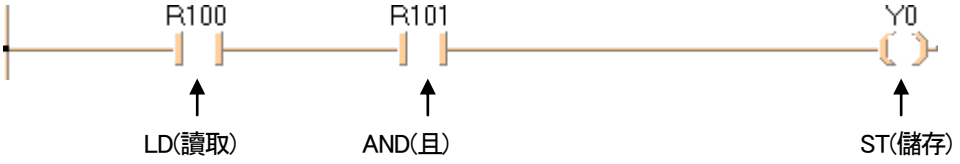
10-3 製作 IL 程式

●程式區細部

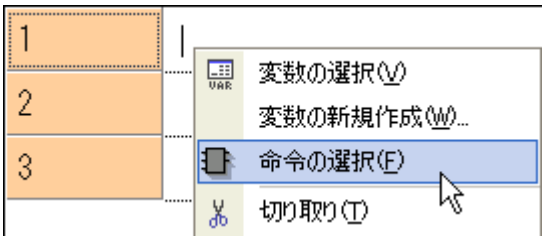


ILNetwork 通常從讀取演算式(LD)開始。連結結果儲存到記憶體。
但是記憶體的內容在移到下一個 Network 處理時會消失。
連接結果之後要使用時、必須要在其他 Network 執行前、將連結結果暫存到變數或是暫存器內。

下圖的階梯圖回路以 IL 編寫程式做為範例。

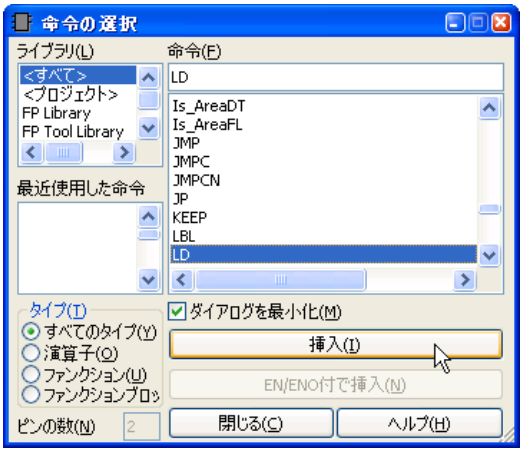


①輸入 LD R100。



在 Network1 的演算式區域按下滑鼠右鍵、選擇「指令的選擇 」。

②輸入 LD



從「OP/FUN/FB 的選擇」對話框、選擇“LD”按下「插入」鍵。

1	LD
2	
3	

※“LD”也可以直接用鍵盤輸入。

③輸入 R100

1	LD R100
2	
3	

用“TAB 鍵”將游標移動到右邊的運算區域、用鍵盤輸入 “R100”。

④增大 Network 行寬

1	LD R100
2	
3	

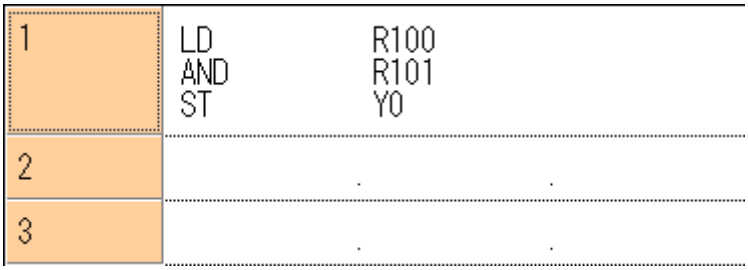
在“R100”的右邊位置有游標的状態下、按下 “Enter”按鍵。

⑤輸入“AND R100”。

1	LD R100 AND R101
2	
3	

依上述、使用相同方式輸入。

⑥輸入“ST Y0”。



在此用上述相同方式增加列寬。

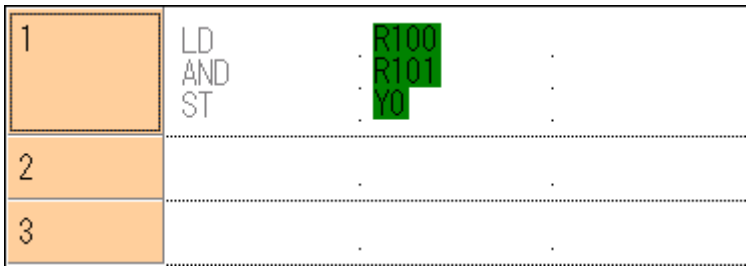
到此程式已經完成。
以下為程式的說明。(累加器是運算用的記憶體)

LD	R100	R100 的狀態存放到累加器中。
AND	R101	R101 的狀態和累加器進行 AND 演算。結果存放到累加器。
ST	Y0	累加器存放到 Y0。

如上面說明也有提到、在 IL 語言中、
存放到累加器、和累加器進行演算、由累加器存取等情況、
使用稱為累加器的運算用記憶體來執行程式。

●動作確認

輸入上圖的程式之後、編譯後確認動作看看。

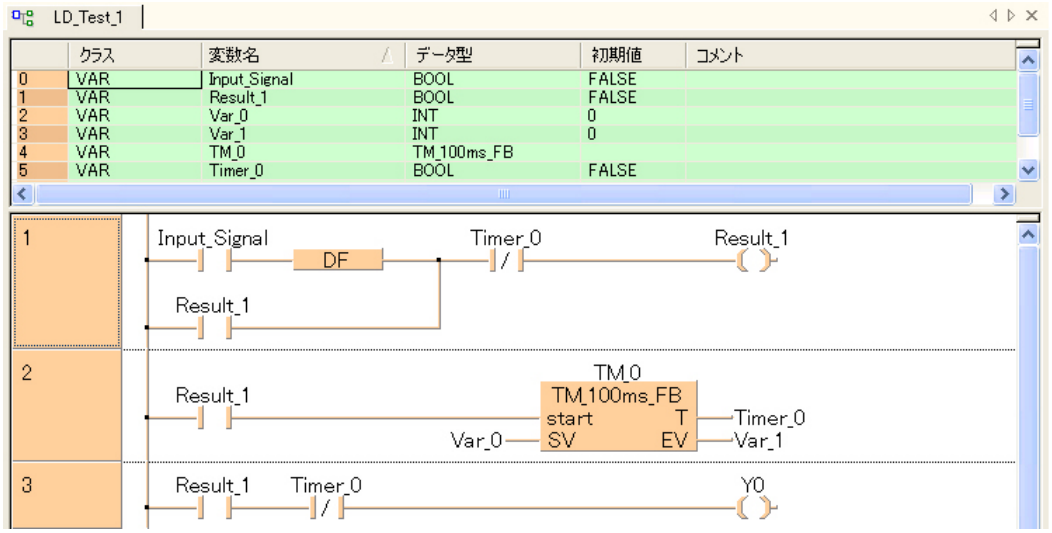


10-4 將 LD 程式更換為 IL 程式

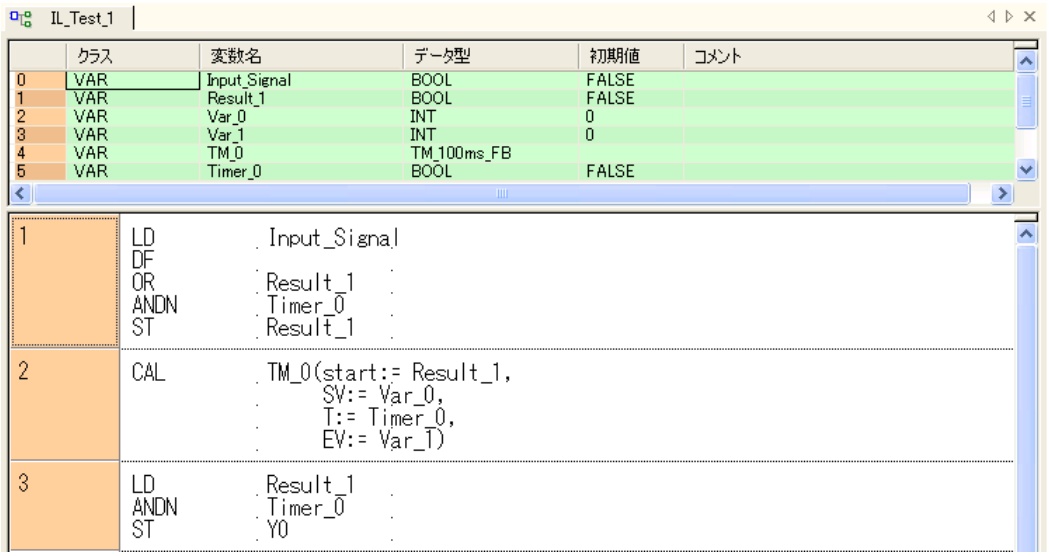
在上一章所作好的 LD 程式轉換為 IL 程式看看。

■LD 程式

上一章所作好的程式。



■IL 程式



輸入程式、確認動作看看。

第11章

用 Structured Text 製作程式

11-1 概要

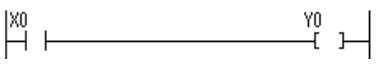
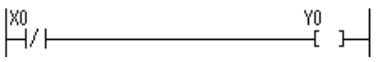
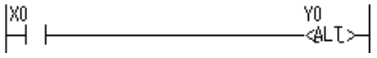
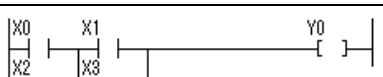
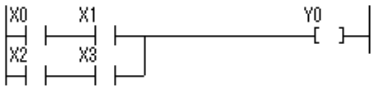
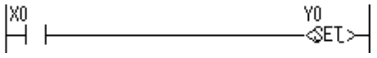
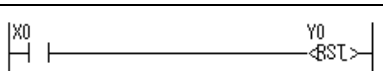
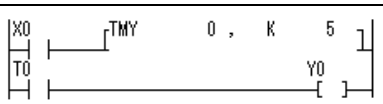
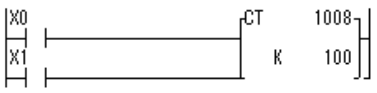
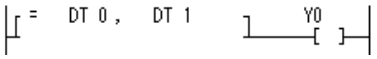
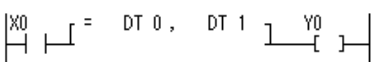
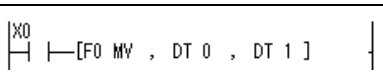
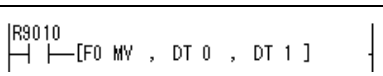
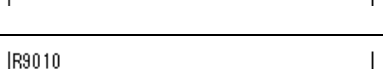
Structured Text 程式(ST)和階梯圖等的圖形語言有所不同、是使用文字形式來製作的程式語言。
和其他語言(LD、IL、SFC、FBD)同樣、敝公司的 PLC 全部機種都有對應。

```
IF D>=0 THEN
    X :=A;
ELSIF codition1 THEN
    X :=A+B;
ELSE
    X :=A+B-C;
END_IF;
```

而且在 ST Edit、可以使用 Windows 所使用的文字編輯的標準功能編集(剪下、貼上、尋找、取代等)。
更進一步的使用指令輸入補助功能(模板)、可以非常簡單的輸入 Function 和 Function Block、IF 句和 CASE 句等。

不過因為程式編輯的記述方法有限制、請閱讀在後述的「11-4-7 ST 語言的注意事項」。

11-2 LD(階梯圖)和 ST 的比對表(例)

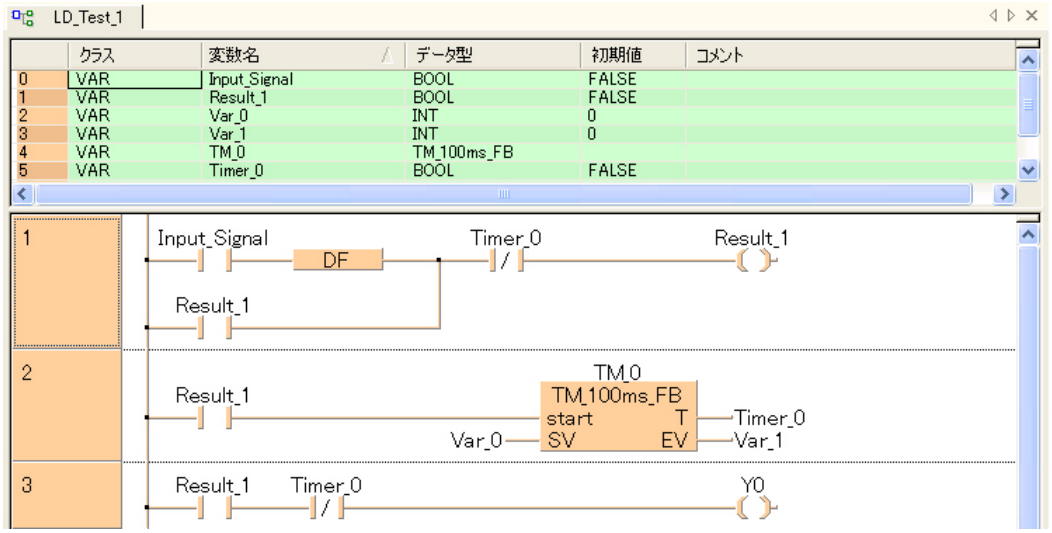
LD 圖示	名稱	ST 碼
	START	Y0 : = X0 ;
	START NOT	Y0 : = NOT(X0) ;
	AND	Y0 : = X0 AND X1 ;
	OR	Y0 : = X0 OR X1 ;
	Alternate OUT	Y0 : = ALT(X0) ;
	ANS	Y0 : = (X0 OR X2) AND (X1 OR X3) ; (在 ST、ANS 指令不可使用)
	ORS	Y0 : = (X0 AND X1) OR (X2 AND X3) ; (在 ST、ORS 指令不可使用)
	上微分	Y0 : = DF(X0) ;
	下微分	Y0 : = DFN(X0) ;
	SET	IF X0 THEN Y0 : = TRUE ; END_IF ; (在 ST、沒有 SET 指令)
	RST	IF X0 THEN Y0 := FALSE ; END_IF ; (在 ST、沒有 RST 指令)
	ON DELAY TIMER	TM_1s_FB(Start :=X0 , SV :=5 , T => Y0, EV => DT0) ; TMX 時 : TM_100ms_FB TMR 時 : TM_10ms_FB TML 時 : TM_1ms_FB
	COUNTER	CT(Count:= X0 , Reset:= X1 , Num:= 1008 , SV:= 100) ;
	16Bit DATA 比較 (START)	Y0 : = (DT0 = DT1) ;
	16Bit BATA 比較 (AND)	Y0 : = X0 AND (DT0 = DT1) ;
	16Bit DATA 傳送指令	IF X0 THEN F0_MV(DT0, DT1) ; END_IF ;
	16BitDATA 傳送指令 (時常執行)	F0_MV(DT0, DT1) ; 或是 DT1 : = DT0 ;
	32BitDATA 傳送指令 (時常執行)	F1_DMV(DDT0, DDT2) ; 或是 DDT2 := DDT0 ; (兩個字元、記述為 DDT**)

11-3 將 LD 程式轉換成 ST 程式

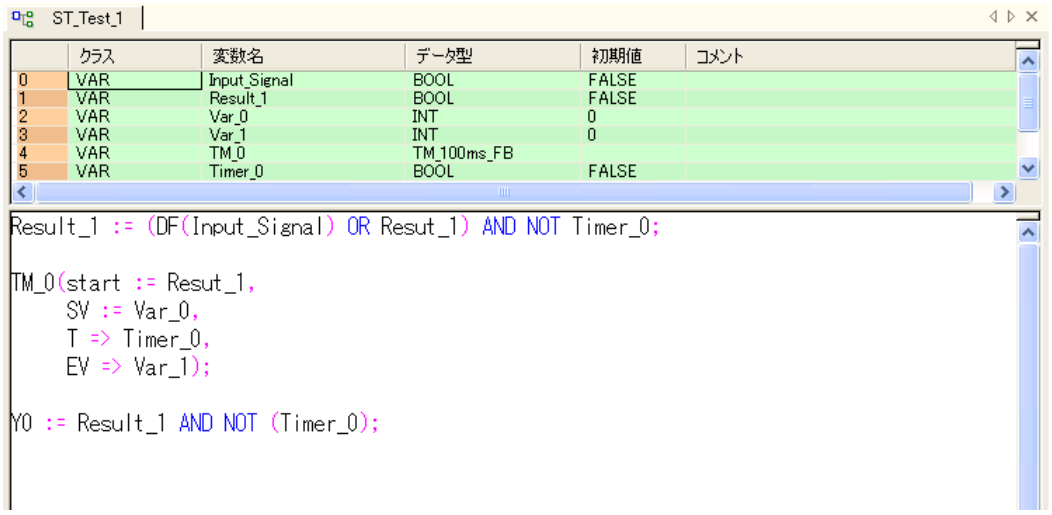
將上一章所作好的 LD 程式轉換成 ST 程式看看。

■LD 程式

上一章所作好的程式。



■ST 程式



輸入程式、請確認動作看看。

11-4 用 ST 製作程式

11-4-1 準備 ST Edit

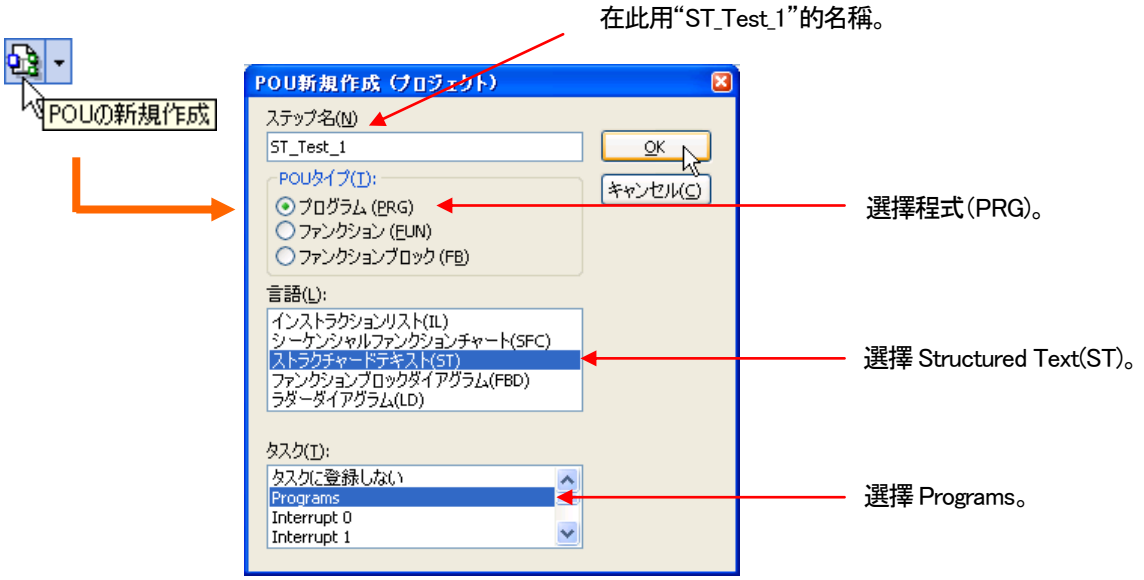
起動 ST Edit、有以下 2 個方法。

1.起動 FPWIN Pro 製作 ST 語言的 POU。
2.編集中的 Project 中追加 ST 語言的 POU。

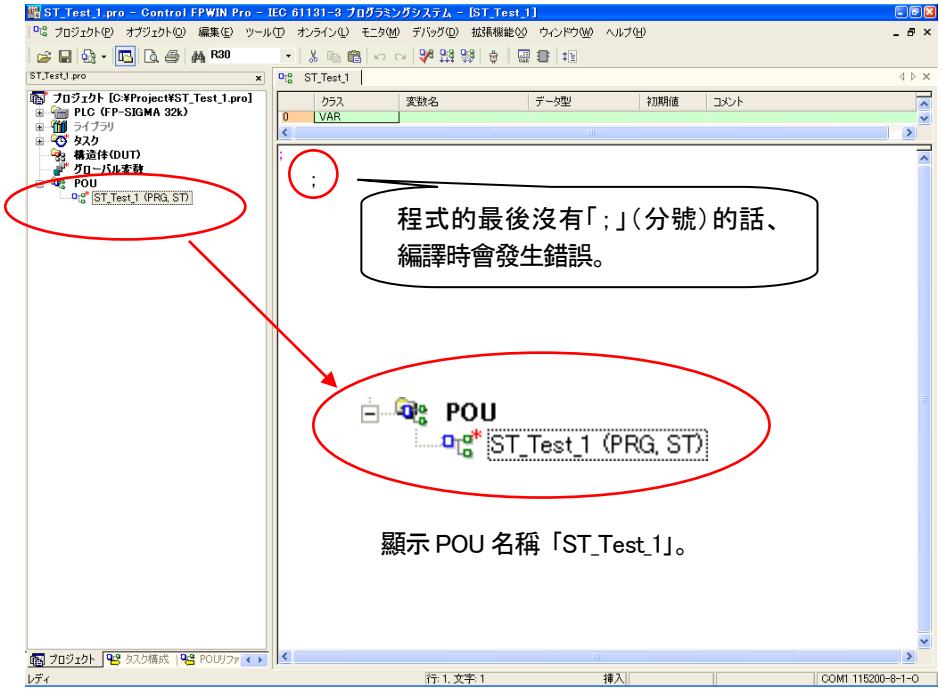
在此説明第 2 個方法。

■編集中的 Project 追加 ST 語言的 POU。

① 點選「製作新 POU」。
② 如下圖輸入名稱並進行選擇。



按下「OK」、會顯示編集用的 Edit。



11-4-2 ST 的寫法

在 ST Edit、用鍵盤輸入文字製作程式。

〈例〉

	クラス	変数名	データ型	初期値	コメント
0	VAR	Initial_SW	BOOL	FALSE	
1	VAR	Data_Area	INT	0	
2	VAR	Timer_1	TM_100ms_FB		
3	VAR	Start_SW	BOOL	FALSE	
4	VAR	Timer_ON	BOOL	FALSE	
5	VAR	KEIKA_VAL	INT	0	
6	VAR				

```
(* 初期化スイッチONでデータエリアを初期化する *)
IF (Initial_SW) THEN
    Data_Area := 0;
END_IF;
(* 起動スイッチがONしたらタイマをONさせる *)
Timer_1(start := Start_SW,(* Start_SWでタイマを起動する *)
        SV := 50, (* 5秒後にTimer_ONリレーがON *)
        T => Timer_ON,
        EV => KEIKA_VAL);(* タイマ起動中に減算される *)
```

註解
條件
指令

```
IF Initial_SW THEN
    DATA_AREA :=  0;
END_IF;
```

IF 以及 FOR 等的指令在下一個句子之間必須間隔 1 個以上的空格。
演算式的情況則不需要。
(如:DATA_AREA:=0)

```
IF Initial_SW THEN DATA_AREA := 0;
END_IF;
```

左邊的條件句子、這樣的寫法也沒關係。
指令的最後請務必要加上分號「;」。

【實習】

請輸入 11-3 頁上的 ST 的程式、確認動作看看。

11-4-3 輸入指令

在此實際用 ST 輸入各種指令。

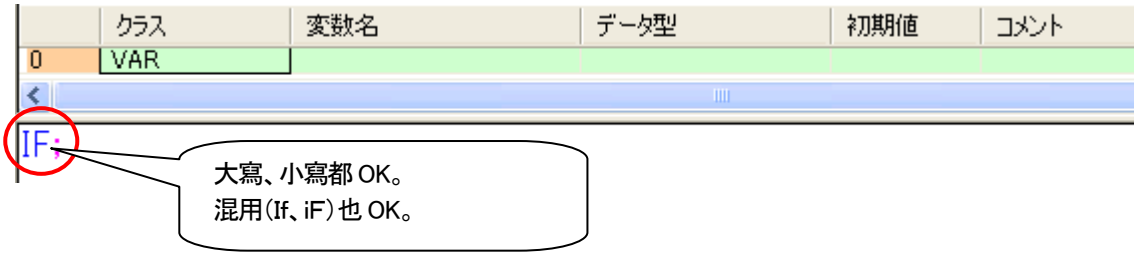
■輸入指令

在 ST 所支援的各種指令基本都由鍵盤直接輸入、
如使用“插入用模板(快速鍵)”編集是非常快速且容易的。

<例>1.輸入 “IF”(條件分岐)指令。

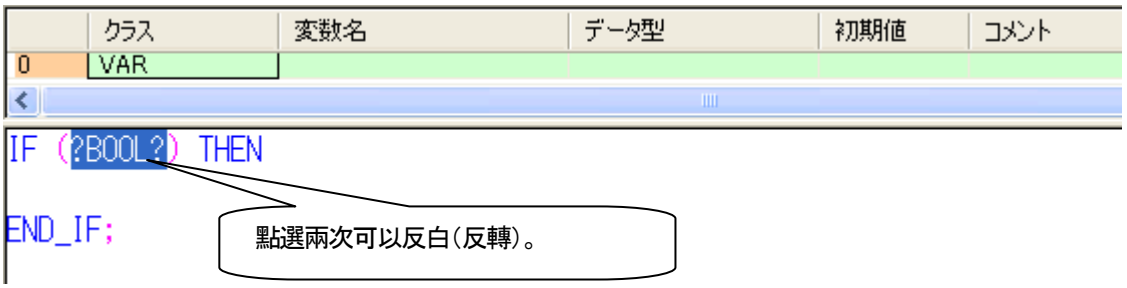
●操作步驟

1. 在 ST 編集画面、最初用鍵盤輸入“IF”。



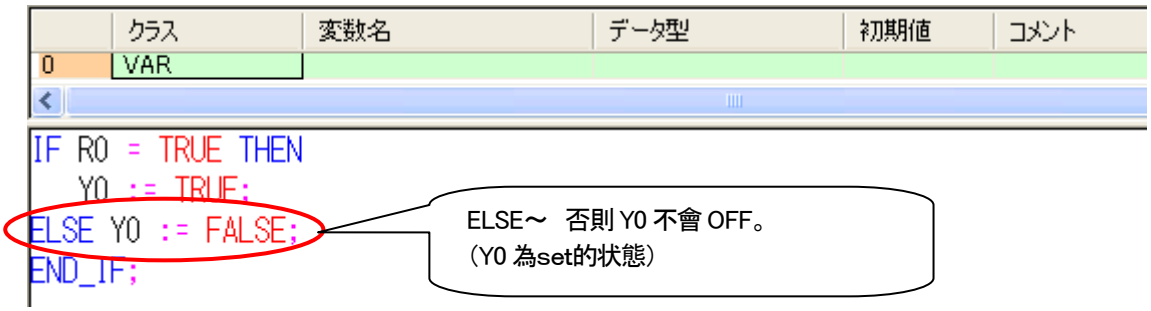
2. 在“IF”指令的 “F” 後面的游標按下空白鍵。

在此會顯示如下圖的 IF 文。
被“?”所包圍的部分輸入任意的變數以及元件。



3.請在 “IF”～“END_IF” 之間進行編集。

【例】R0 為 ON 的時候、Y0 會 ON。

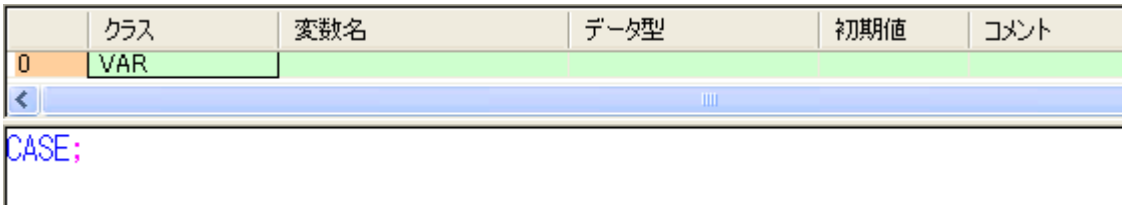


【課題】
請製作 SW_1(BOOL 型變數)如果 ON、Y0 為 ON (Y1 為 OFF)、SW_1 如果 OFF、Y1 為 ON(Y0 為 OFF)的程式。

<例>2. 輸入“CASE”(複數選擇)指令。

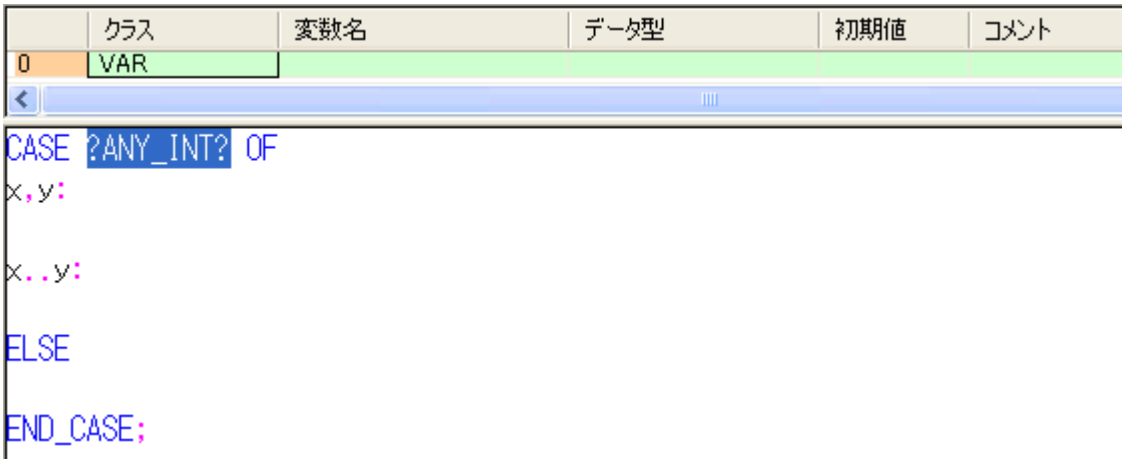
●操作步驟

1. 在 ST 編集画面、最初用鍵盤輸入 “CASE”。



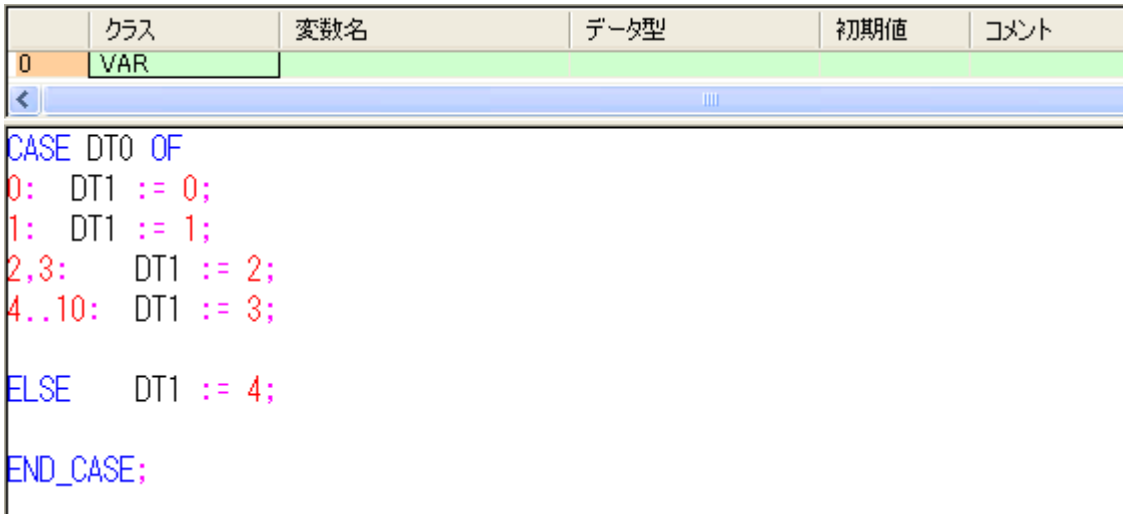
2. “CASE”指令的 “E” 後面用游標標示後、按下空白鍵。

在此會顯示如下圖的 CASE 文。
被“?”所包圍的部分輸入任意的變數以及元件。



3. 請在“CASE”～“END_CASE”之間進行編集。

【例】DT0 為 0 的時候、DT1 為 0、DT0 為 1 的時候、DT1 為 1、DT0 為 2、3 的時候、DT1 為 2、DT0 為 4～10 的時候、DT1 為 3、DT0 為 0～10 以外的時候、DT1 為 4。



【課題】

請製作 Data_1(INT 類型變數)為 1、3、5、7、9 的時候、Data_2(INT 類型變數))為1、Data_1 為 0、2、4、6、8、1 的時候、Data_2 為 2、Data_1 為 11 以上的時候、Data_2 為 3 的程式。

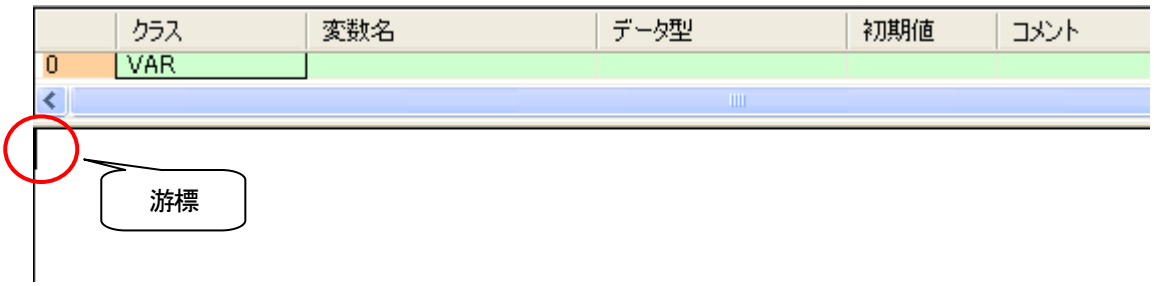
■輸入 OP/FUN/FB

在 ST 編集畫面的 OP(演算式)/FUN(Function)/FB(Function Block)的輸入方法如以下所示。

<例> 輸入資料傳送指令 F0(MV)指令。
(此例為輸入[F0(MV) DT0, DT1])

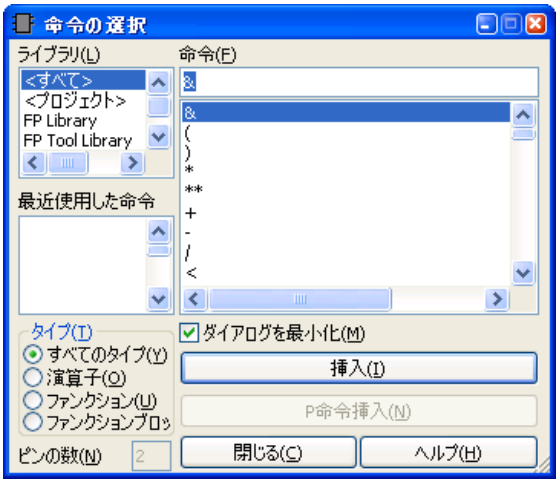
●操作步驟

1. **游標設在 ST 編集畫面的指令輸入位置。**



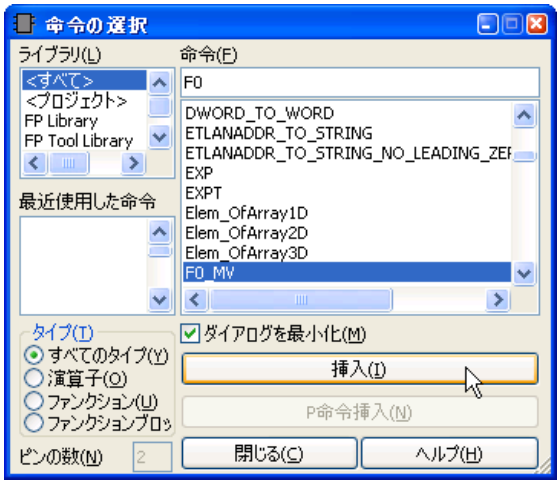
2. **點選工具列上的 按鍵、或是選擇(選單)工具 → 指令的選擇。**

顯示以下的對話框。

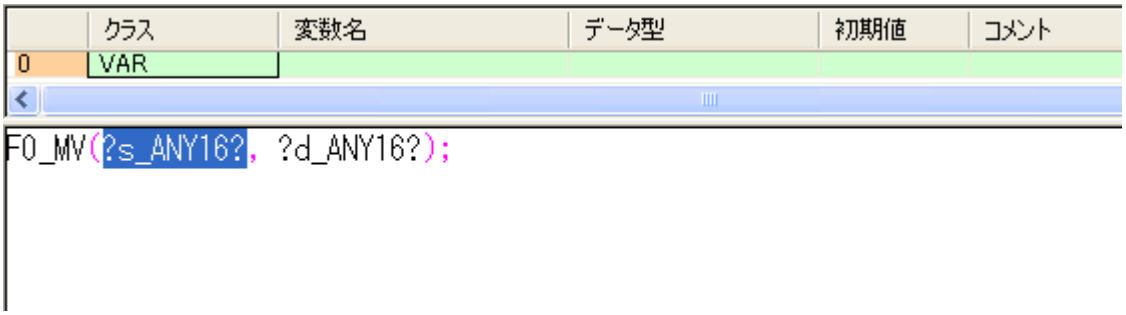


3. **在「Library」的選擇區域從「FP Library」之中選擇「F0_MV」、點選兩次或是按下「插入」按鍵。**

（雖然 Library 選擇區域用初始值<全部>選擇、
不過也可以直接將“F0_MV”指令存放在 List 之中）

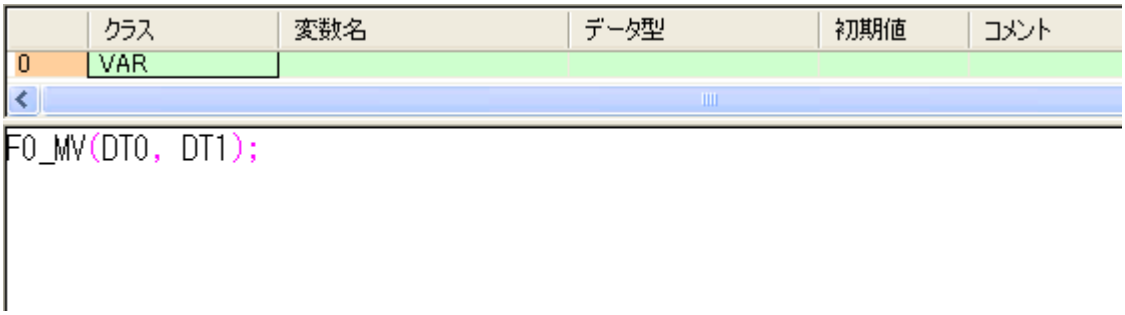


4. **如下插入 F0_MV 指令。**



＊上面的 2, 3 不操作、直接用“F0”＋空白鍵也能顯示出 4。

5. **被“?”所包圍的部分（顯示輸入資料類型）輸入資料名稱則完成。**



11-4-4 用 ST 可以使用的運算/指令/演算式

■用 ST 可以運算和資料型態如以下表。
這些運算是演算式,和指令一起使用。

名稱	資料類型	例
定值	值 字串 時間	55,-3.14159 'This is a sample text' T#8d_3h_23m
變數	變數 陣列的要素 DUT(結構)的要素 結構內的陣列要素	Var1 Array[15] Dut1.Var1 Dut1.Array[i+5]
FUNCTION	FUNCTION CALL	Fun1(a,b,c)

用 ST 撰寫 PLC 元件(定義全域變數, 在程式中直接使用)時、
請用"全部大寫"。
(用小寫記述在編譯時會被認為是變數、如果不進行變數宣告會變成錯誤)

	名稱	例(全部大寫記述)	補充
R E L A Y	外部入力 X	X0	
	外部出力 Y	Y0	
	内部 RELAY R	R0	
	連結 RELAY L	L0	
	計時器/計數器 T/C	T0/C200	
記憶體區域	外部輸入 WX	WX0	2 個字元指定時は DWX0
	外部輸出 WY	WY0	2 個字元指定時は DWY0
	内部 RELAY WR	WR0	2 個字元指定時は DWR0
	連結 RELAY WL	WL0	2 個字元指定時は DWL0
	資料暫存器 DT	DT0	2 個字元指定時は DDT0
	連結暫存器 LD	LD0	2 個字元指定時は DLD0
	計時器/計數器 SV 設定値區域	SV0	2 個字元指定時は DSV0
	計時器/計數器 EV 經過値區域	EV0	2 個字元指定時は DEV0
	索引暫存器 I	IX,IY	I0-ID 不可使用
位數	10 進位數	100,1000,255,-15	直接記述數字
	16 進位數	16#0000,16#50AC,16#FFFF	前面附上「16#」
	10 進位數(浮點小數型)	0.0,100.0,12.3,-0.5	附上小數點作為實數。

資料類型	内容	Bit 數	範圍
BOOL	0:FALSE,1:TRUE	1	-
DINT	倍精度整數	32	-2147483648 ～ 2147483647
DWORD	32Bit 文字列	32	-
INT	整數	16	-32768 ～ 32767
REAL	實數	32	-3.402823*E38 ～ -1.175494*E-38, 0.0 +1.175494*E-38 ～ +3.402823*E38
STRING	字串	-	最大 255 文字
TIME	連接時間	16(FP1,3 など) 32(FP0,Σ,2/2SH,10SH)	0.01 ～ 327.67 秒 0.01 ～ 21474836.47 秒
WORD	16Bit 字串	16	-

■在 ST 可使用的指令一覽如以下所示。

指令	例	說明
:= (代入)	y := (a + b + c)/3 ;	右邊的值代入到左邊的變數。 此例為、a,b,c 的平均值代入到 y。
(呼叫 FUN)	Y := SIN(x) ;	呼叫單純給 1 個參數的 FUNCTION。 此例為計算輸入變數 x 的正弦代入到變數 Y。
	Y := LIMIT(MN :=0,IN :=X,MX :=100) ;	呼叫從參數代入的 FUNCTION。 此例為、參數 IN 為 MN 和 MX 之間的值的話、就代入到 Y。
(呼叫 FB)	Ton1(IN:=Start1, PT:=T#300ms, End1:=Ton1.Q ; Ev1:=Ton1.ET) ;	呼叫直接代入輸入參數的 FB。 在此例、變數 Start1 為 ON 時 300ms 之後、變數 End1 會 ON。 經過值會代入到變數 Ev1。

IF
(條件分岐)
	IF a>=0 THEN
	b :=0 ;
	ELSIF a>=100 THEN
	b :=1 ;
	ELSE
	b :=2 ;
	END_IF ;
		依所記述的真理值、執行分岐的代入文。
		使用 IF 時、必須輸入 END_IF。

CASE (複數選擇)	CASE a OF 0 : b :=0 ; 1,2 : b :=1 ; 3,4,10..20 : b :=2 ; 100..110 : b :=3 ; ELSE b :=4 ; END_CASE;	依所記述的變數值、選擇要執行的文句。 變數 a 必須為、INT 型(或是 DINT 型)。
FOR (回圈)	FOR i:=0 TO 100 DO SUM :=SUM+a[i] ; END_FOR;	使用回圈執行。 此例為 i 每次增加 1、到 100 則結束處理。
	FOR i:=0 TO 100 BY 10 DO SUM:=SUM+a[i] ; END_FOR;	此例為 i 每次增加 10、到 100 則結束處理。
WHILE (回圈)	i:=0 ; WHILE i<=100 AND a[i]<100 DO i:=i+10 ; END_WHILE;	WHILE 跟以下條件一致時、會回圈處理。 條件在處理執行前會確認。
REPEAT (回圈)	i:=0 ; REPEAT i:=i+10 ; UNTIL I >100 OR a[i] >=100 END_REPEAT ;	UNTIL 滿足以下條件前、會回圈處理。 條件在處理執行後會確認。
EXIT (中止)	EXIT ;	無條件終止回圈處理。
RETURN (回到 JUMP)	RETURN;	回到呼叫的原 POU 程式。

這些指令可以在全部的 PLC 使用。

■在 ST 所可以使用的指令一覧如以下所示。

演算式	可用資料	例	結果	演算順位
( )		(1+2)*(3+4)	21	最上位
**	REAL	3.0**2.0	9.000435	
- (符号反轉) NOT (論理否定)	INT,DINT,REAL BOOL,WORD,DWORD	-DATA0 (DATA0 作為 100) NOT DATA0 (DATA1 作為 16#0064)	-100 16#FF9B	
*(乘算) /(除算) MOD(求餘)	INT, DINT, REAL	10*5 20/4 12 MOD 10	50 5 2	
+(加) -(減)	INT, DINT, REAL	10 + 20 5 - 1 2 - 10	30 4 -8	
> < >= <= (比較)	ANY (ANY_BIT は不可)	1 > 3 1 < 3 5 >= 5 2 <= 1	FALSE TRUE TRUE FALSE	
=(等式) <>(不等式)	ANY	X0 = X1 (X0:ON 状態, X1:OFF 状態) 100 <> 100	FALSE FALSE	
&,AND (且)	BOOL, WORD, DWORD	16#A5F8 AND 16#B0B0	16#A0B0	
XOR (異或)	BOOL, WORD, DWORD	16#A5F8 XOR 16#B0B0	16#1548	
OR (或)	BOOL, WORD, DWORD	16#5555 OR 16#AAAA	16#FFFF	最下位

這些演算式可以使用在全部的 PLC。
(不過可以用 REAL 的只有 FP-e,FP0, FP Σ, FP2/2SH)

【補充】 並且、三角函數(SIN, COS, TAN 等)和指數, 對數, 開平方等的算術指令被作為 IEC 指令所定義、
可以使用在 FP-e,FP-X, FP0, FP Σ, FP2/2SH(可用實數的 PLC)。

演算式	可使用資料類型	例	結果
ABS (絕對值)	ANY_NUM	Result := ABS(-5.0) (以下、Result 為 REAL 型的變數)	5.0
SQRT (開平方)	REAL	Result := SQRT(4.0)	2.0
LN (自然數)	REAL	Result := LN(100.0)	4.605168
LOG (對數)	REAL	Result := LOG(100.0)	1.999999
EXP (自然指數)	REAL	Result := EXP(3.0)	20.08554
SIN (正弦)	REAL (角度資料用弧度指定)	Result := SIN(1.0)	0.841471
COS (餘弦)	REAL (角度資料用弧度指定)	Result := COS(1.0)	0.5403023
TAN (正切)	REAL (角度資料用弧度指定)	Result := TAN(1.0)	1.557408
ASIN (逆正弦)	REAL (角度資料用弧度指定)	Result := ASIN(1.0)	1.570796
ACOS (逆餘弦)	REAL (角度資料用弧度指定)	Result := ACOS(1.0)	0.0
ATAN (逆正切)	REAL (角度資料用弧度指定)	Result := ATAN(1.0)	0.7853

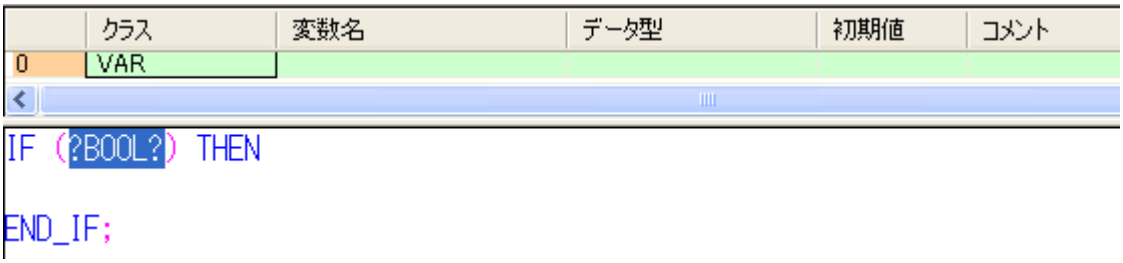
並且、和這些同等指令在 FPWIN Pro 系列指令(FP Library)也有定義。
無論使用那個都不會有問題。

11-4-5 使用變數

和 LD 等的其他編集畫面相同的、可以使用 ST Edit 編集變數。
可以使用已登録的變數、也可以在程式中登録新的變數。

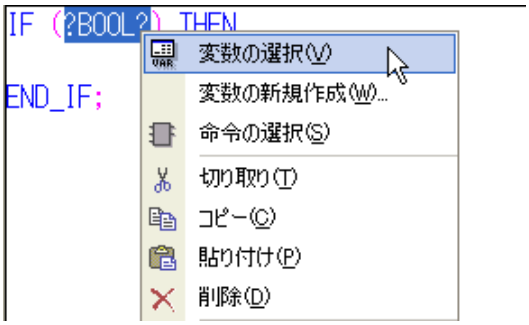
<例>1. 使用已登録的變數。

在以下的 ST 程式、輸入變數到"?BOOL?"。

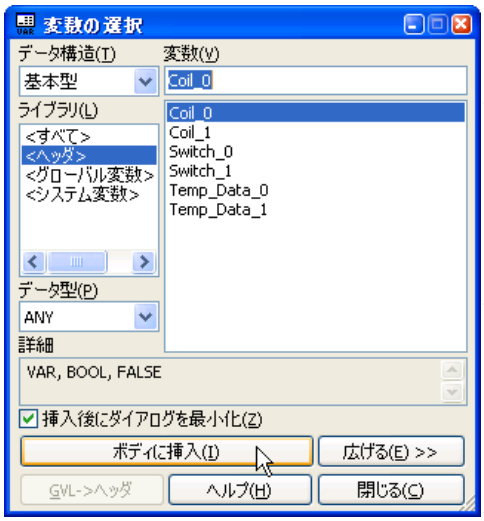


■操作步驟

1. **點選"?BOOL?"兩次、**
   點選工作列上 按下<F2>按鍵、或是用滑鼠右鍵選擇「變數的選擇」。

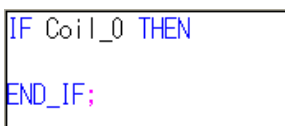


2. **會顯示以下的對話框、選擇任意的變數並點選兩次或是點選「插入到程式區」。**



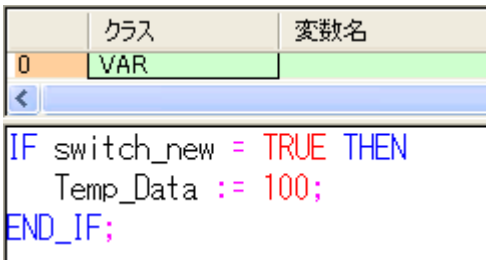
預先登録幾個變數。

3. **輸入變數到 ST 編集畫面。**



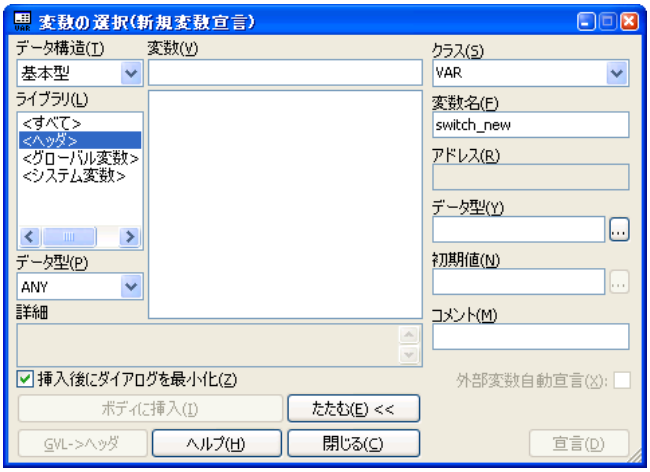
<例>2. 程式中登録新變數。

將以下的 ST 程式、編集新的變數 “switch_new”登録到 Header。
（新變數未登録的時候、檢查程式時會發生錯誤）



■操作步驟

1. **點選“switch_new”兩次、**
選擇、(選單)工具 → 製作新變數、或是按滑鼠右鍵選擇「製作新變數」、
會顯示以下畫面、請輸入「變數名稱」「資料型態」「初始值」。



2. **此例的變數名稱為 “switch_new”，資料型態為″BOOL″，初始值設定為″FALSE″。**

資料型態、按下輸入區域旁邊的 … 按鍵、會顯示選擇資料型態的對話框。

直接輸入 “BOOL”到輸入區域也 OK。

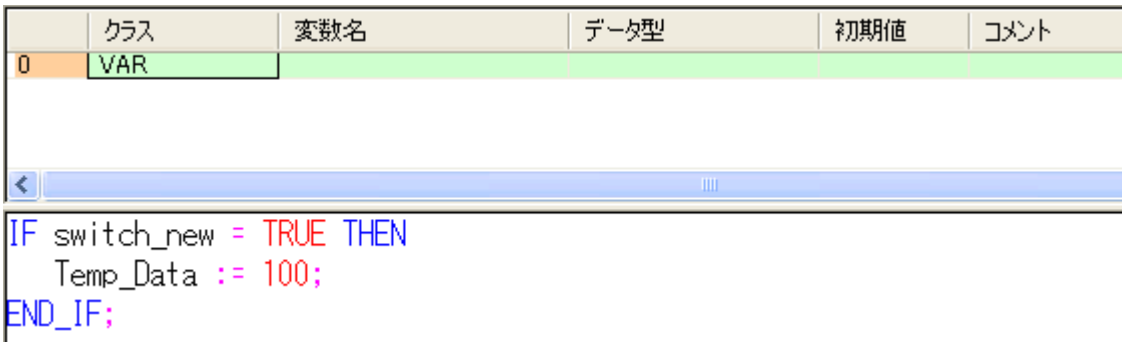
<例>3. 登録變數到 Header。

預先將變數登録到 POU Header 後、編輯程式作業會變得較順利。
以下為登録變數到 POU Header 的步驟。

■操作步驟

1. 顯示 Header。

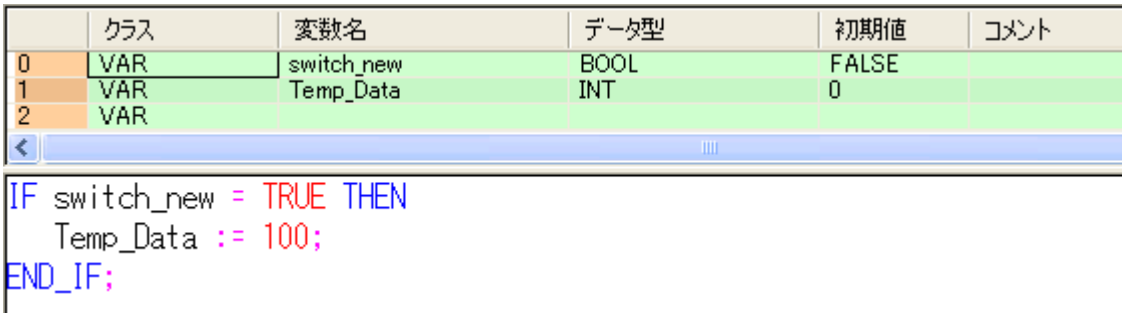
請輸入變數名稱, 資料型態, 初始值。(備註為任意)
舉例設定、
變數名稱:switch_new, 資料類型:BOOL, 初始值:FALSE
變數名稱:Temp_Data, 資料類型:INT, 初始值:0



	クラス	変数名	データ型	初期値	コメント
0	VAR				

```
IF switch_new = TRUE THEN
   Temp_Data := 100;
END_IF;
```

2. 輸入後、按下工具列上的 按鍵、或是選擇(選單)元件 → 檢查
請確認登録沒有問題。



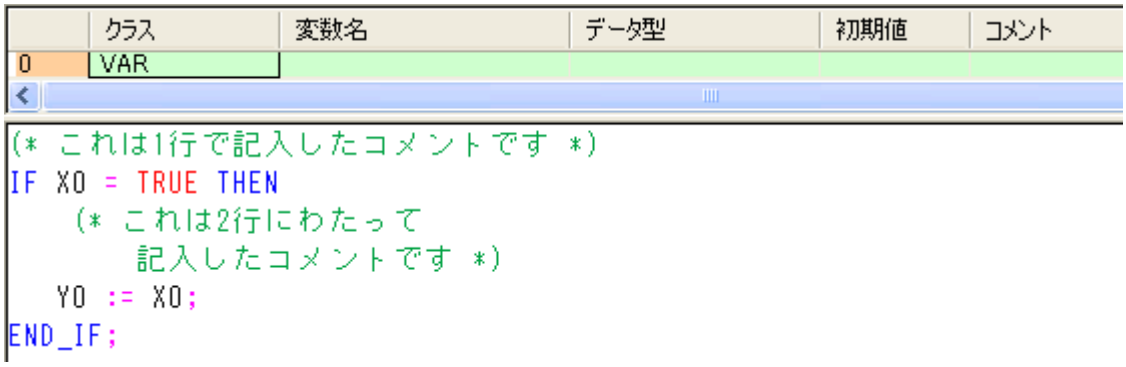
	クラス	変数名	データ型	初期値	コメント
0	VAR	switch_new	BOOL	FALSE	
1	VAR	Temp_Data	INT	0	
2	VAR				

```
IF switch_new = TRUE THEN
   Temp_Data := 100;
END_IF;
```

11-4-6 輸入註解

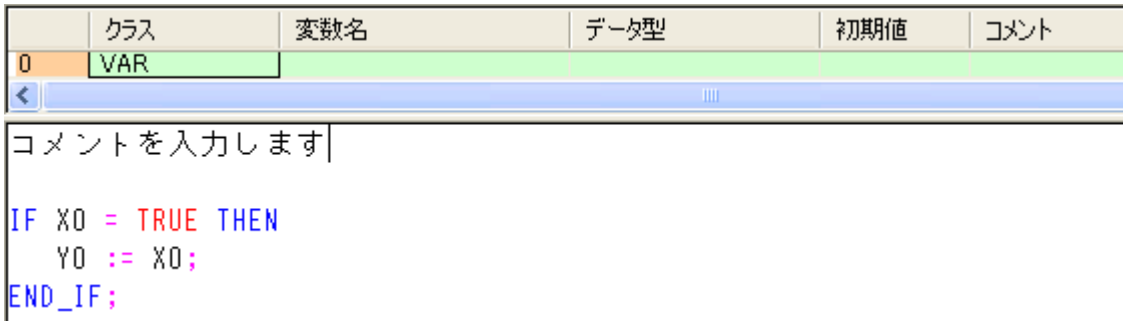
在程式區的任意行中、可以輸入註解。
註解請使用跨弧加上米字符號“(*” 和 “*)” 並前後包住。

<例>

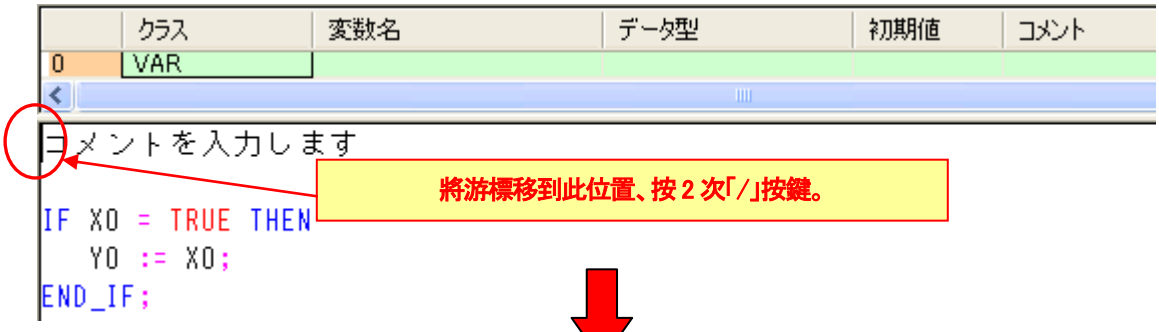


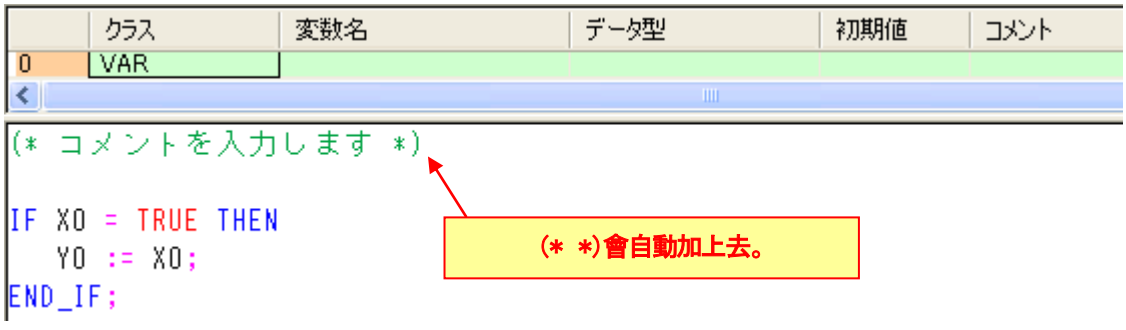
■操作步驟

1. 輸入文字。



2. 將游標移到已輸入的文字前面、按 2 次「/」按鍵。

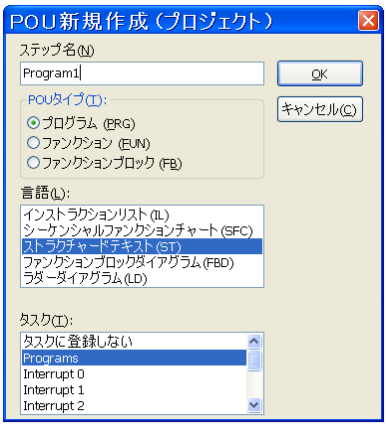




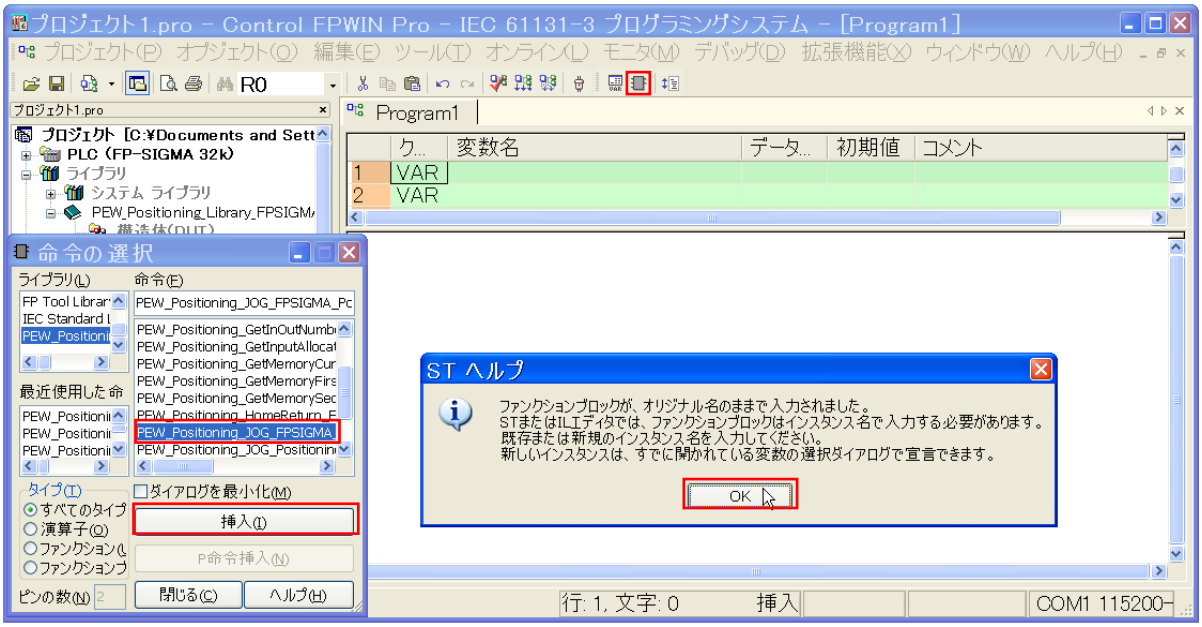
11-5 在 ST 程式語言使用 Function Library

■在第 8 章所介紹過的 Function Library (**PEW_Positioning_JOG _FPSIGMA_PosUnit 在 FPΣ定位模組進行 JOG 運轉的 FB**) 讀出到 ST 語言的 POU、並製作成 Function Block 看看。

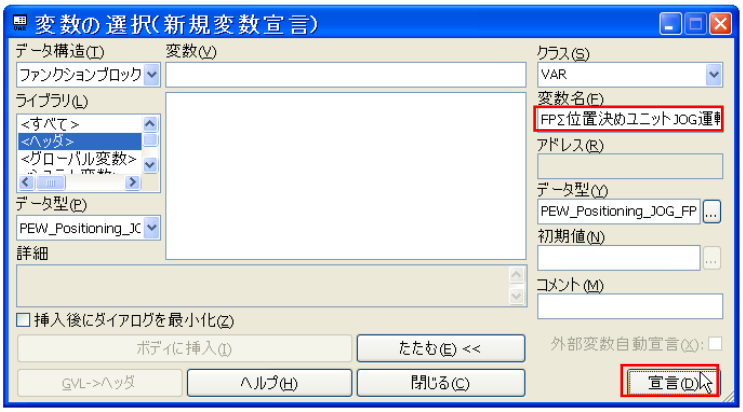
1. 用 POU 類、PRG 型、ST 語言製作新的 POU。



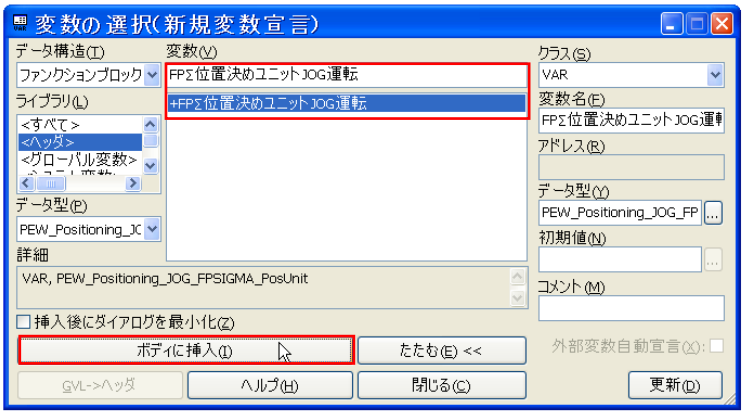
2. 按下 OP/FUN/FB 選擇按鍵、**PEW_Positioning_JOG _FPSIGMA_PosUnit 選擇(用 FPΣ定位模組進行 JOG 運轉的 FB)** 並按下插入按鍵。
用以下索引名稱訊息、選擇 OK 按鍵。



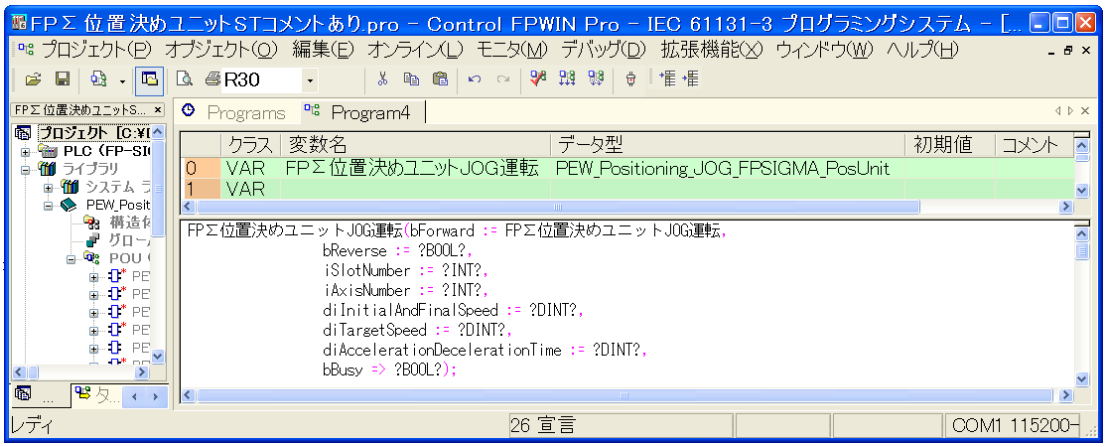
3. 輸入「FPΣ定位模組 JOG 運轉」到變數名稱、(變數名稱可以輸入任意文字。)
   按下宣告按鍵。



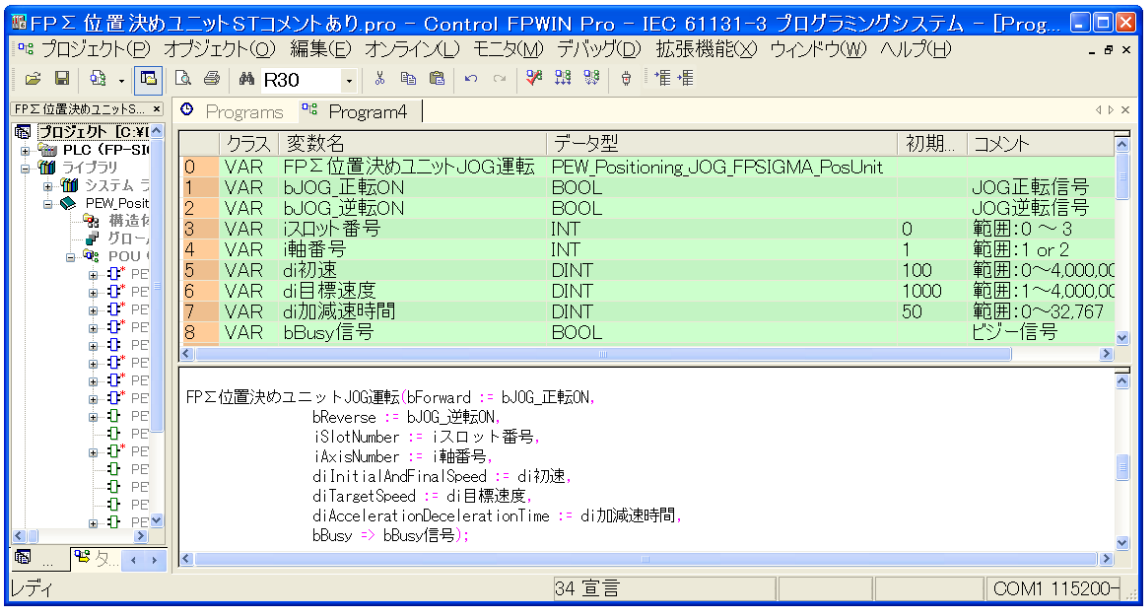
4. 設定好變數名稱後按下插入程式區鍵、再按下關閉鍵關閉視窗。



5. PEW_Positioning_JOG _FPSIGMA_PosUnit 作為變數登録到 Header、
   會顯示如以下的 ST 程式語言。

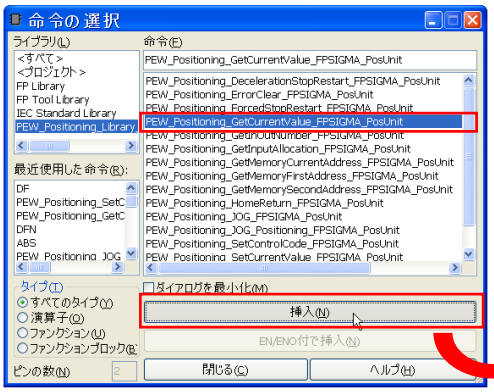


6. 登録以下的變數並配置看看。



■課題 1

PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(從 FPSIGMA 定位模組中讀出經過值的 FUN)讀出到POU、並登錄變數和製作 Function Block。




```
PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit((* EN := TRUE, *)
                                                iSlotNumber := ?INT?,
                                                iAxisNumber := ?INT?
                                                (* , ENO => ?BOOL? *));
```

```
FPΣ位置決めユニットJOG運転(bForward := bJOG_正転ON,
                bReverse := bJOG_逆転ON,
                iSlotNumber := iスロット番号,
                iAxisNumber := i軸番号,
                diInitialAndFinalSpeed := di初速,
                diTargetSpeed := di目標速度,
                diAccelerationDecelerationTime := di加減速時間,
                bBusy => bBusy信号);
```

Function Library Help

變數

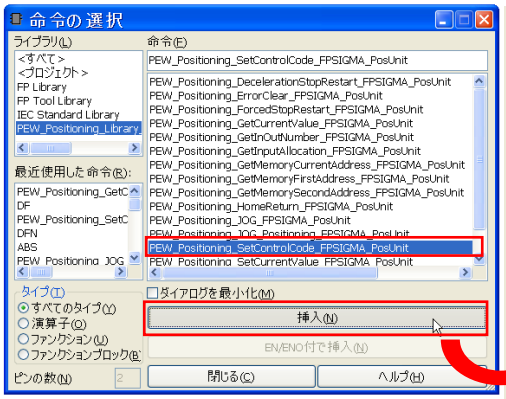
參數	資料型態	備註
iSlotNumber	INT	Slot No. 範圍:0～3
iAxisNumber	INT	軸 No. 範圍:1 or 2

ST 記述範例

```
diCurrentValue := PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(iSlotINumber := 0, iAxisNumber := 1);
```

■課題 2

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(設定 FPSIGMA 定位模組控制碼的 FUN)讀出到 POU、並登錄變數和製作 Function Block




```
PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit((* EN := TRUE, *)
                                                iSlotNumber := ?INT?,
                                                iAxisNumber := ?INT?
                                                (* , ENO => ?BOOL? *));

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := ?BOOL?,
                                                iSlotNumber := ?INT?,
                                                iAxisNumber := ?INT?,
                                                wPositionCommand := ?WORD?,
                                                wAccelerationDeceleration := ?WORD?,
                                                wDirectionOfHomeReturn := ?WORD?,
                                                wStartupSpeed := ?WORD?,
                                                wHomeInputLogic := ?WORD?,
                                                wNearHomeInputLogic := ?WORD?,
                                                wHomeSearch := ?WORD?,
                                                wLimitInputLogic := ?WORD?,
                                                wS_Pattern := ?WORD?,
                                                wRotationDirection := ?WORD?,
                                                wOutputMode := ?WORD?,
                                                wDeviationCounterClearTime := ?WORD?);

FPΣ位置決めユニットJOG運転(bForward := bJOG_正転ON,
                bReverse := bJOG_逆転ON,
                iSlotNumber := iスロット番号,
                iAxisNumber := i軸番号,
                diInitialAndFinalSpeed := di初速,
                diTargetSpeed := di目標速度,
                diAccelerationDecelerationTime := di加減速時間,
                bBusy => bBusy信号);
```

Function Library Help

變數

參數	資料型態	備註
bSetCode	BOOL	控制碼設定信號 範圍:TRUE or FALSE(ON or OFF)
iSlotNumber	INT	Slot No. 範圍:0～3
iAxisNumber	INT	軸 No. 範圍:1 or 2
wPositionCommand	WORD	控制方式 範圍: 16#0000(整數)、16#0001(絕對值)
wAccelerationDeceieration	WORD	加減速方式 範圍: 16#0000(直線加減速)、16#0002(S 字加減速)
wDirectionOfHomeReturn	WORD	原點復歸方向 範圍: 16#0000(經過值-方向)、16#0004(經過值+方向)
wStartupSpeed	WORD	起動速度 範圍: 16#0000(0.02ms)、16#0008(0.005ms)
wHomeInputLogic	WORD	原點輸入理論 範圍: 16#0000(不通電時輸入有效)、16#0010(通電時輸入有效)
wNearHomeInputLogic	WORD	原點近傍理論 範圍: 16#0000(通電時輸入有效)、16#0020(不通電時輸入有效)
wHomeSearch	WORD	原點搜尋 範圍: 16#0000(無效)、16#0040(有效)
wLimitInputLogic	WORD	極限輸入理論 範圍: 16#0000(不通電時輸入有效)、16#0080(通電時輸入有效)
wS_Pattern	WORD	S 字類型 16#0000(Sin 曲線) 16#1000(2 次曲線) 16#2000(圓軌跡曲線) 16#3000(3 次曲線)
wRotationDirection	WORD	回轉方向 範圍: 16#0000(正轉)、16#0100(逆轉)
wOutputMode	WORD	輸出模式 範圍: 16#0000(Pulse/Sign)、16#0200(CW/CCW)
wDeviationCounterClearTime	WORD	偏差計數清除時間 範圍: 16#0000(1ms)、16#8000(10ms)

ST 記述範例

```
PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := sys_bIsFirstScan,
                    iSlotNumber := 0,
                    iAxisNumber := 1,
                    wPositionCommand := 16#0001,
                    wAccelerationDeceleration := 16#0000,
                    wDirectionOfHomeReturn := 16#0000,
                    wStartupSpeed := 16#0000,
                    wHomeInputLogic := 16#0010,
                    wNearHomeInputLogic := 16#0000,
                    wHomeSearch := 16#0040,
                    wLimitInputLogic := 16#0080,
                    wS_Pattern := 16#0000,
                    wRotationDirection := 16#0000,
                    wOutputMode := 16#0200,
                    wDeviationCounterClearTime := 16#0000);
```

■課題 1、2 解答 設定例

	クラス	変数名	データ型	初期値	コメント
0	VAR	iスロット番号	INT	0	範囲:0 ～ 3
1	VAR	i軸番号	INT	1	範囲:1 or 2
2	VAR	di経過値	DINT	0	-2,147,483,648 +2,147,483,647
3	VAR	w制御方式	WORD	16#0000	0:インクリメント 1:アブソリュート
4	VAR	w加減速方式	WORD	16#0000	0:直線加減速 1:S字加減速
5	VAR	w原点復帰方向	WORD	16#0000	0:マイナス方向 1:プラス方向
6	VAR	w起動速度	WORD	16#0000	0:0.02ms 8:0.005ms
7	VAR	w原点入力論理	WORD	16#0000	0:非通電時有効 10:通電時有
8	VAR	w原点近傍論理	WORD	16#0000	0:通電時有効 20:非通電時有
9	VAR	w原点サーチ	WORD	16#0040	0:無効 40:有効
10	VAR	wリミット入力論理	WORD	16#0000	0:非通電時有効) 80:通電時有
11	VAR	wS字パターン	WORD	16#0000	0:Sin曲線1000:2次曲線2000:サ
12	VAR	w回転方向	WORD	16#0000	0:正転 100:逆転
13	VAR	w出力モード	WORD	16#0200	0:Pulse/Sign 200:CW/CCW
14	VAR	w偏差カウンタクリア時間	WORD	16#0000	0:1ms 8000:10ms
15	VAR	FPΣ位置決めユニットJOG運転	PEW_Positioning_JOG_FPSIGMA_PosUnit		
16	VAR	bJOG_正転ON	BOOL		JOG正転信号
17	VAR	bJOG_逆転ON	BOOL		JOG逆転信号
18	VAR	di初速	DINT	100	範囲:0～4,000,000
19	VAR	di目標速度	DINT	1000	範囲:1～4,000,000
20	VAR	di加減速時間	DINT	50	範囲:0～32,767
21	VAR	bBusy信号	BOOL		ビジー信号
22	VAR	エラークリア	BOOL	FALSE	

```
di経過値:= PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(iSlotNumber := iスロット番号, iAxisNumber := i軸番号);

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := sys_bIsFirstScan,
                                               iSlotNumber := iスロット番号,
                                               iAxisNumber :=i軸番号,
                                               wPositionCommand := w制御方式,
                                               wAccelerationDeceleration := w加減速方式,
                                               wDirectionOfHomeReturn := w原点復帰方向,
                                               wStartupSpeed := w起動速度,
                                               wHomeInputLogic := w原点入力論理,
                                               wNearHomeInputLogic := w原点近傍論理,
                                               wHomeSearch := w原点サーチ,
                                               wLimitInputLogic := wリミット入力論理,
                                               wS_Pattern := wS字パターン,
                                               wRotationDirection := w回転方向,
                                               wOutputMode := w出力モード,
                                               wDeviationCounterClearTime := w偏差カウンタクリア時間);

FPΣ位置決めユニットJOG運転(bForward := bJOG_正転ON,
                bReverse := bJOG_逆転ON,
                iSlotNumber := iスロット番号,
                iAxisNumber := i軸番号,
                diInitialAndFinalSpeed := di初速,
                diTargetSpeed := di目標速度,
                diAccelerationDecelerationTime := di加減速時間,
                bBusy => bBusy信号);
```

■課題 3

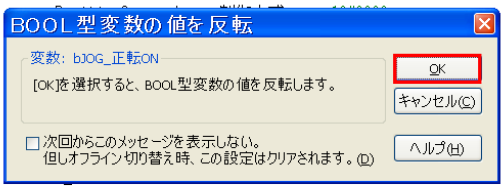
將 FPΣ定位膜組裝到 FPΣ控制器、確認課題 1、2 的程式動作看看。

用 JOG 正轉 ON 執行 JOG 正轉動作(往正方向計數)、用 JOG 逆轉 ON 執行 JOG 逆轉動作(往負方向計數)。

```
di経過値:= PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(iSlotNumber := iスロット番号, iAxisNumber := i軸番号);    0      0      1

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := %s_bIsFirstScan,
                                               iSlotNumber := iスロット番号,     0
                                               iAxisNumber :=i軸番号,     1
                                               wRotationDirection := w回転方向,     16#0000
                                               wOutputMode := w出力モード,     16#0200
                                               wDeviationCounterClearTime := w偏差カウンタクリア時間);     16#0000

FPΣ位置決めユニットJOG運転(bForward := %JOG_正転ON,
                bReverse := %JOG_逆転ON,
                iSlotNumber := iスロット番号,     0
                iAxisNumber := i軸番号,     1
                diInitialAndFinalSpeed := di初速,     100
                diTargetSpeed := di目標速度,     1000
                diAccelerationDecelerationTime := di加減速時間,     50
                bBusy => %Busy信号);
```



經過値

ダブルクリックして OK を選択します。

JOG 正轉 ON

```
di経過値:= PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(iSlotNumber := iスロット番号, iAxisNumber := i軸番号);    5923      0      1

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := %s_bIsFirstScan,
                                               iSlotNumber := iスロット番号,     0
                                               iAxisNumber :=i軸番号,     1
                                               wPositionCommand := w制御方式,     16#0000
                                               wAccelerationDeceleration := w加減速方式,     16#0000
                                               wDirectionOfHomeReturn := w原点復帰方向,     16#0000
                                               wStartupSpeed := w起動速度,     16#0000
                                               wHomeInputLogic := w原点入力論理,     16#0000
                                               wNearHomeInputLogic := w原点近傍論理,     16#0000
                                               wHomeSearch := w原点サーチ,     16#0040
                                               wLimitInputLogic := wリミット入力論理,     16#0000
                                               wS_Pattern := wS字パターン,     16#0000
                                               wRotationDirection := w回転方向,     16#0000
                                               wOutputMode := w出力モード,     16#0200
                                               wDeviationCounterClearTime := w偏差カウンタクリア時間);     16#0000

FPΣ位置決めユニットJOG運転(bForward := %JOG_正転ON,
                bReverse := %JOG_逆転ON,
                iSlotNumber := iスロット番号,     0
                iAxisNumber := i軸番号,     1
                diInitialAndFinalSpeed := di初速,     100
                diTargetSpeed := di目標速度,     1000
                diAccelerationDecelerationTime := di加減速時間,     50
                bBusy => %Busy信号);
```

往正方向計數

JOG 逆轉 ON

```
di経過値:= PEW_Positioning_GetCurrentValue_FPSIGMA_PosUnit(iSlotNumber := iスロット番号, iAxisNumber := i軸番号);    3202      0      1

PEW_Positioning_SetControlCode_FPSIGMA_PosUnit(bSetCode := %s_bIsFirstScan,
                                               iSlotNumber := iスロット番号,     0
                                               iAxisNumber :=i軸番号,     1
                                               wPositionCommand := w制御方式,     16#0000
                                               wAccelerationDeceleration := w加減速方式,     16#0000
                                               wDirectionOfHomeReturn := w原点復帰方向,     16#0000
                                               wStartupSpeed := w起動速度,     16#0000
                                               wHomeInputLogic := w原点入力論理,     16#0000
                                               wNearHomeInputLogic := w原点近傍論理,     16#0000
                                               wHomeSearch := w原点サーチ,     16#0040
                                               wLimitInputLogic := wリミット入力論理,     16#0000
                                               wS_Pattern := wS字パターン,     16#0000
                                               wRotationDirection := w回転方向,     16#0000
                                               wOutputMode := w出力モード,     16#0200
                                               wDeviationCounterClearTime := w偏差カウンタクリア時間);     16#0000

FPΣ位置決めユニットJOG運転(bForward := %JOG_正転ON,
                bReverse := %JOG_逆転ON,
                iSlotNumber := iスロット番号,     0
                iAxisNumber := i軸番号,     1
                diInitialAndFinalSpeed := di初速,     100
                diTargetSpeed := di目標速度,     1000
                diAccelerationDecelerationTime := di加減速時間,     50
                bBusy => %Busy信号);
```

往負方向計數

11-6 ST 語言的注意事項

用 ST 編集程式的注意事項。

■關於編碼的敘述

1.IF 和 FOR 等的指令、和下個文字之間必須留 1 個以上的空格。

<例>

IF switch_0 THEN → **OK**
IFswitch_0THEN → **NG**

2.對於 1 個指令或是演算式、最後一定要加上 1 個分號「;」。

3. 指令或是演算式請用半角。

4 . X , Y , R , T , C , L, DT, WR , LD, FL 等的 PLC 元件請用半角的大寫。
（小寫會被認為是變數、沒有在 POU Header 宣告則在檢查時會發生錯誤）

<例>

x0→變數
X0→PLC 的外部輸入 X0

5. PLC 元件用 2 個字元顯示時、請在前面加上「D」。

<例>

DT0 のダブルワード → DDT0
WR0 のダブルワード → DWR0

6. 關於 BOOL 型變數、TRUE 可以省略。

<例>

IF switch_0 = TRUE THEN
改為
IF switch_0 THEN
也是 OK。

■演算順位

・對於下個演算式、變數的值為 A:=1.0; B:=2.0, C:=3.0; D:=4.0。

X := A + B − C * SQRT(D) ;
演算結果為-3。

有括弧則演算順位會變、例如以下情況。

X := A + (B − C) * SQRT(D) ;
演算結果為-1。

■影響 PLC 動作的記述限制

・請勿使用"FOR "和"WHILE "在長時間的回圈(LOOP)。

PLC 在 RUN 模式則 ST 程式會從前面執行到最後、接下來執行 I/O、Reflash 和通訊、而且從頭開始執行程式。
此動作為 PLC 變成 PROG 模式為止會重複循環。
長時間使用 "FOR" 和 "WHILE" 等回圈會發生導致 I/O、Reflash 和通訊無法執行、PLC 的看門狗會有逾時的情況發生。

<例>

```
(i 和 j 為 INT 型的變數)
FOR i:=0 TO 32767 DO
    j :=j+1;
END_FOR;
```

以上程式的 FOR 回圈將時間變長後、會引起 PLC 的看門狗逾時。
(FP0 和 Σ 的場合為 ERR.的 LED 亮燈、FP2SH 等為 ALARM 的 LED 亮燈)

・WHILE 等的回號請不要用在外部輸出入(X ,Y)的控制條件上。

(即使使用也不會有編譯錯誤)
PLC 到達 TASK 的最後為止、不會執行 I/O Reflash、此語法會變成死回路
PLC 的看門狗有可能會逾時。

<例>

```
WHILE X0=TRUE DO
    DT0:=DT1 ;
END_WHILE ;
```

如上面的記述、X0 在 ON 時 DO 以後的程式循環重複、並不執行 I/O Reflash、
程式到最後為止不會執行。
所以實際上 X0 就算 OFF 也不會反映到程式、看門狗會逾時導致 PLC 發生 ERROR。

第12章

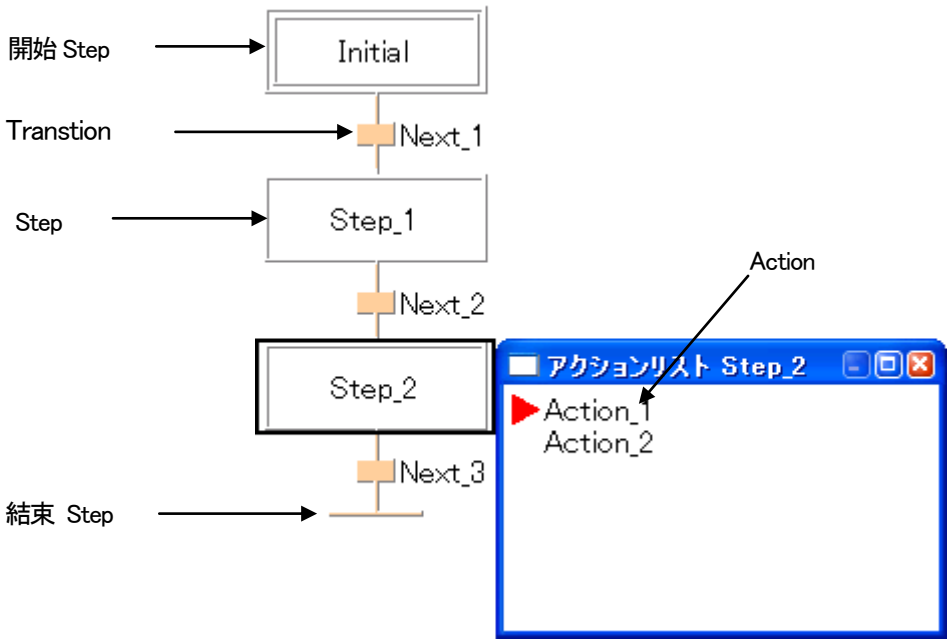
用 Sequential function chart(SFC)製作程式

12-1 概要

Sequential function chart(SFC)可以將複雜的程式，分成多個程序、並用容易了解方式記述。

12-1-1 Sequential function chart 構成的要素

Sequential function chart(SFC)是為一串的 STEP 流程、使用 Step・Transtion(遷移)來表示。

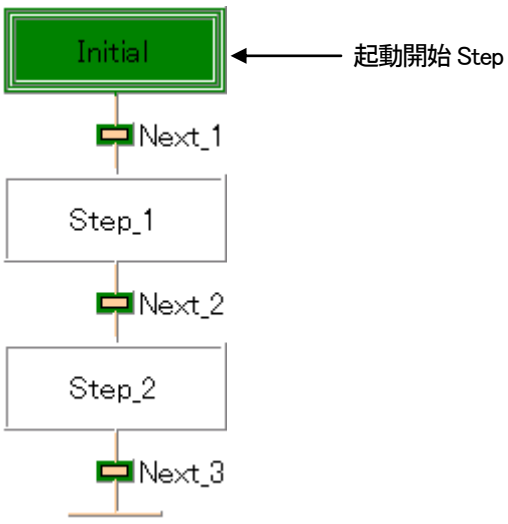


利用上圖來說明 SFC 的流程。

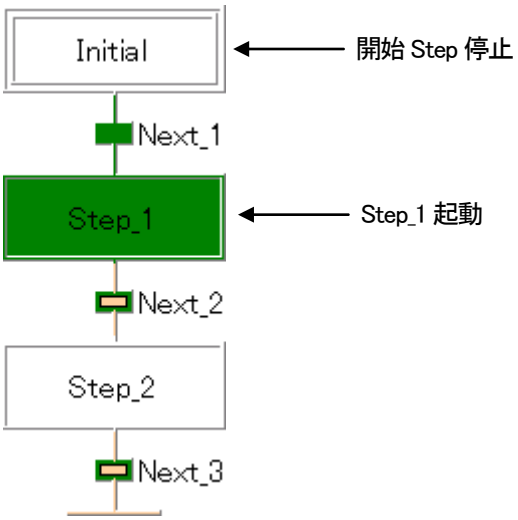
■STEP

所謂 STEP 就是如上圖的長方形方格的[Step_1], [Step_2]之部分的 Task。

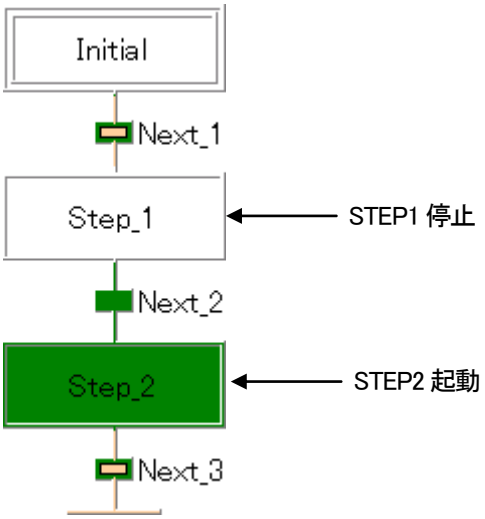
初期 Step(Initial)作為起動開始 Step。



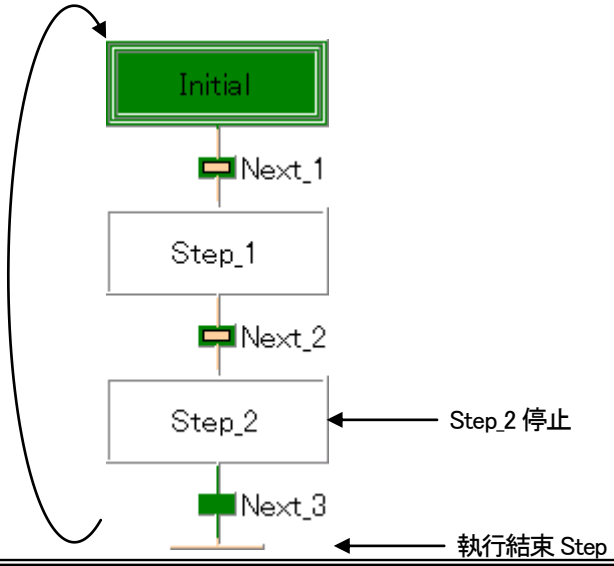
Transtion 的"Next_1"變成 ON(TRUE)時、Step_1 就會被起動、開始 Step 會停止。



接下來的 Transtion 的"Next_2"變成 ON(TRUE)時、Step_2 會被起動、Step_1 就會停止。

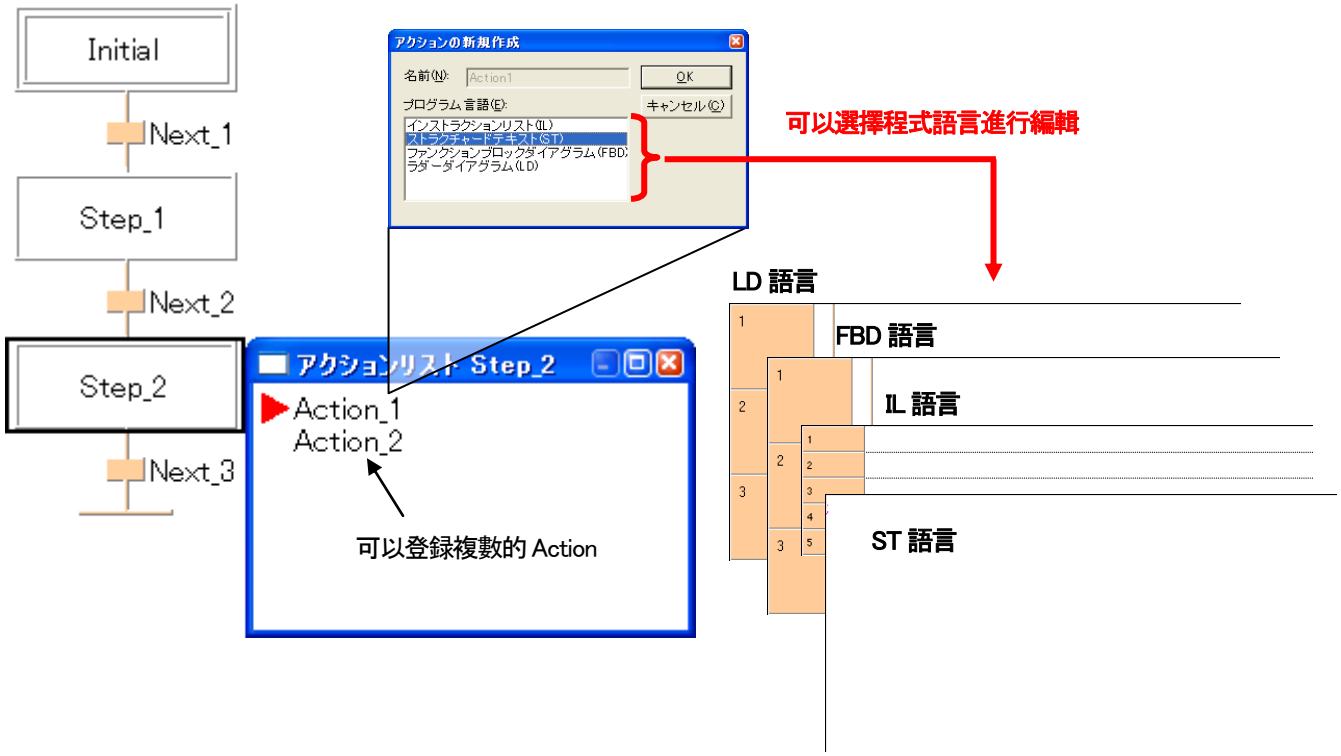


接下來、Transtion 的"Next_3"會變 ON、執行到結束 Step 為止、Step_2 就會停止、
再次從開始 Step 進行一連串的順序處理。



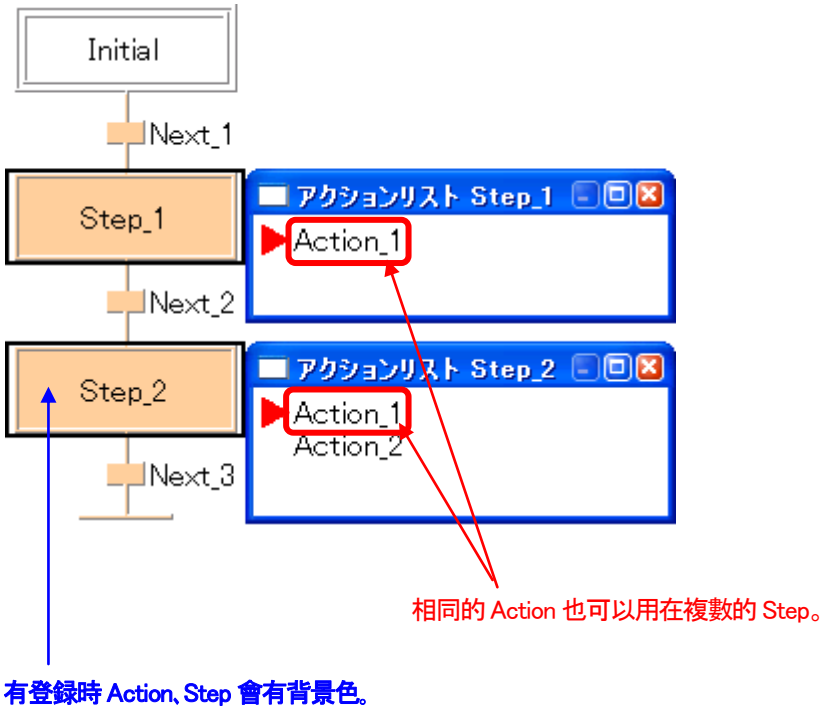
■Action

所謂 Action 就是從某個 Step 變為 Active(起動状態)時、所執行的 logic。
Action 可以登録到複数個的 Step、可以用 IL, FBD, LD, ST 語言編集。
而且毎個 Step 可以登録複数的 Action(Step 内的程式)。

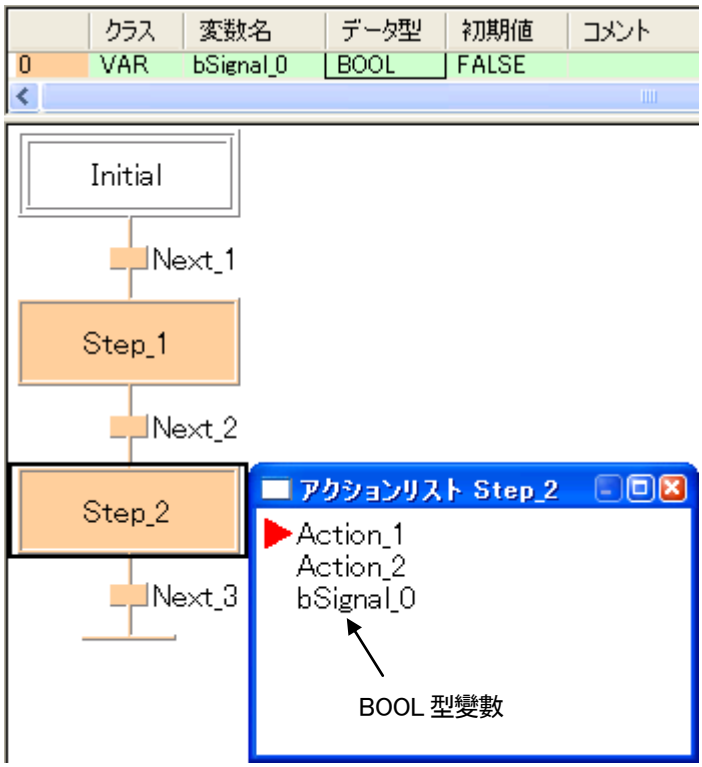


Step 為 Action 時、一定會執行 Step 内一連串的 Action。
沒有登録 Action 的 Step、緊接著的 Transtion 條件成立為止前都會為等待状態。

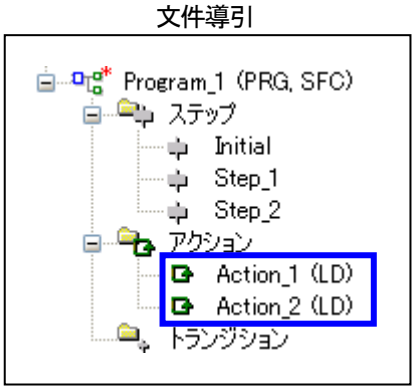
相同的 Action 也可以用在複数的 Step。



BOOL 型變數也可以作為 Action 來使用。



製作的 Action 會登録到 Project 導引中的 Action 子路徑



●注意

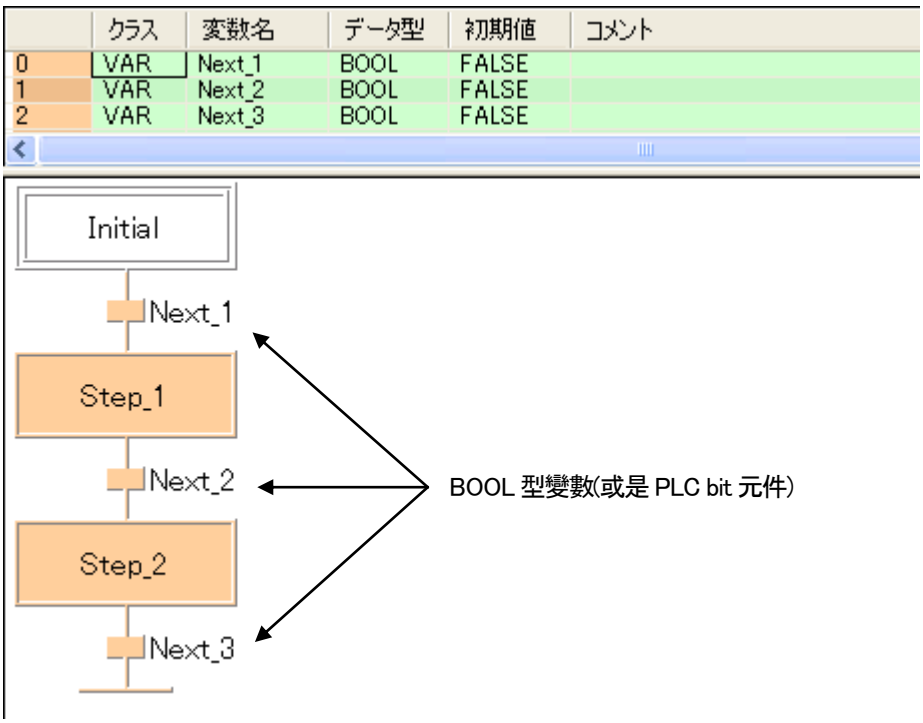
以 Action 所登録的程式不可使用 Label。

■Transtion

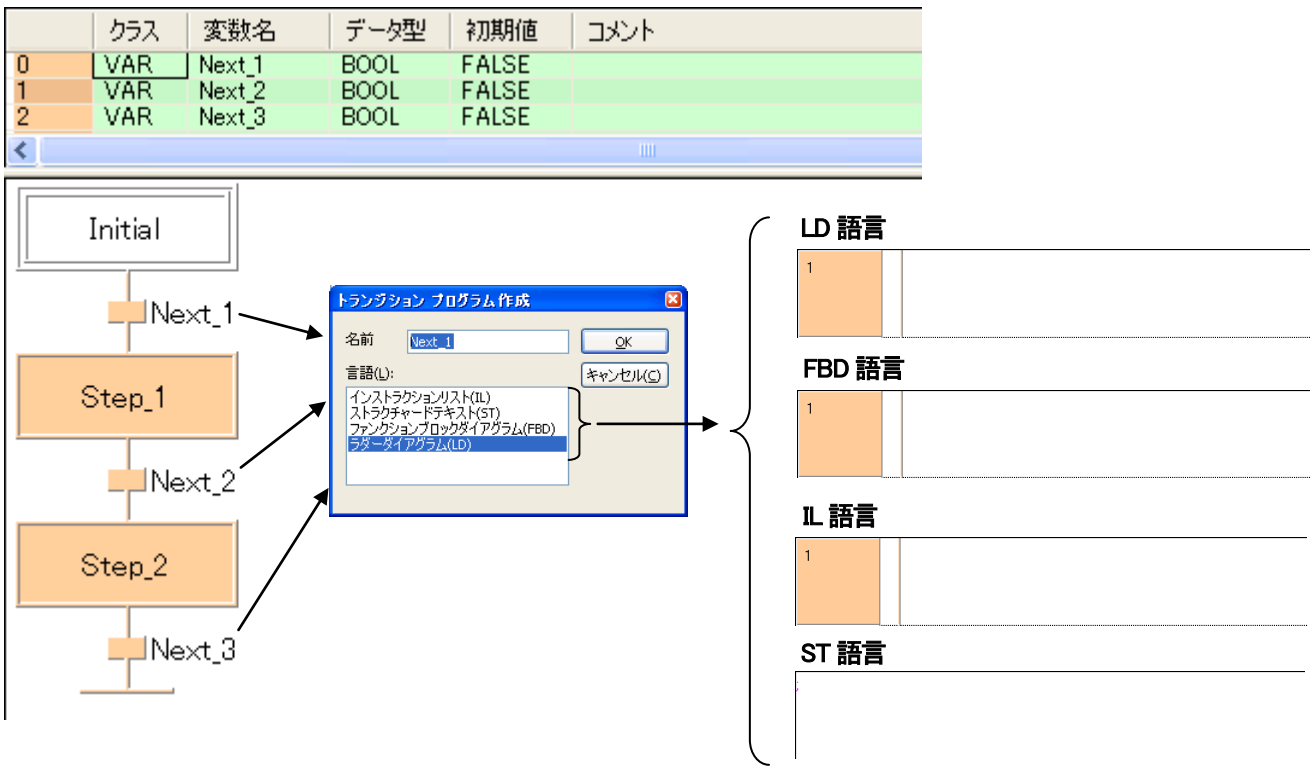
所謂 Transtion 就是有條件的 Jump。
Transtion 的條件成立時、就起動下個 Step。

BOOL 型變數(或是 PLC bit 元件)和用 IL, FBD, LD, ST 語言所作成的程式會配置到 Transtion。

配置 BOOL 型變數(或是 PLC bit 元件)時



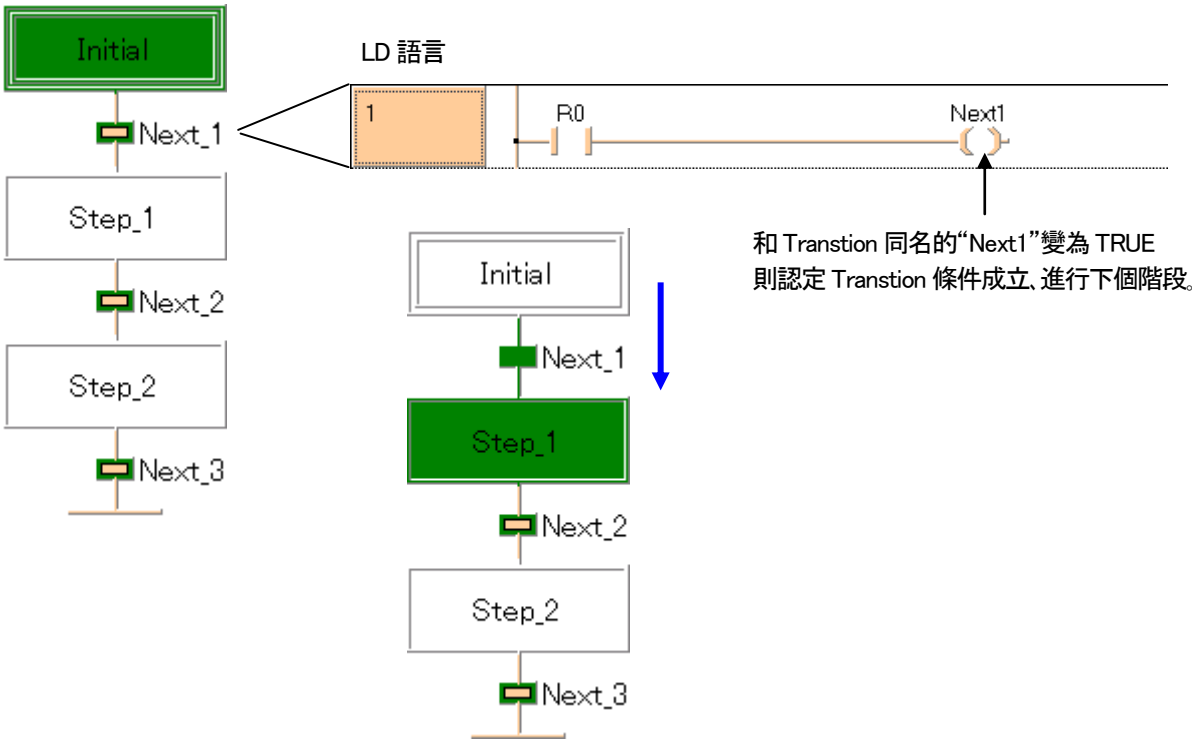
配置程式時



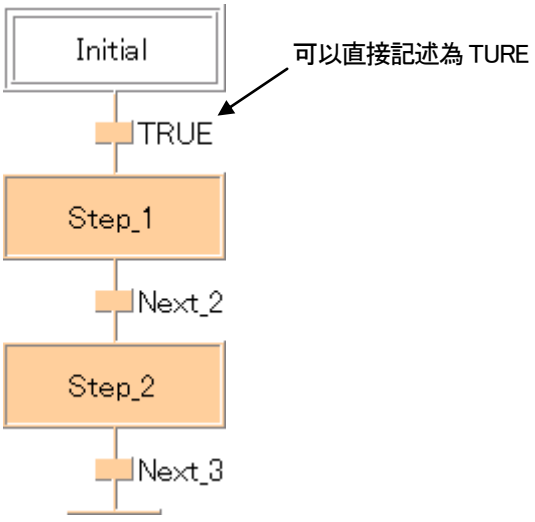
配置 BOOL 型變數或是 PLC bit 元件的時候
所配置的變數值變為 TRUE 則會認定 Transtion 為成立。

配置程式的時候
和 Transtion 同名的變數值變成 TRUE 則認定 Transtion 條件成立。

LD 語言的範例



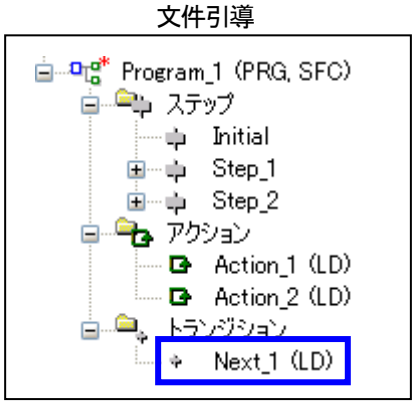
並且可以直接記述為「TRUE」。
這時候為前面的 Step 執行後、自動的起動下個 Step。



上圖的場合、"Initial"Step 起動後
"Step_1"會自動起動。

將程式配置到 Transtion 時,會登録到文件引導的 Transtion 子路徑。

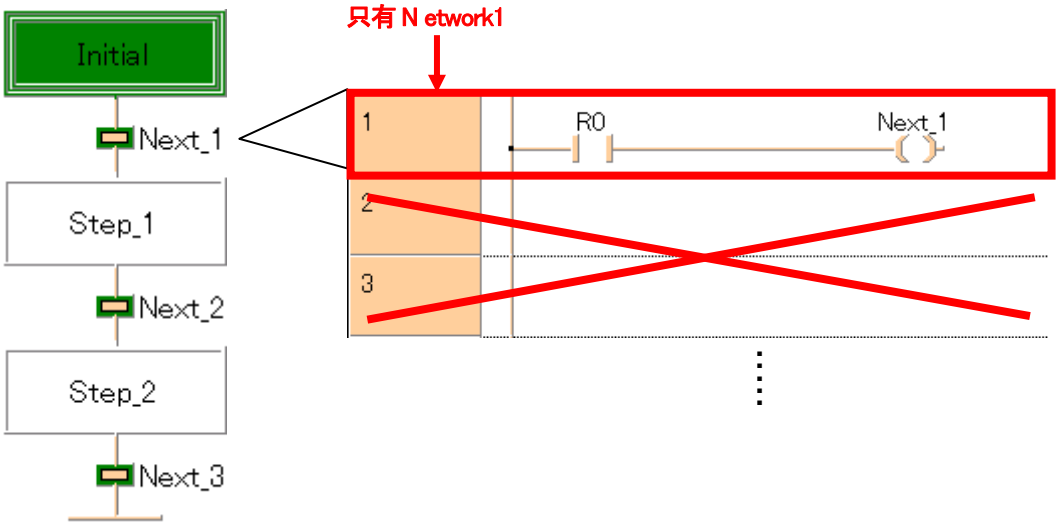
文件引導



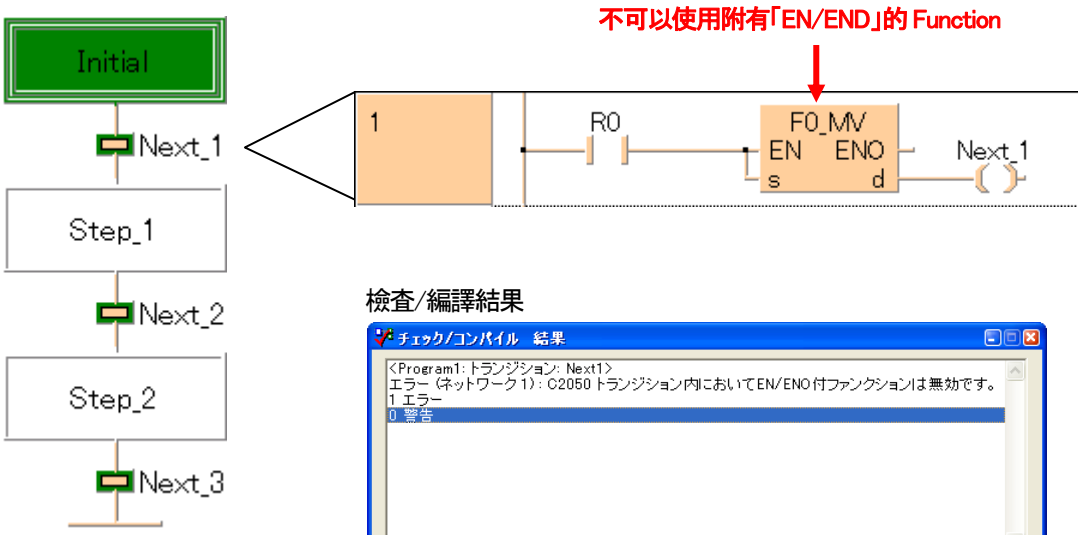
●注意

配置程式到 Transtion 時、有以下制限。

①在 Transtion 區可以作成的程式只有 Network1。



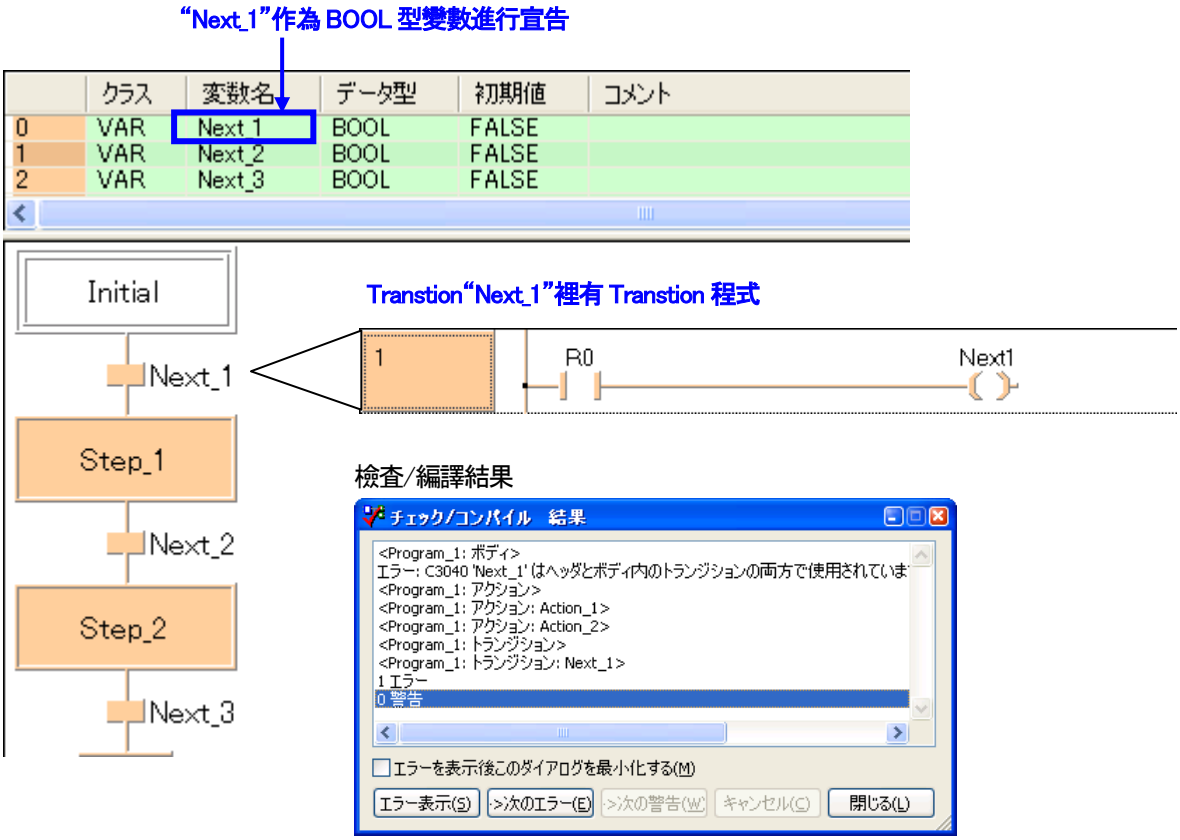
②不可以使用附有「EN/END」的 Function。



檢查/編譯結果

在檢查/編譯時、
發生「在 Transtion 內 EN/ENO 的 Function 為無效」的錯誤。

③同時和 BOOL 型變數(或是 PLC bit 元件)不可以配置。



在檢查/編譯時、
「"Next_1"在 Header 和程式區內的 Transtion 兩邊都正在使用。」的錯誤發生。

■Step Flag

只有特定的 Step 起動時、把 ON 的 Flag 稱為 Step Flag。
Step Flag 會用「Step 名稱.X」顯示、在程式中可以使用。
(請參考「12-3 Step Flag 設定的時間變化」)

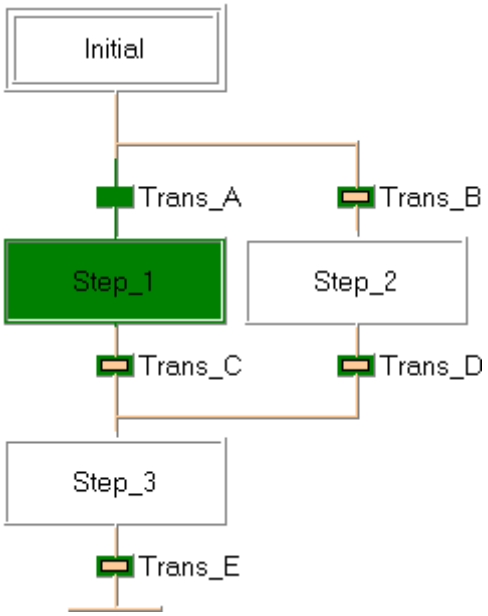
■MacroStep

複數的 Step 可以整理為 1 個 MacroStep、期流程也可以簡單明瞭。
MacroStep 是附上 2 條橫線的符號。
(請參考「12-4-5 MacroStep」)

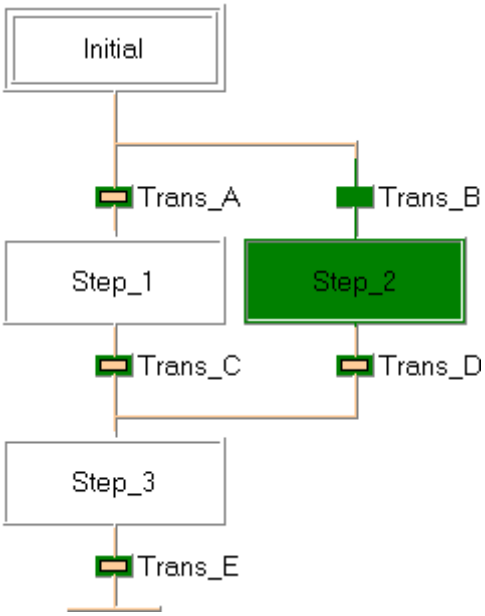
12-1-2 分岐和並列分岐

■分岐

分岐的記号為一條横線です。
在 2 個 Transtion 中、任一個條件(Trans_A 或是 Trans_B)成立時、就執行成立的一方。



上圖的状態為 Trans_B 即使變 ON(TRUE)
、Step_2 也不會起動。



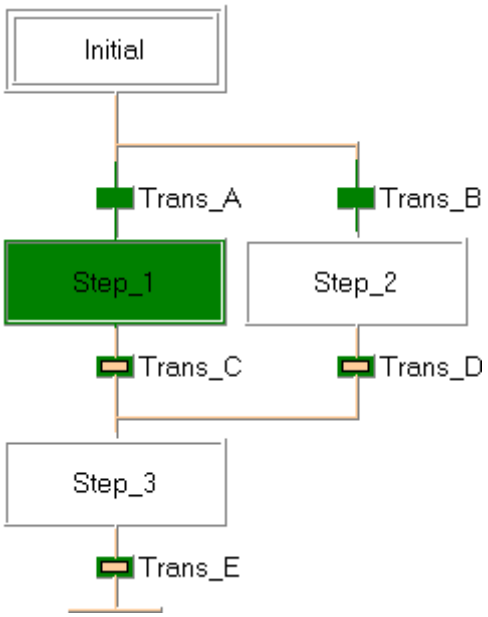
上圖的状態為 Trans_A 即使變 ON(TRUE)、
Step_1 也不會起動。

●注意

兩方 Transtion 條件同時變 ON(TRUE)時、以左邊的 Step 為優先執行。

<例>

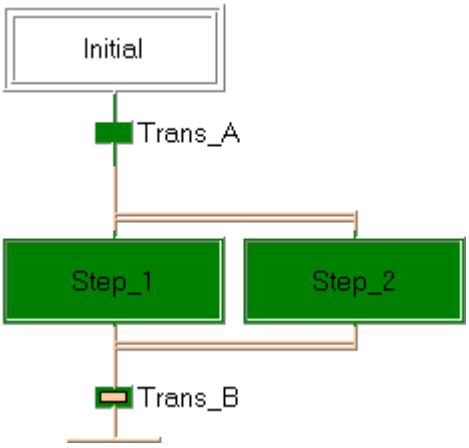
Trans_A 和 Trans_B 同時變 ON(TRUE)時、只有 Trans_A 的 Step(Step_1)會執行。



■並列分岐

並列分岐的記号是二條横線。
並列分岐之直的 Transtion(Trans_A)條件成立時、同時執行並列處理行 Step_1、Step_2。

<例>



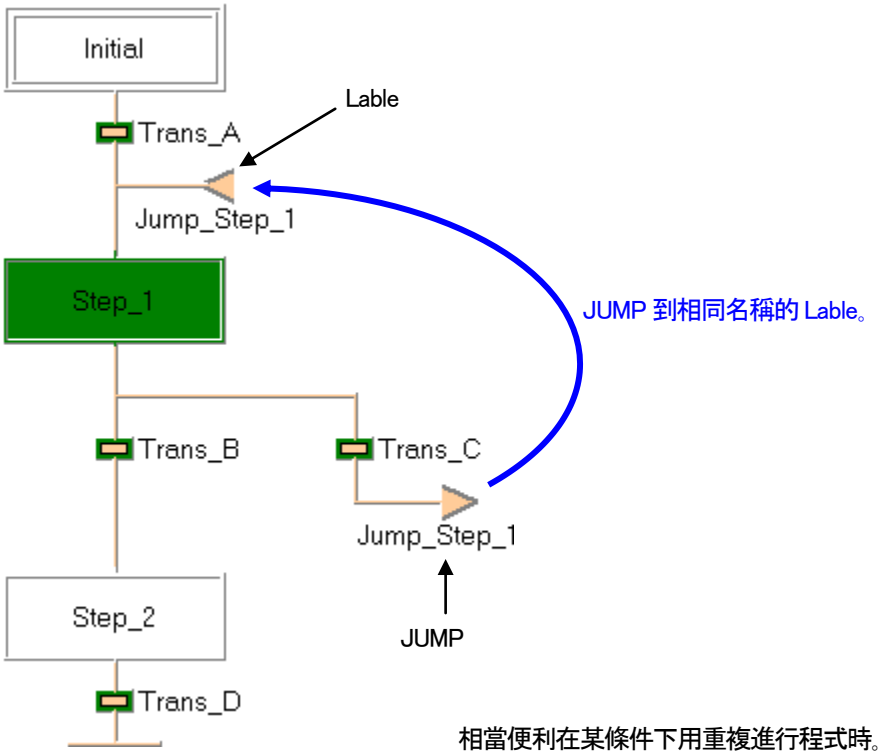
通過並列分岐、分岐過的全部處理再經過並列結合到 Transtion(Trans_B)再次結合。

●結合時的注意

並列結合是將之前全部執行中的 Step(Step_1, Step_2)、
並且 Transition(Trans_B)成立時、會結合成 1 個 Step。

■JUMP 和 LABLE

Transition(Trans_C)條件成立、則再次執行 Step_1。



相當便利在某條件下用重複進行程式時。

12-2 編集

在 SFC Edit 使用工具列上的小圖示、可以編輯要素(element)和 Object 的「插入」,「顯示」,「打開」等。
並且可以分割編集視窗和程式、從使用者所作好的程式可以將各種的 Object 同時在畫面上顯示。

12-2-1 工具列上的小圖示和按鍵

打開 SFC Body、則以下的小圖示會顯示在工具列上。

■插入新的 Object 的小圖示

「編集」選單 →「插入」也能選擇同樣功能。

-  : 將附有 Transition 的 Step 插入到現在位置的前面
-  : 將 Step 插入到現在位置的前面
-  : 將 Transition 插入到現在位置的前面
-  : 將往左分岐／並列分岐插入到現在位置的前面
-  : 將往右分岐／並列分岐插入到現在位置的前面
-  : 將左邊開始的結合／並列結合插入到現在位置的前面
-  : 將右邊開始的結合／並列結合插入到現在位置的前面
-  : 將 Label 插入到 Step 的前面
-  : 插入 JUMP 到 Transition 的後面

■顯示 Object 或是打開小圖示

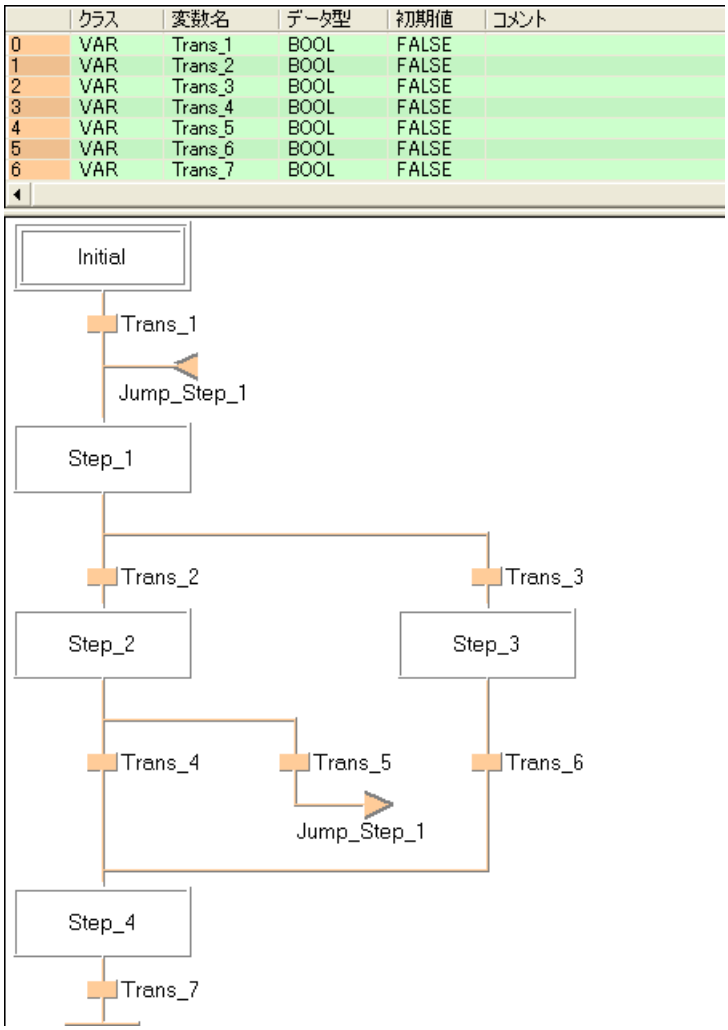
跟工具選單的某個項目相同功能。

-  : 打開變數的選擇對話框（只有在 Transition 編集状態時）
-  : 打開 Mirco 構成 List
-  : 編輯 Step 的 Command
-  : 打開所選擇的 Object
-  : 關閉所打開的 Object
-  : 檢索下個相同的 Error

12-2-2 SFC 的畫圖

在此介紹關於 SFC 的畫圖方法。

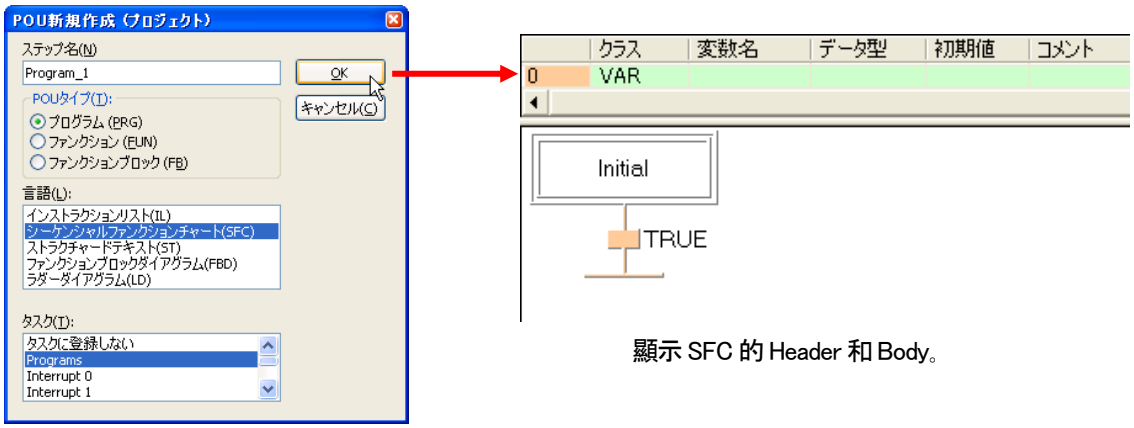
畫出下圖的 SFC、學習操作方法。



■操作步驟

1. 進行 POU 的新作成。

輸入檔案名稱後、在程式語言中選擇「Sequential function chart(SFC)」點擊 "OK"。



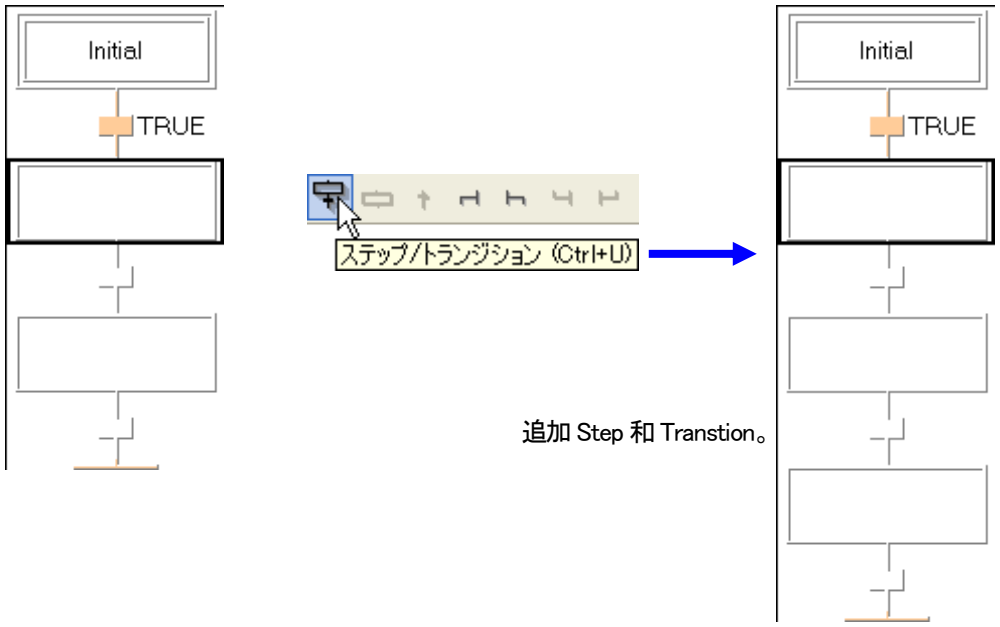
顯示 SFC 的 Header 和 Body。

2. 畫出 SFC 的流程。

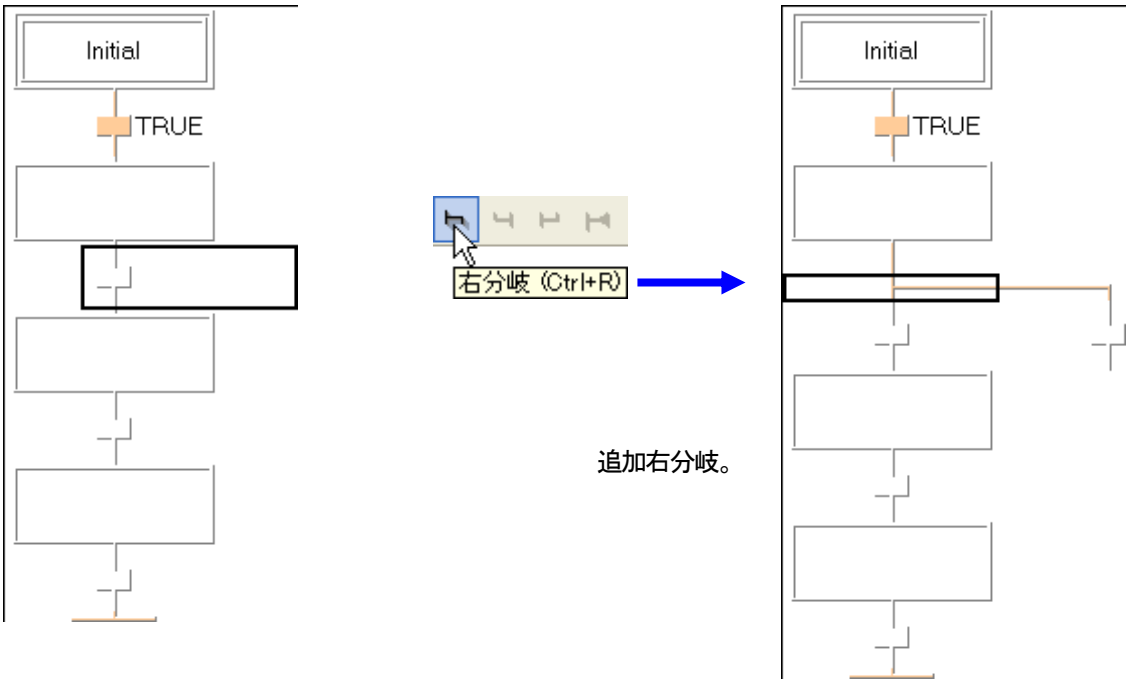
2-1 將鼠標移往到最後的 Step、點擊「Step/Transition」小圖示。

2-2 將鼠標移動到下圖的 Step 位置、點擊「Step/Transtion」小圖示。

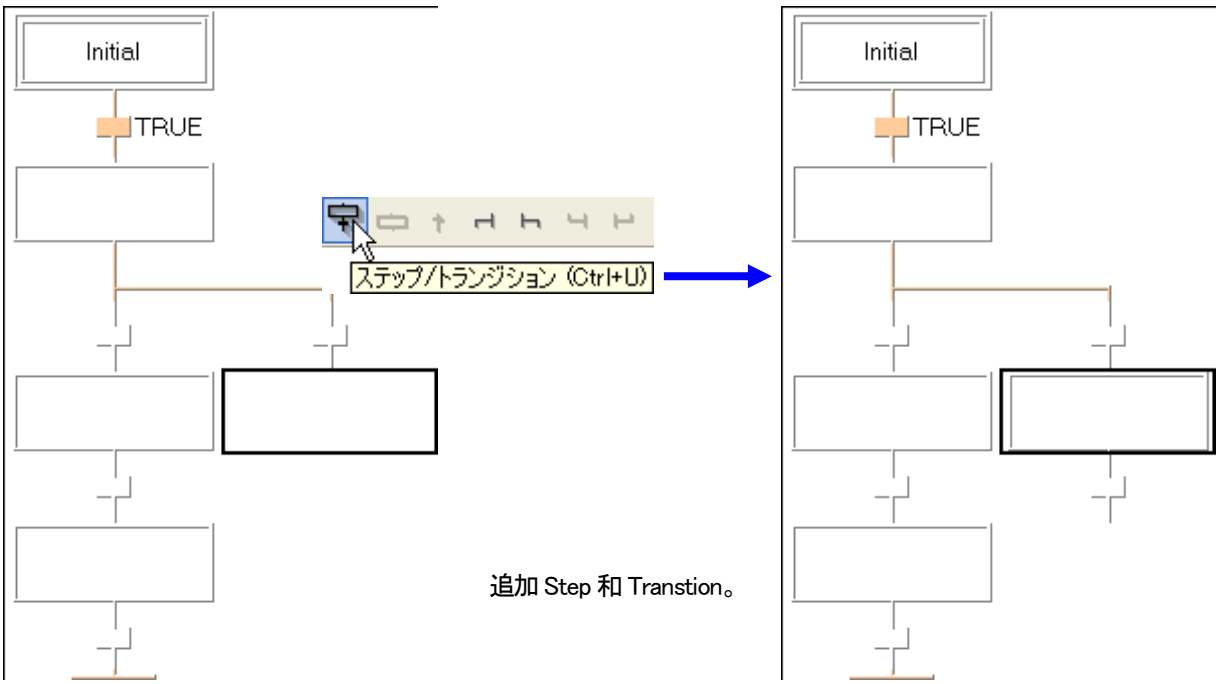
2-3 將鼠標移動到下圖的 Step 位置、點擊「Step/Transtion」小圖示。



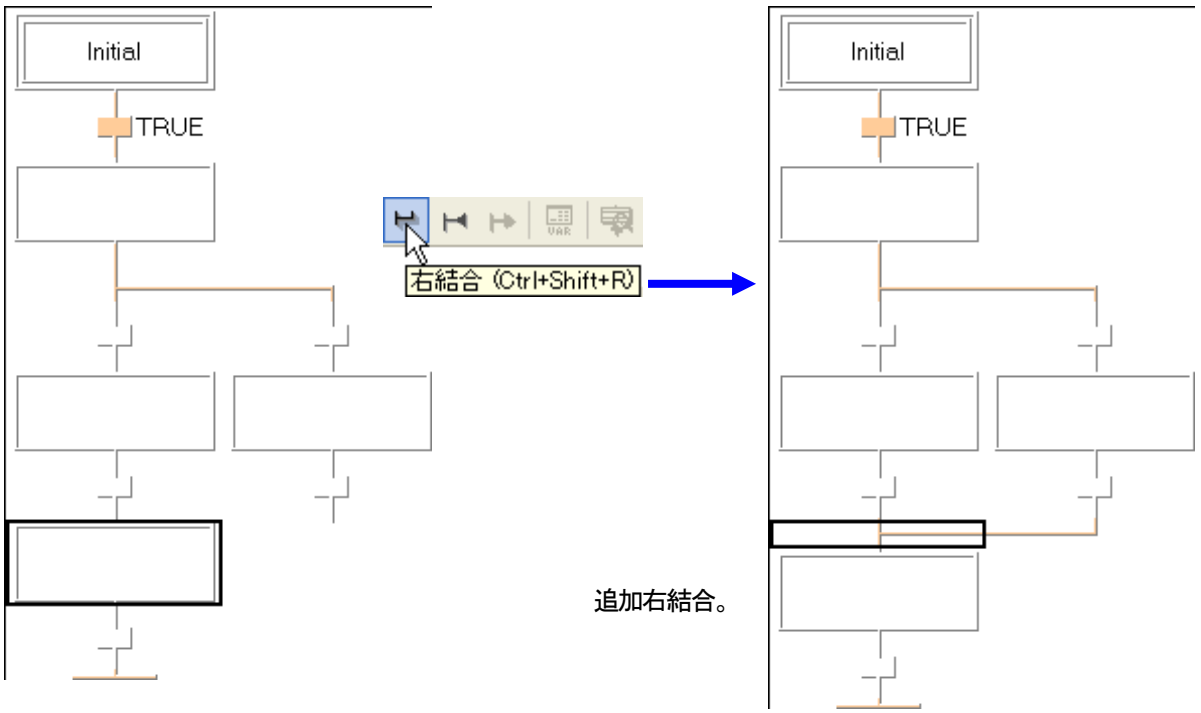
2-4 將鼠標移往到最後的 Step、點擊「右分岐」 小圖示。



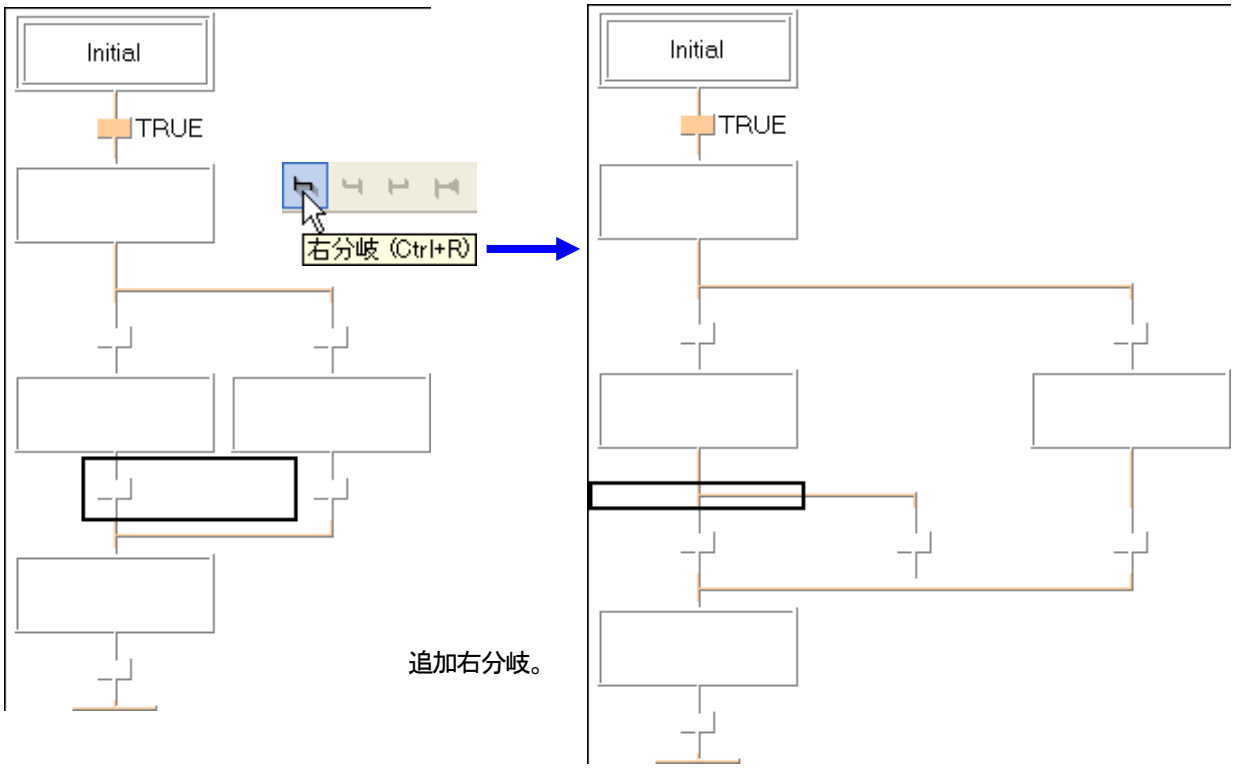
2-5 將鼠標移動到下圖的 Step 位置、點擊「Step/Transtion」 小圖示。



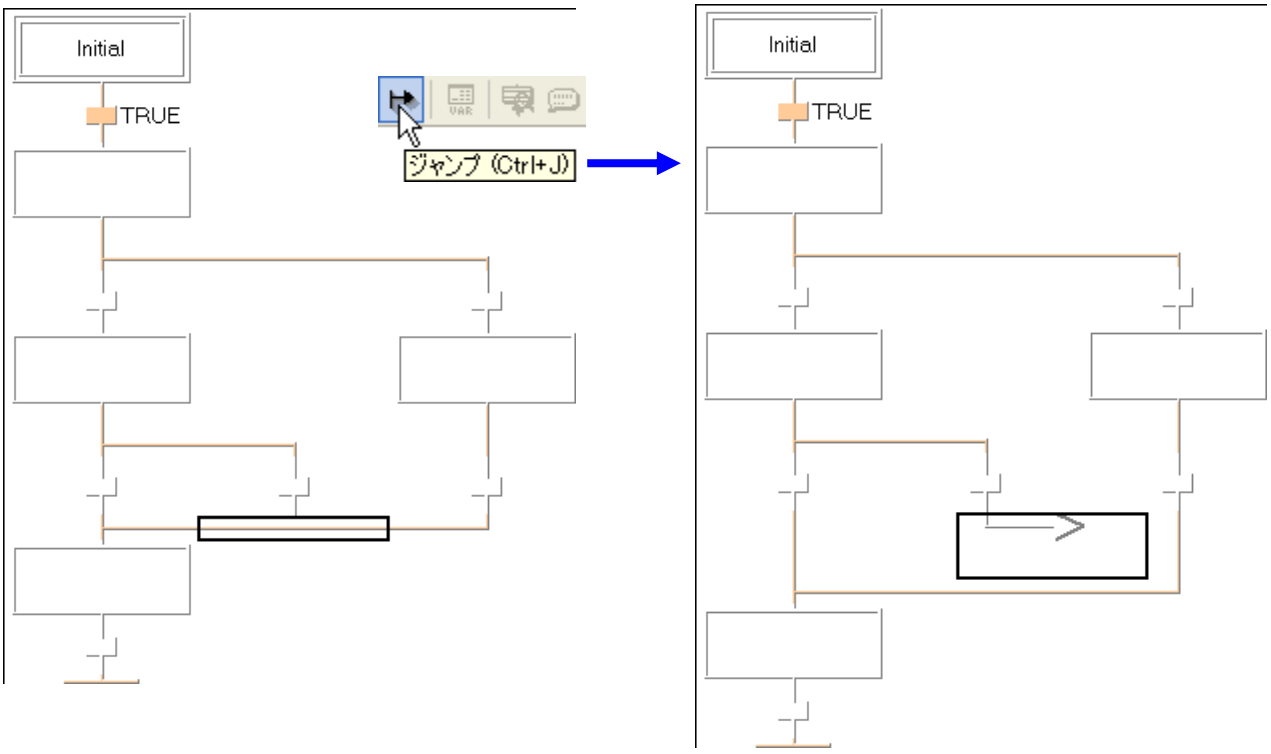
2-6 將鼠標移動到下圖的 Step 位置、點擊「右結合」 小圖示。



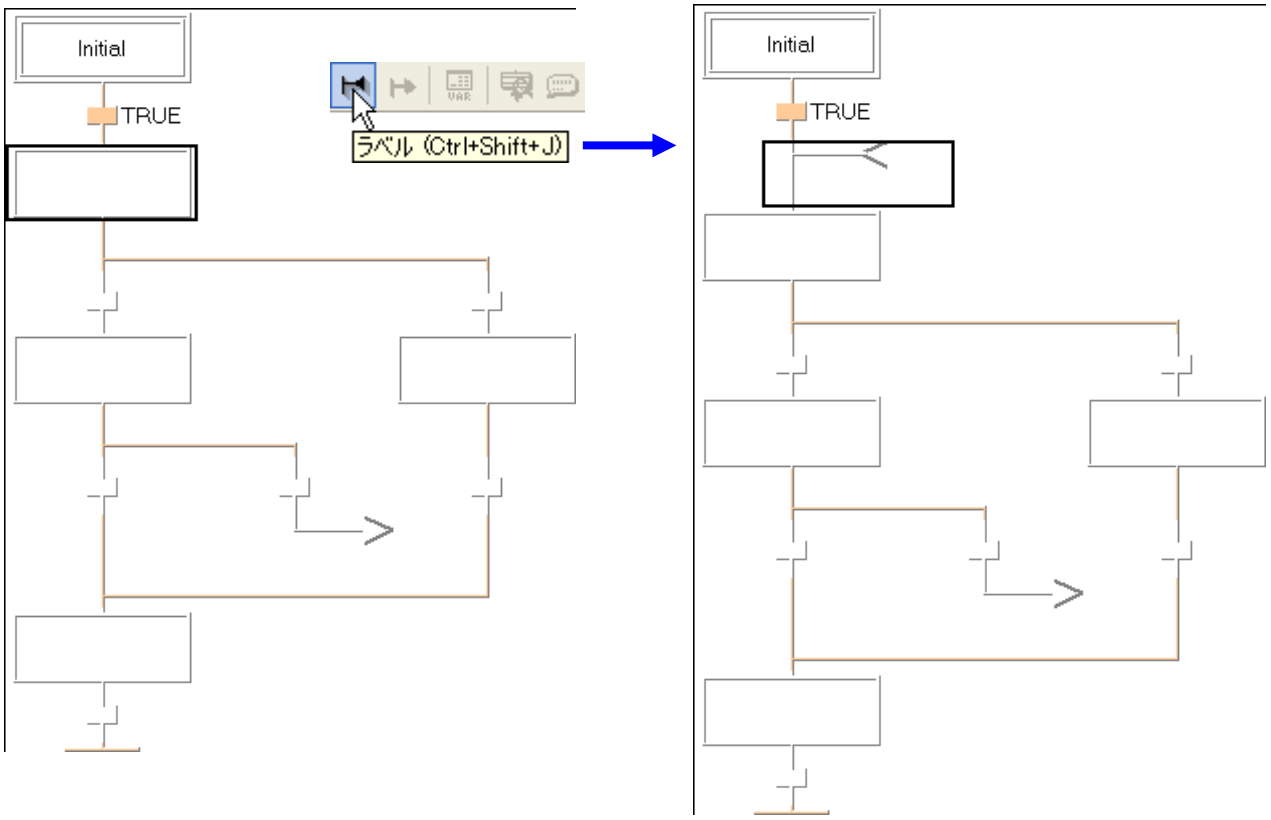
2-7 將鼠標移動到下圖的 Step 位置、點擊「右分岐」 小圖示。



2-8 將鼠標移動到下圖的 Step 位置、點擊「JUMP」 小圖示。



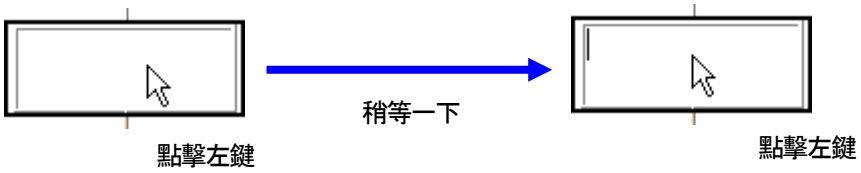
2-9 將鼠標移動到下圖的 Step 位置、點擊「Label」 小圖示。



到此流程製作完成。

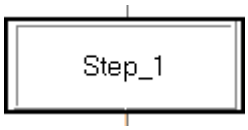
3. STEP 附上名稱。

將對象的 Step 用滑鼠左鍵用鼠標對準後、再次點擊左鍵。
請注意並非點擊兩次。



催促輸入名稱的游標會在 STEP 上面出現。

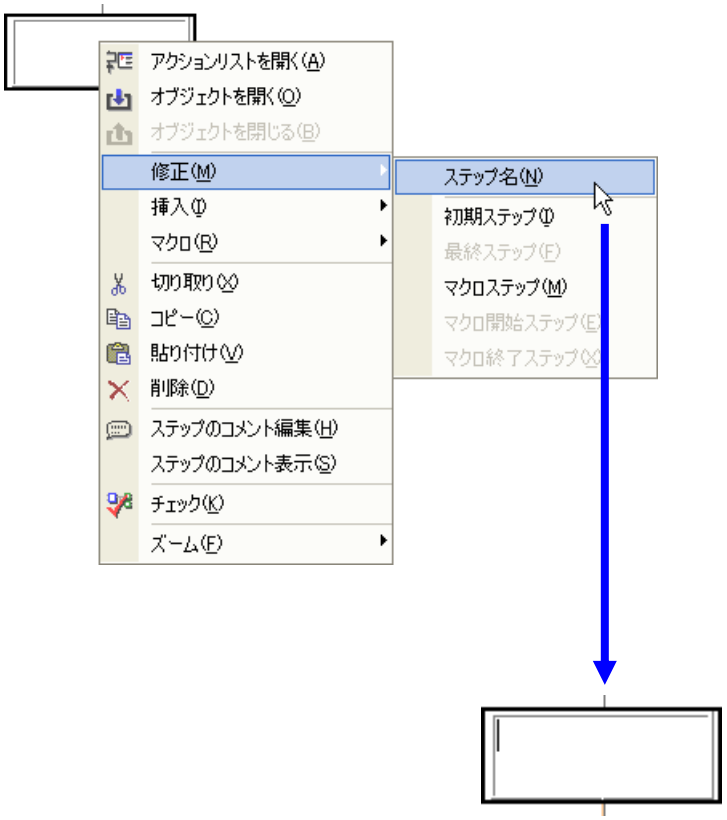
在此輸入 STEP1。(輸入後、用 Enter 鍵確定)



●備考

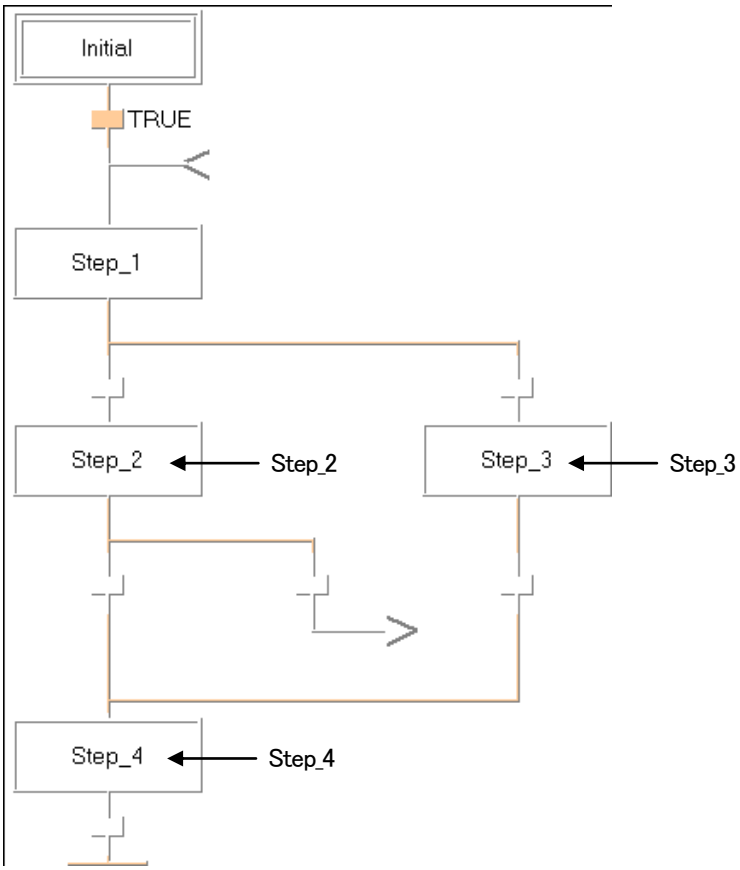
下面的方法也是相同。

將游標對準對象 STEP、點擊右鍵從選單選擇「修正」→「STEP 名」。



變成催促輸入 STEP 名稱的狀態。

用相同方法、對所有的 STEP 加上名稱。



4.分配 BOOL 型變數到 Transtion。

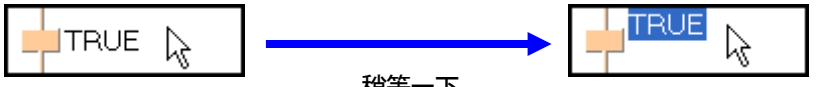
在 Transtion 之中、雖然可以分配 BOOL 型變數(或是 PLC Bit 元件)或是 Transtion 程式、
在此為分配 BOOL 型變數。

宣告 BOOL 型變數。

	クラス	変数名	データ型	初期値	コメント
0	VAR	Trans_1	BOOL	FALSE	
1	VAR	Trans_2	BOOL	FALSE	
2	VAR	Trans_3	BOOL	FALSE	
3	VAR	Trans_4	BOOL	FALSE	
4	VAR	Trans_5	BOOL	FALSE	
5	VAR	Trans_6	BOOL	FALSE	
6	VAR	Trans_7	BOOL	FALSE	

宣告變數名稱 Trans_1～Trans_7 共 7 個 BOOL 型變數。

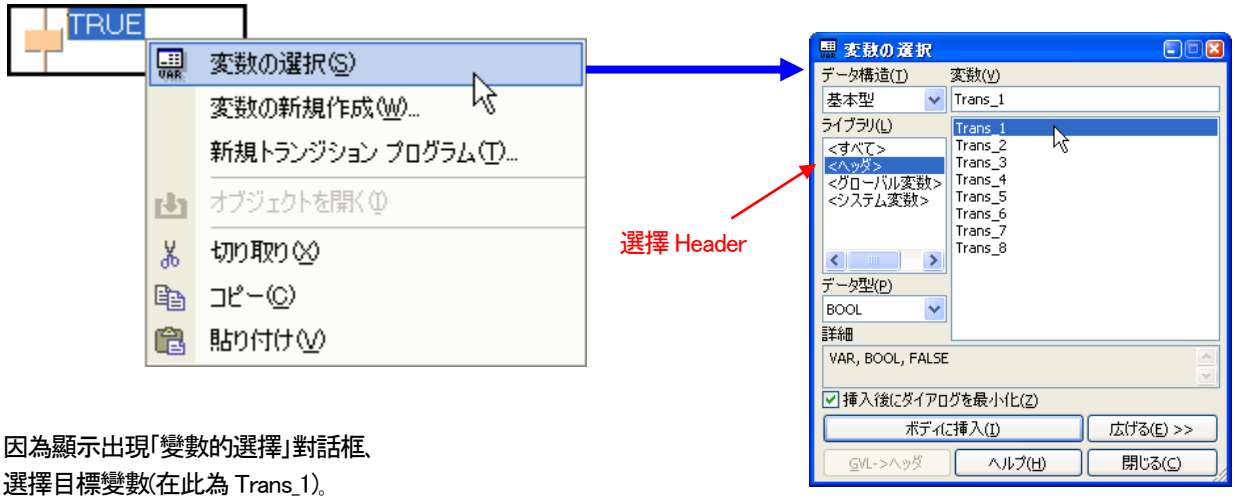
在對象的 Transtion 用滑鼠點擊左鍵對準鼠標、再點擊左鍵。
請注意並非是點擊兩次。



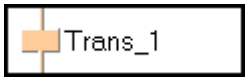
點擊左鍵　稍等一下　點擊左鍵

催促輸入的游標會上 STEP 上面出現。

點擊右鍵從選單選擇「變數的選擇」。



因為顯示出現「變數的選擇」對話框、
選擇目標變數(在此為 Trans_1)。

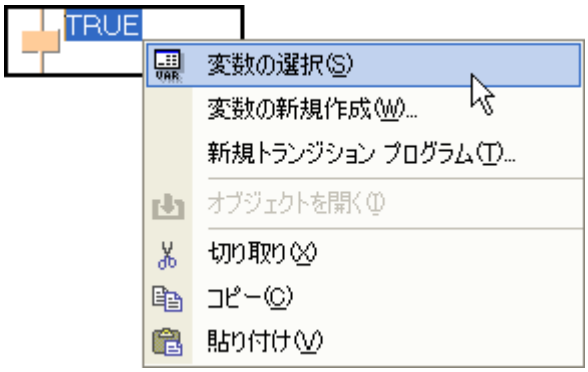


上面已經預先輸入為"TRUE"的状態、分配新的 BOOL 型變數時也是同樣的。

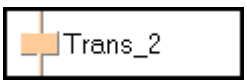
點擊左鍵　稍等一下　點擊左鍵

催促輸入的游標會出現到 STEP 上。

點擊右鍵從選單選擇「變數的選擇」。

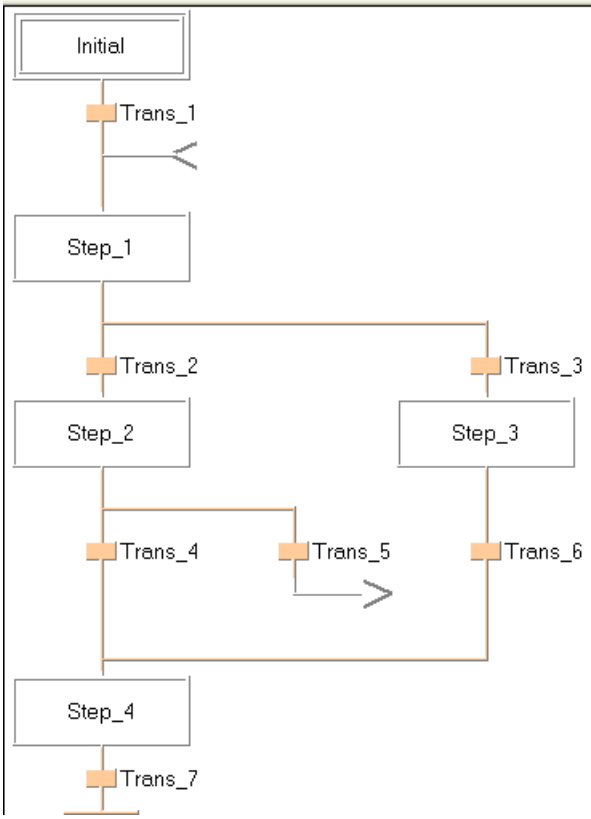


選擇目標的變數。



用同樣方法分配 BOOL 型變數到 Transtion。

	クラス	変数名	データ型	初期値	コメント
0	VAR	Trans_1	BOOL	FALSE	
1	VAR	Trans_2	BOOL	FALSE	
2	VAR	Trans_3	BOOL	FALSE	
3	VAR	Trans_4	BOOL	FALSE	
4	VAR	Trans_5	BOOL	FALSE	
5	VAR	Trans_6	BOOL	FALSE	
6	VAR	Trans_7	BOOL	FALSE	



5. 到 JUMP 加上名稱。

到對象的 JUMP 位置點擊兩次。

點擊兩次

催促輸入名稱的游標會在 JUMP 上面出現。

在此輸入 Jump_Step_1。(輸入後、按下 Enter 按鍵確定)

6.到 Lable 加上名稱。

在對象的 Lable 位置點擊兩次。

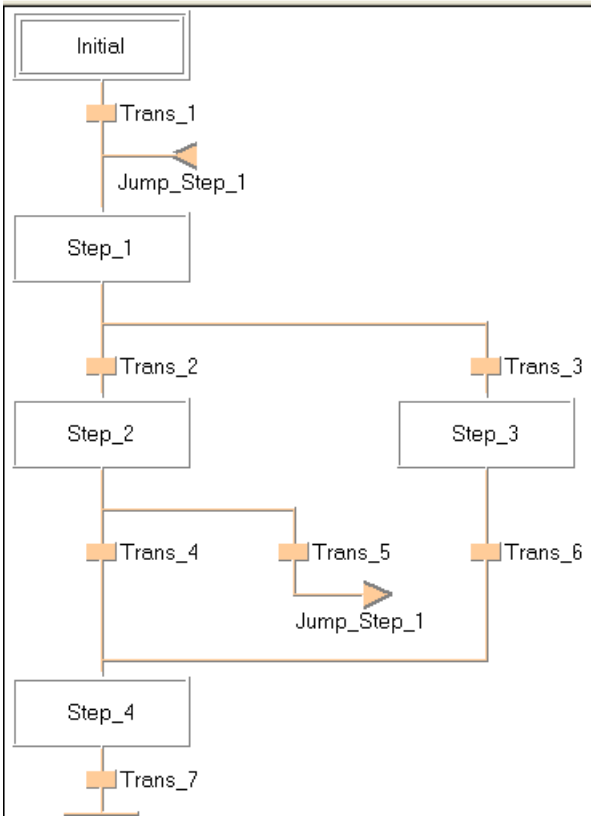
點擊兩次

催促輸入名稱的游標會在 JUMP 上面出現。

在此輸入和 JUMP 的相同名稱 Jump_Step_1。(輸入後、按下 Enter 按鍵確定)

SFC 的流程到此已經完成。

	クラス	変数名	データ型	初期値	コメント
0	VAR	Trans_1	BOOL	FALSE	
1	VAR	Trans_2	BOOL	FALSE	
2	VAR	Trans_3	BOOL	FALSE	
3	VAR	Trans_4	BOOL	FALSE	
4	VAR	Trans_5	BOOL	FALSE	
5	VAR	Trans_6	BOOL	FALSE	
6	VAR	Trans_7	BOOL	FALSE	



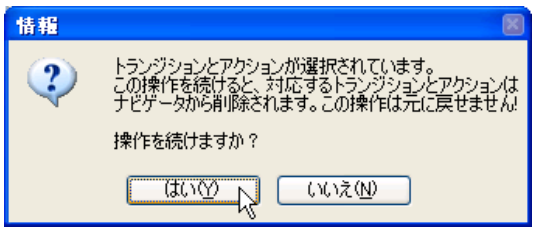
SFC 的畫圖沒有順序。

以上的步驟只是一個例子、沒有先後順序。

12-2-3 SFC 流程的削除和修正

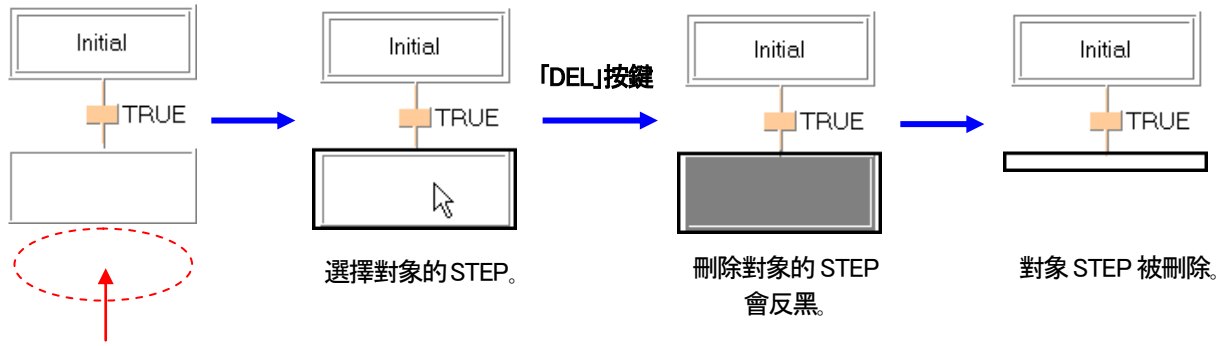
將游標移動到想要刪除的 STEP 及 Trastion、按下「DEL」按鍵即可刪除。
不過 STEP 下面有 Transtion 或是 Transtion 下面有 STEP 的話，會被一併刪除，請務必注意。

還有當使用「DEL」按鍵刪除時、雖然會顯示如下圖的對話框、
請繼續選擇「YES」。

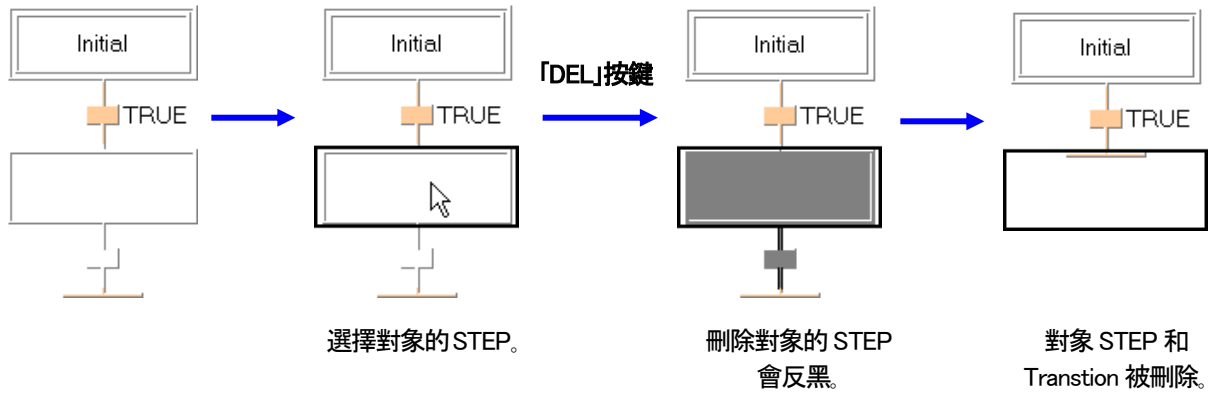


■STEP 的刪除

①STEP 下面的 Transtion 不存在時

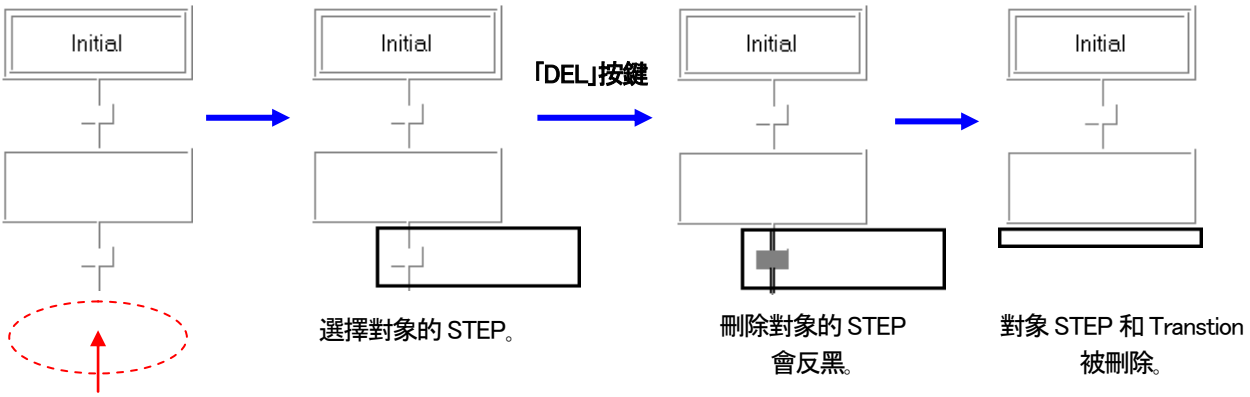


②STEP 下面的 Transtion 存在時



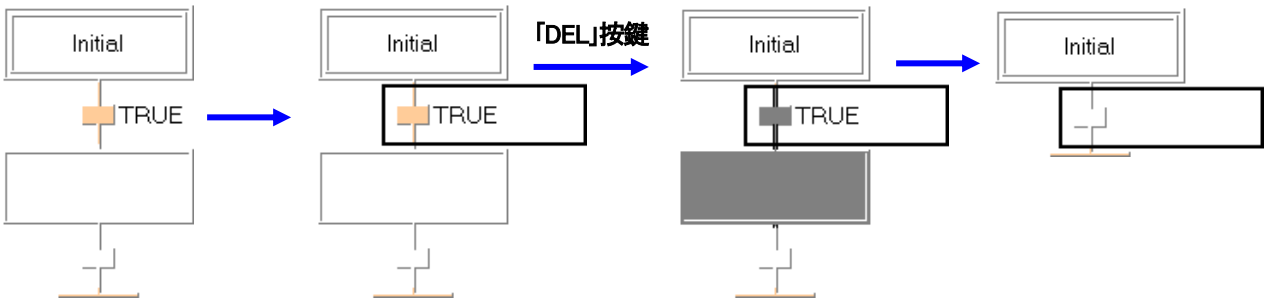
■Transtion 的刪除

①STEP 下面的 Transtion 不存在時

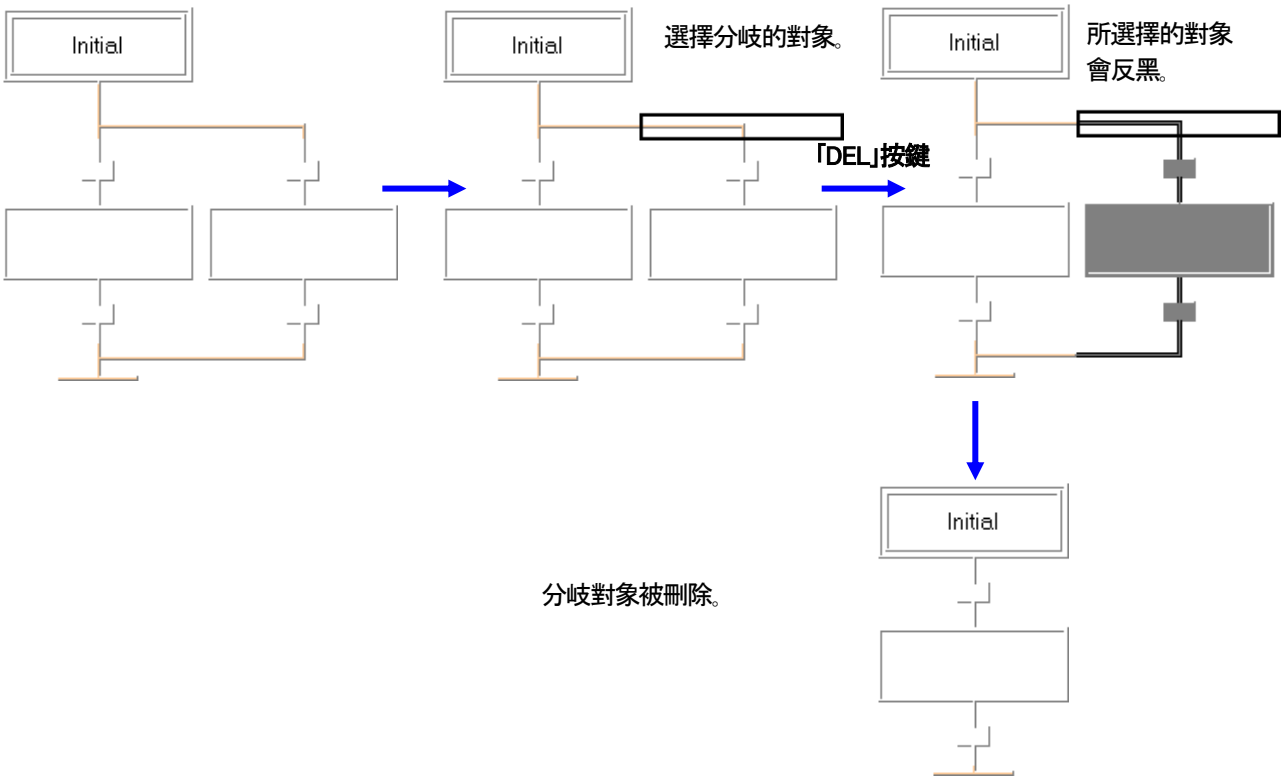


位於 STEP 的下面沒有 Transtion 的時候

②STEP 下面的 Transtion 存在時



■分岐全部刪除

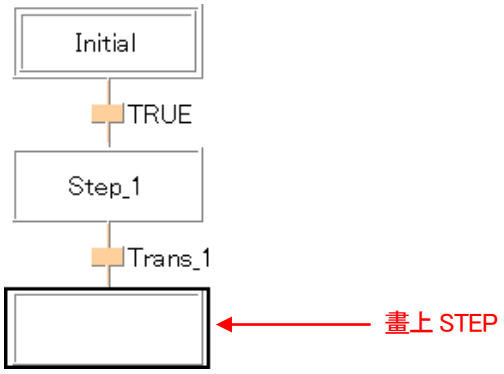


■STEP 的修正

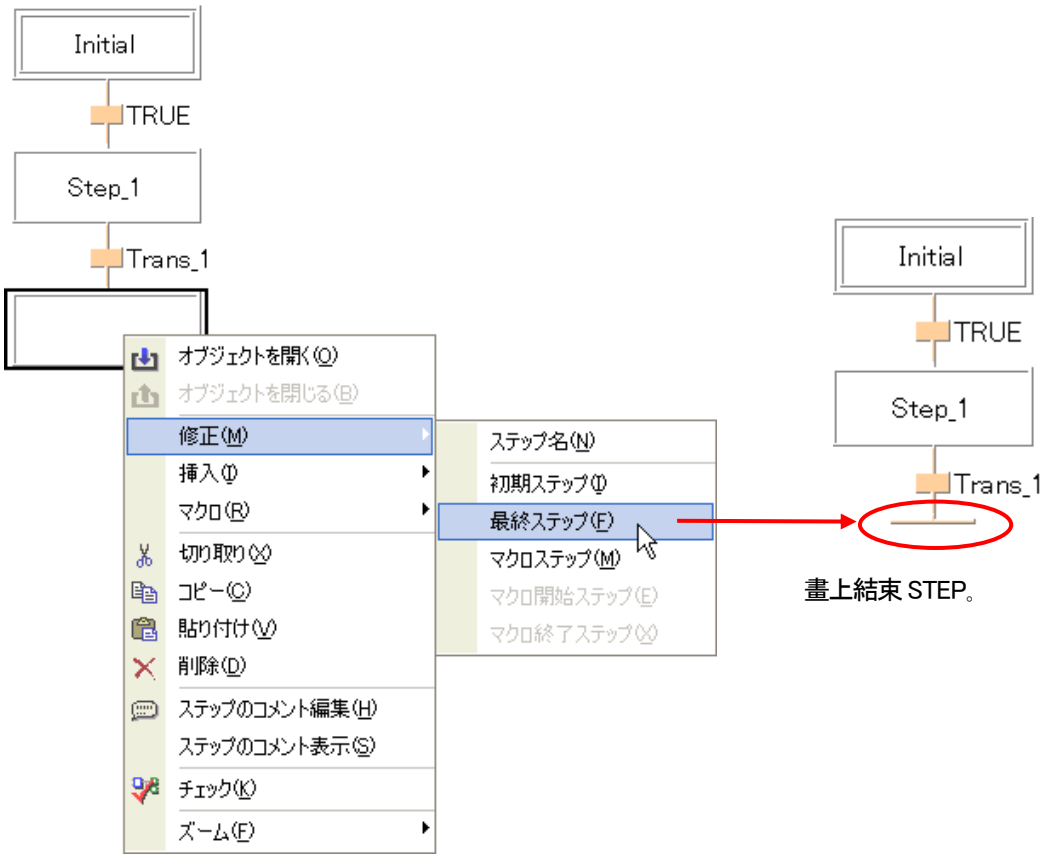
製作新的 POU 時、在 SFC 語言會預先會畫出開始 STEP・結束 STEP。
當刪錯 STEP 以及 Transition 的時候、可以用此 STEP 進行修正。

在此用結束 STEP 的描繪方法作為例子進行説明。

1. 在結束位置畫上 STEP。



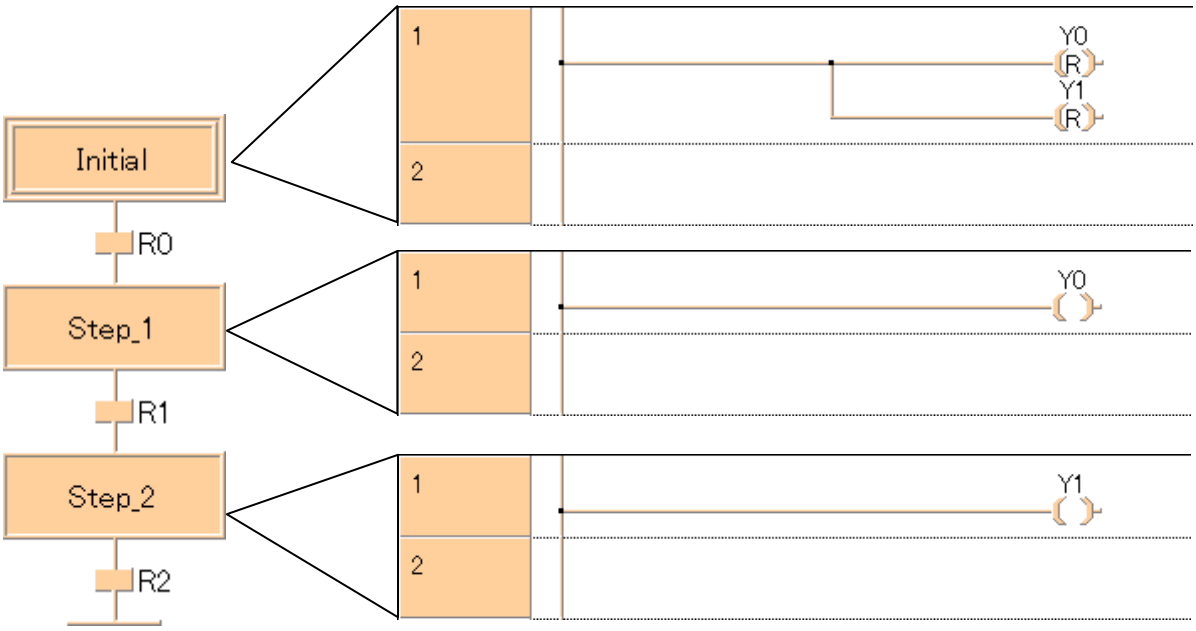
2. 將游標對準新畫上的 STEP、點擊右鍵→用「修正」點擊「結束 STEP」。



12-3 讓 SFC 動作

那麼實際確認 SFC 的動作。

製作如下圖的 SFC。

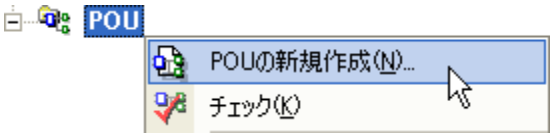


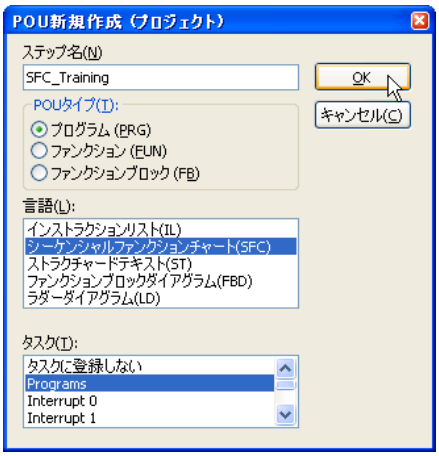
●動作內容

①在 Initial STEP 將 Y0、Y1 使其 OFF。
②Transition 的 R0 因為 ON、移到 Step_1 的 Y0 就會 ON。
③Transition 的 R1 因為 ON、移到 Step_2 的 Y1 就會 ON。
④Transition 的 R2 因為 ON、因為執行結束 STEP 會移到 Initial STEP。
（以後、從①開始循環）

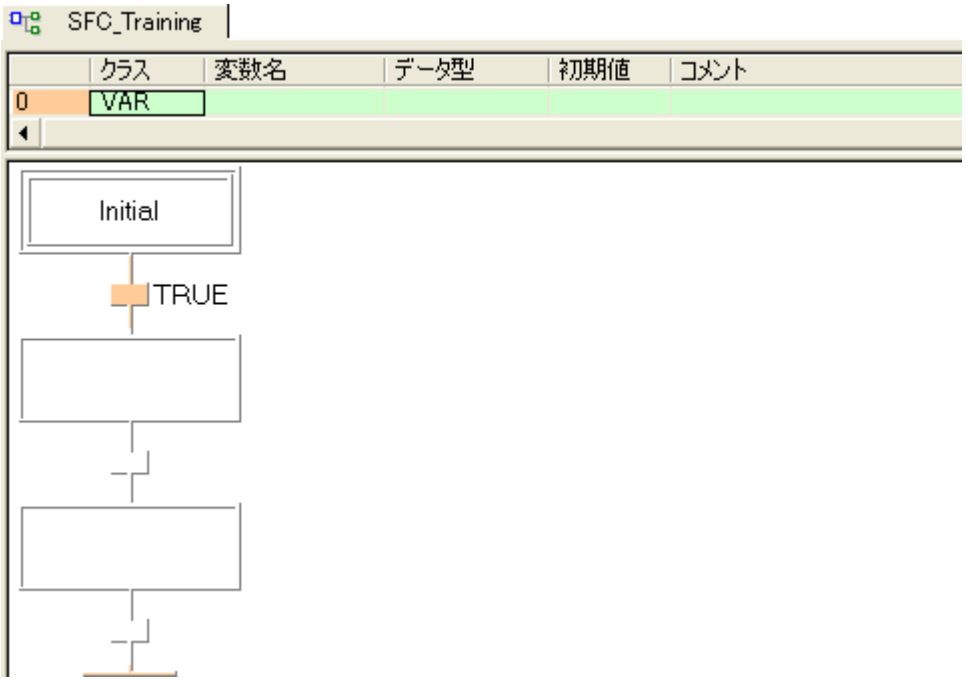
●操作步驟

①從 POU 新製作開始、將名稱為“SFC_Training”的 POU 用 SFC 語言進行製作。





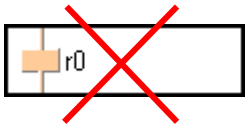
②畫出 SFC 的流程。



③將 PLC 內部 Relay 分配到 Transition。

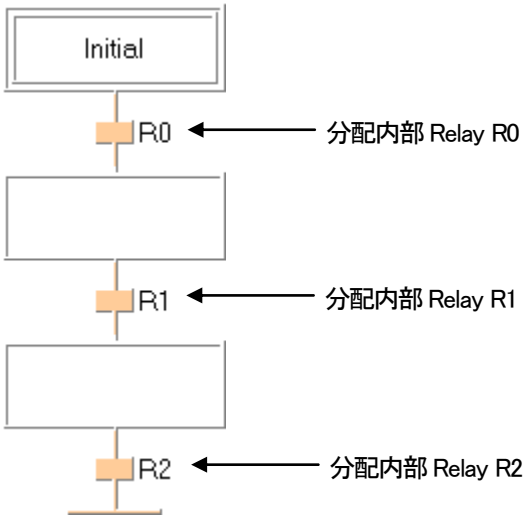
點擊所要的 Transition

輸入「R0」。
輸入結束後、用「Enter」按鍵確定。

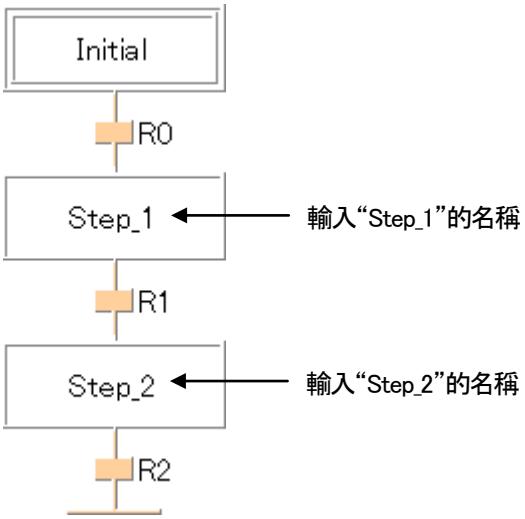


輸入小寫的「r」會被認為是變數、
請務必輸入用大寫的「R」。

同樣的分配內部 Relay 到全部的 Transition。



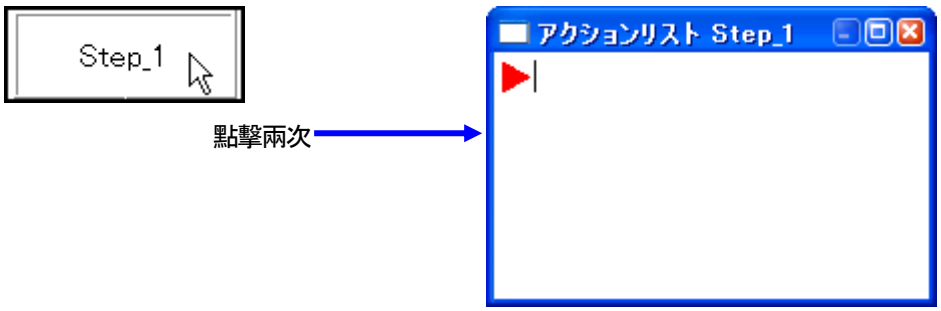
④附上名稱到 STEP。



⑤製作 Action 程式。

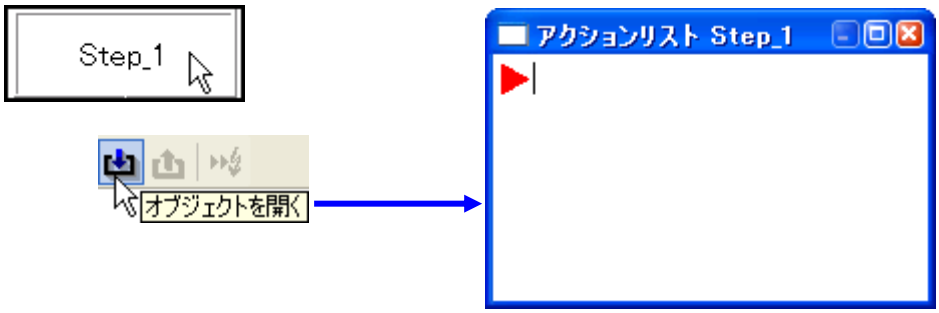
點擊所要的 STEP、在選擇 STEP 的狀態下、
點擊選單列的「打開 Object」小圖示 則 Action list 就會打開。

將所要的 STEP 點擊兩次



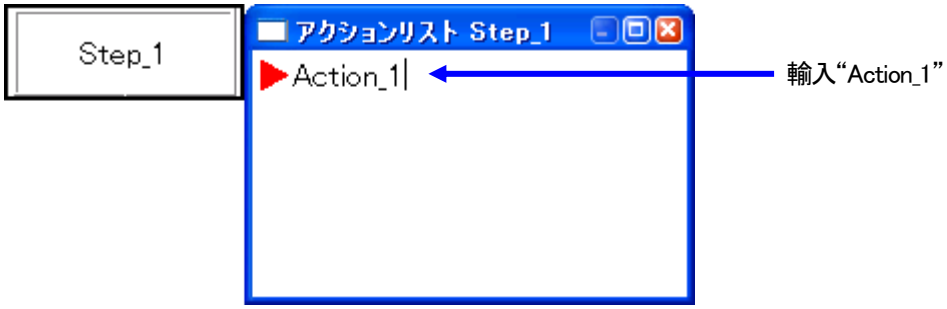
Action list 打開。

選擇所要的 STEP

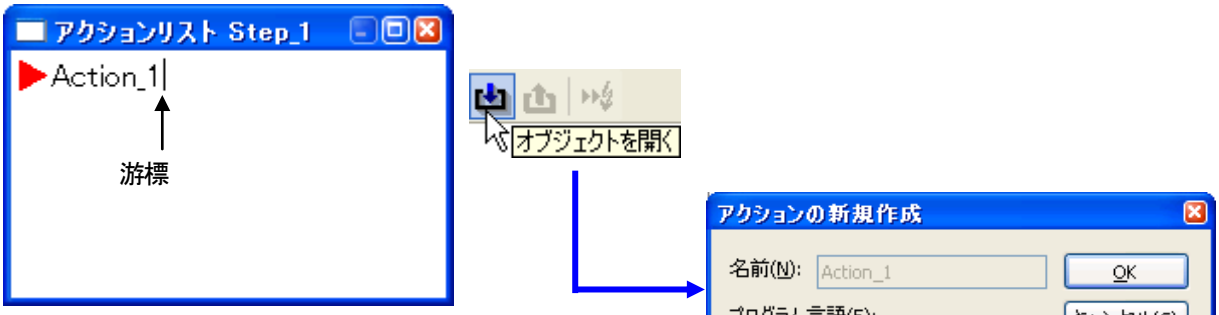


Action list 打開。

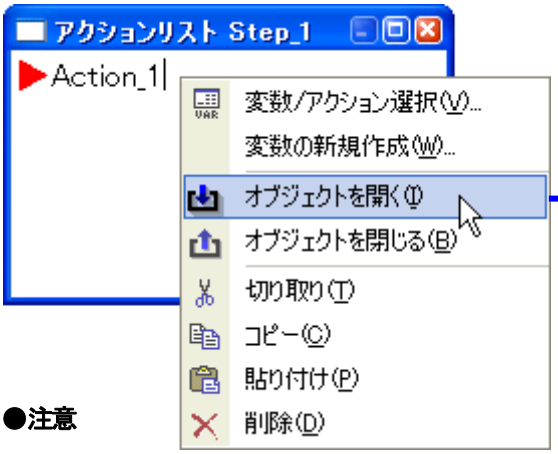
登録 Action 到 Action list
在此用“Action_1”的名稱登録 Action



製作 Action 程式。
游標會在“Action_1”的行上呈現閃爍状態、點擊「打開 Object」小圖示
打開「Action 的新製作」對話框。

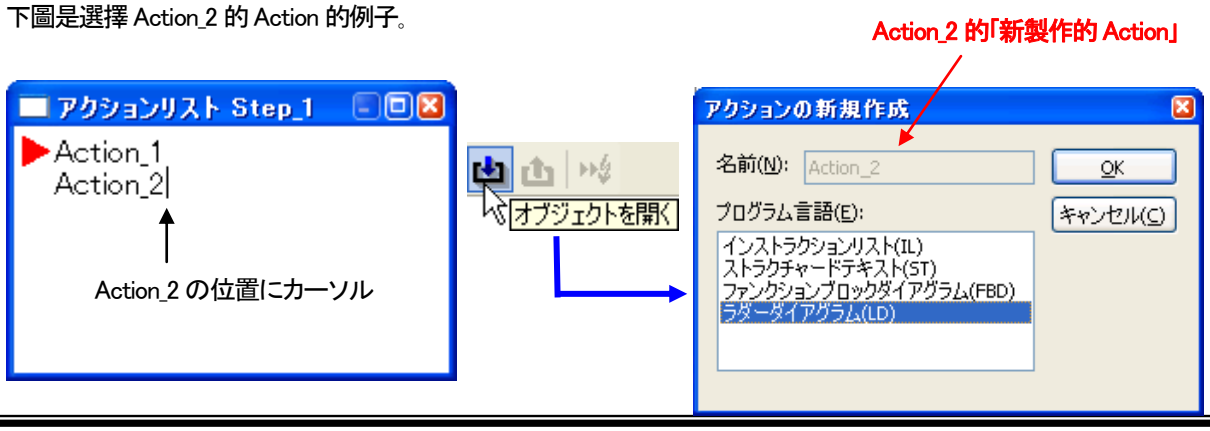


或是按右按鍵從選單打開。

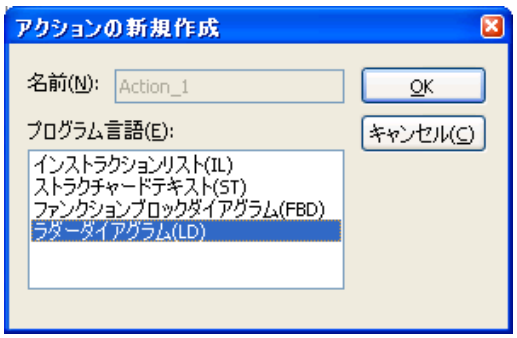


●注意

打開「Action 的新製作」對話框時、游標位置相當重要。
有 2 個以上的 Action 存在時、為了判斷是那個 Action。
下圖是選擇 Action_2 的 Action 的例子。

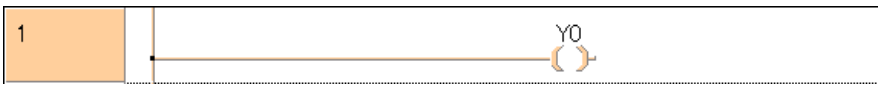


在此因為在程式語言選擇 LD、
選擇「階梯圖(LD)」請按「OK」鍵。

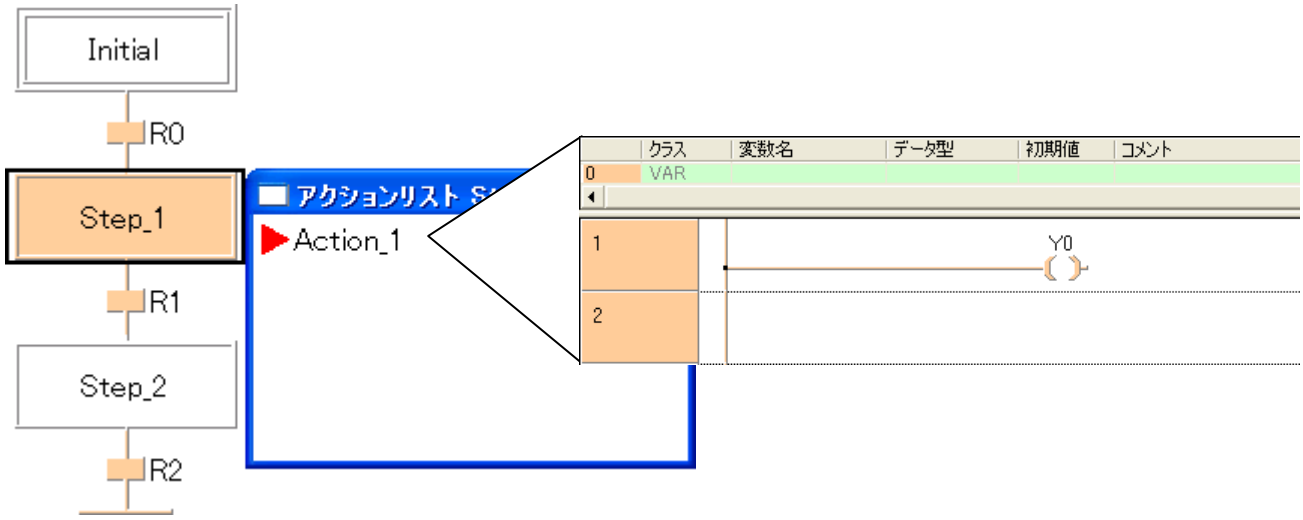


打開 Action_1。

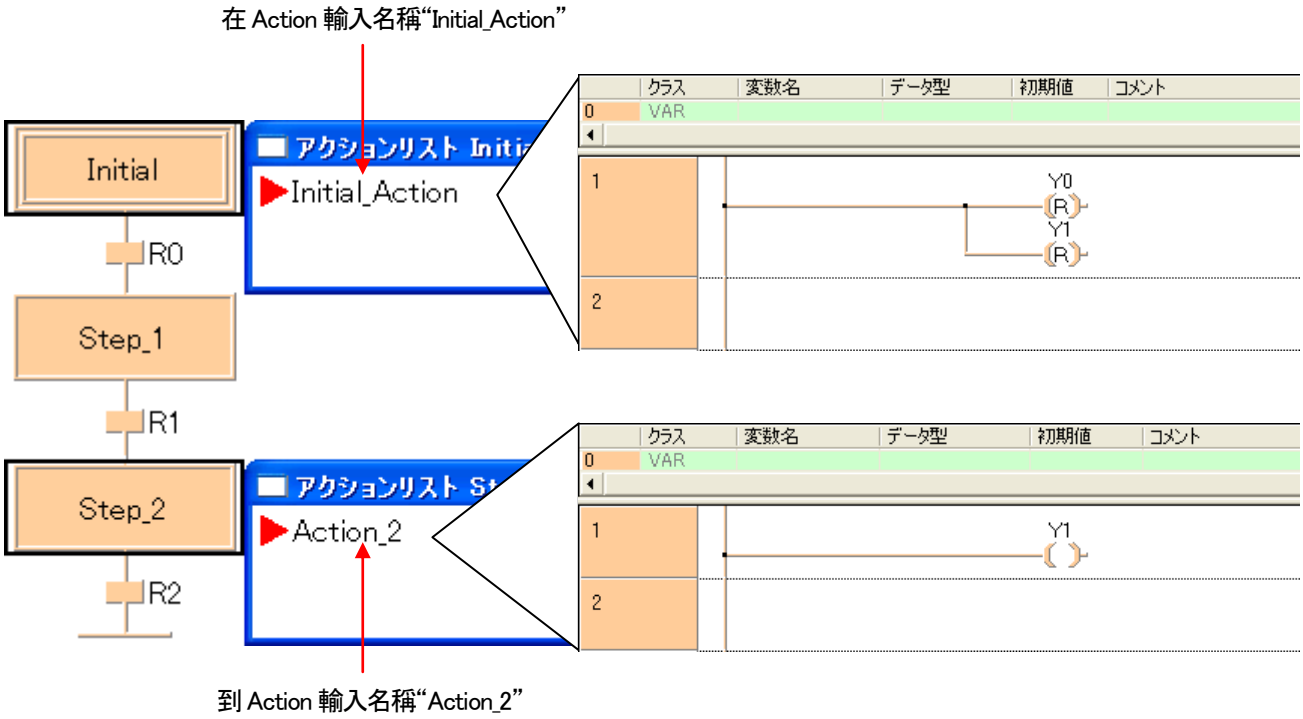
製作程式、完成 Step_1 的 Action(Action_1)。



到此的操作到下圖狀態為完成。



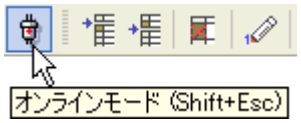
對全部的 STEP 同樣的登録 Action。

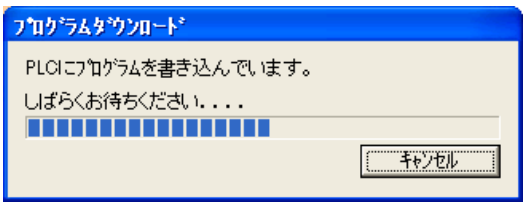


到此畫 SFC 流程為完成。

⑥將 POU 下載到 PLC。

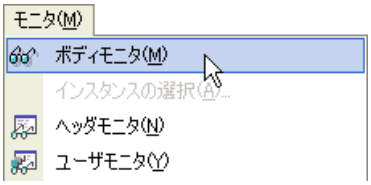
編譯後、請打開 On line 下載到 PLC。



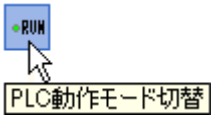


⑦確認動作

請確認是否變成 Monitor Mode 執行状態。

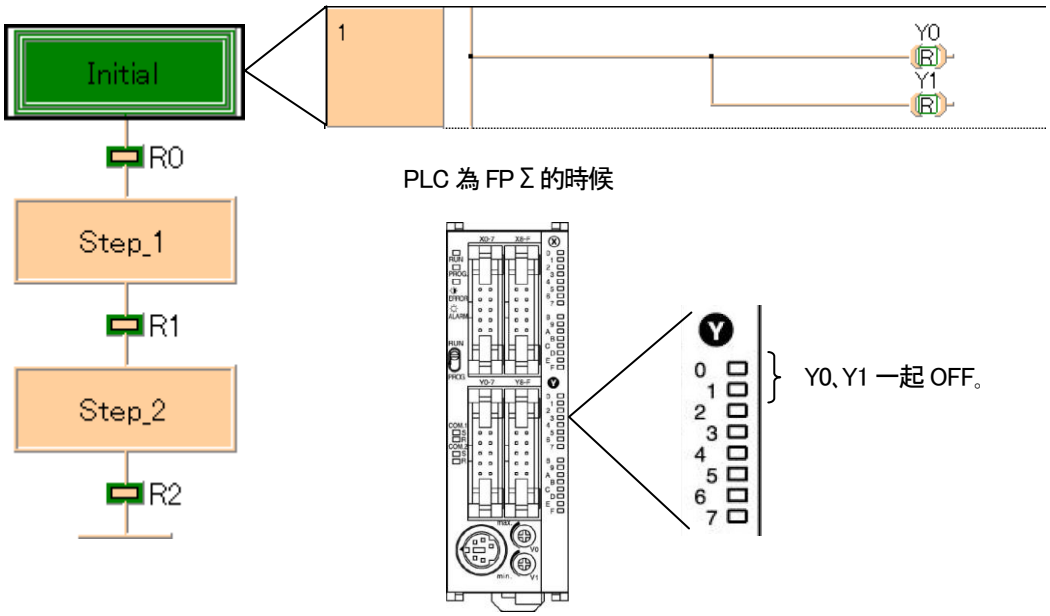


PROG. Mode 的時候請切換程 RUN Mode。



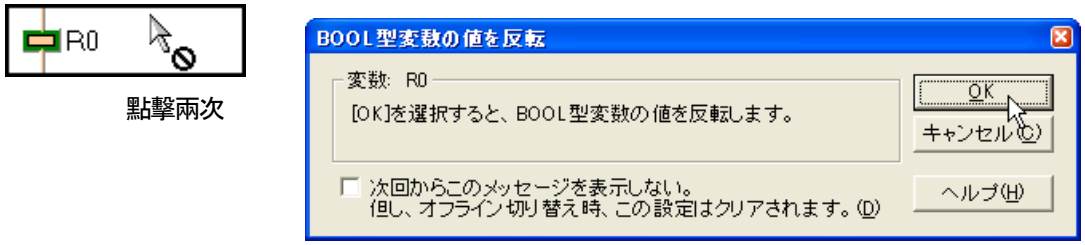
移到「Initial」。

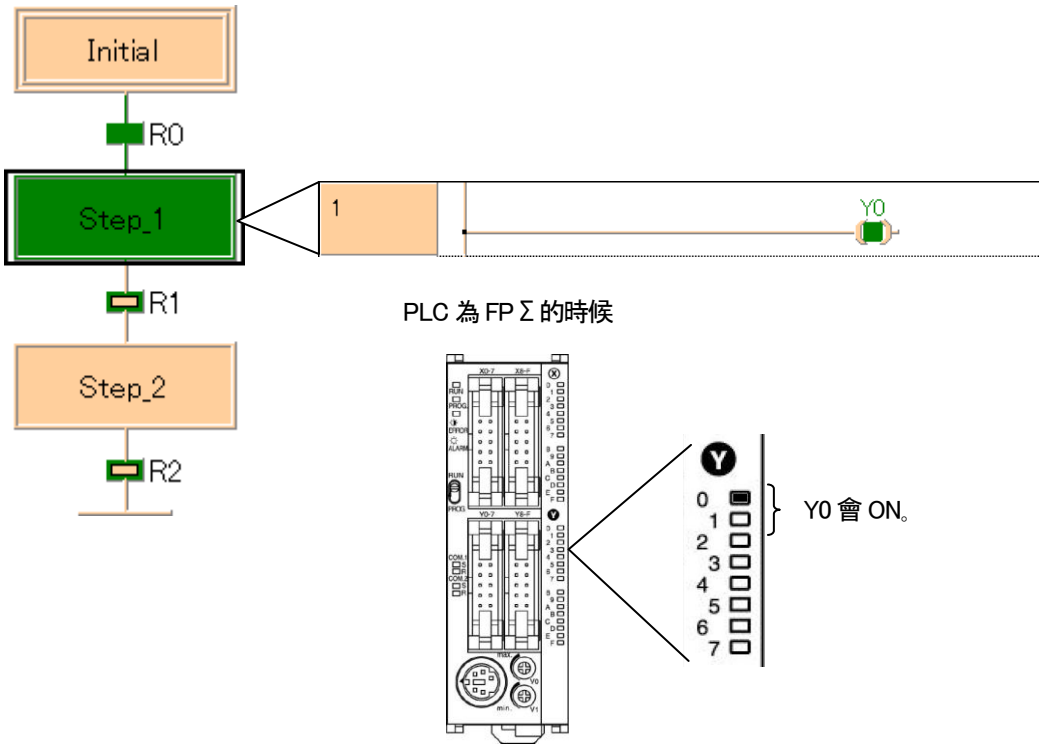
PLC 從 PROG 變為 RUN 狀態時會移到「Initial」STEP。



移到「STEP1」。

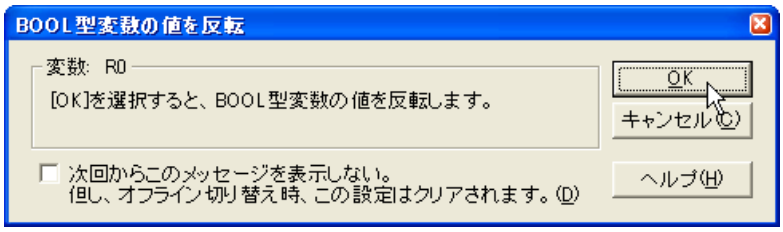
將 Transition 的 R0 使其 ON 移到「Step_1」STEP。





將 Transition R0 使其 OFF。

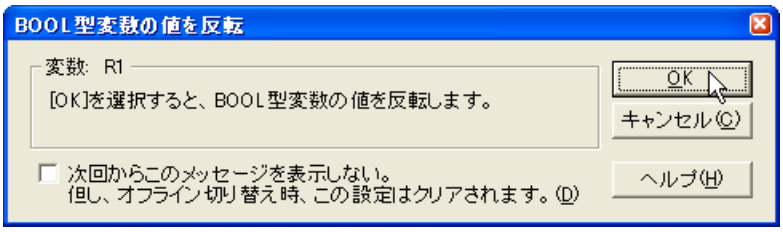
點擊兩次



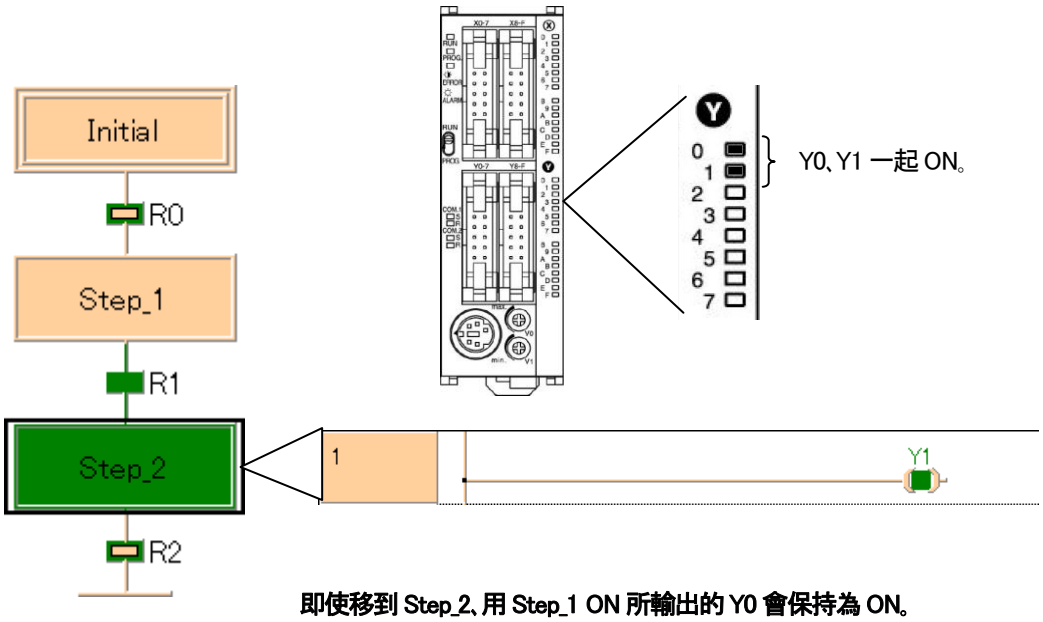
移到「Step_2」。

將 Transition R1 使其 ON 移到「Step_2」 STEP。

點擊兩次



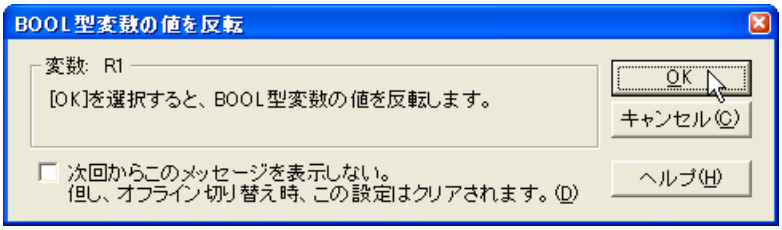
PLC 為 FP Σ 的時候



即使移到 Step_2, 用 Step_1 ON 所輸出的 Y0 會保持為 ON。
STPE 移動到下個 STEP 時, 會保持移行時的状態。

TransitionR1 將其 OFF。

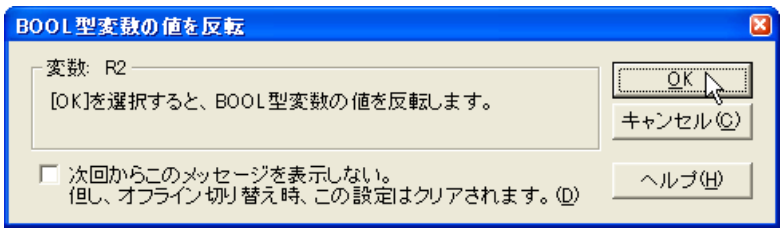
點擊兩次

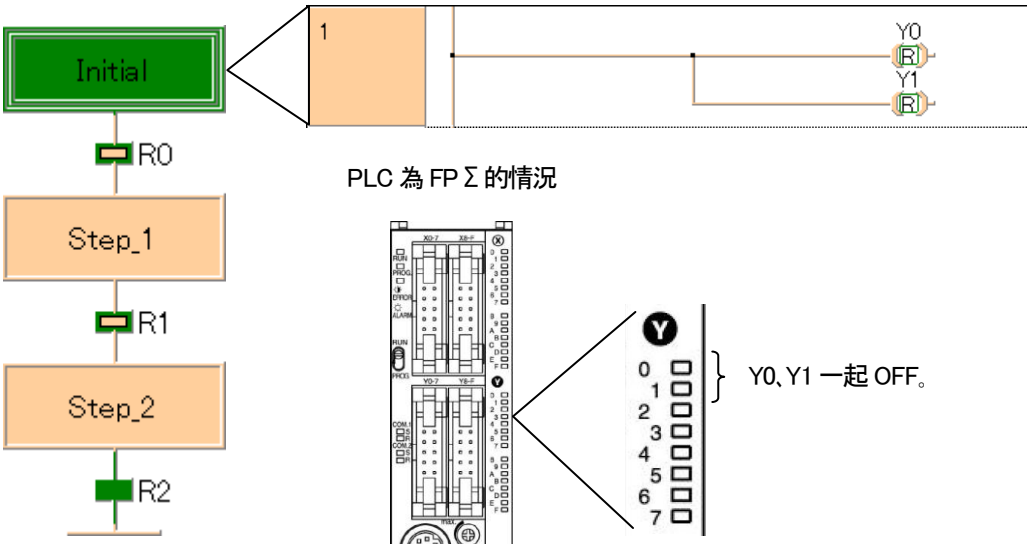


移到「結束 STEP」。

Transition R2 ON 會移動到「結束 STEP」。

點擊兩次





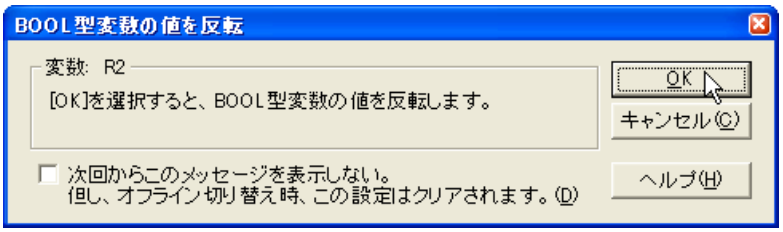
PLC 為 FP Σ 的情況

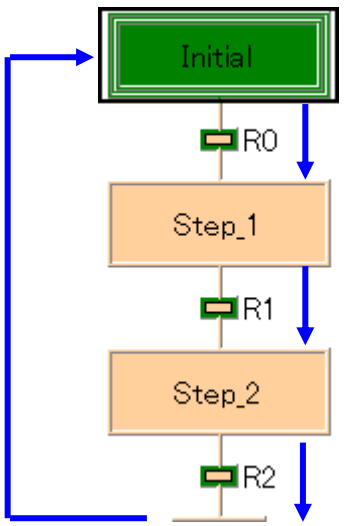
Y0、Y1 一起 OFF。

在執行結束 STEP 時，會回到「Initial」STEP，
為了讓 Initial STEP 的 Action 動作，Y0、Y1 會 OFF。

Transition R2 讓其 OFF。

點擊兩次

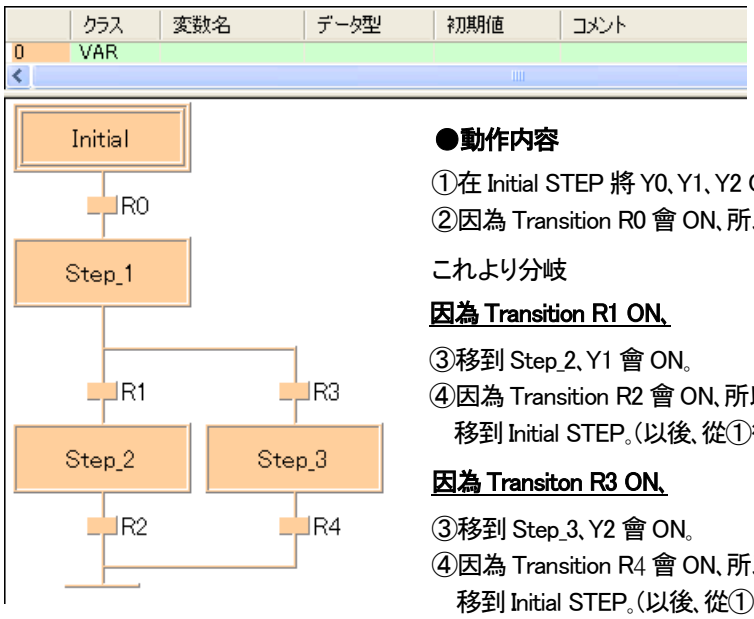




因為會回到 Initial STEP，
可以再次重複同樣動作。

■課題

練習下圖的課題看看。



●動作內容

①在 Initial STEP 將 Y0、Y1、Y2 OFF。
②因為 Transition R0 會 ON、所以會移動到 Step_1 讓 Y0 變 ON。

これより分岐

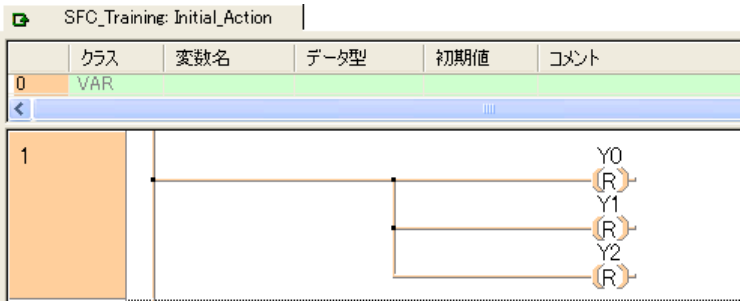
因為 Transition R1 ON、

③移到 Step_2、Y1 會 ON。
④因為 Transition R2 會 ON、所以會執行結束 STEP
移到 Initial STEP。(以後、從①循環)

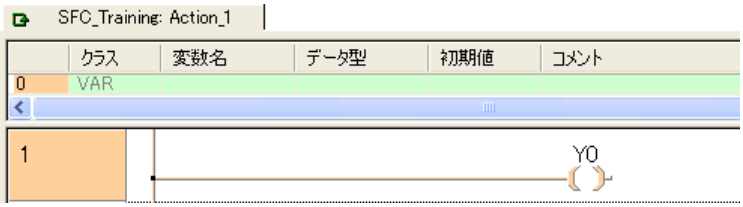
因為 Transiton R3 ON、

③移到 Step_3、Y2 會 ON。
④因為 Transition R4 會 ON、所以會執行結束 STEP
移到 Initial STEP。(以後、從①循環)

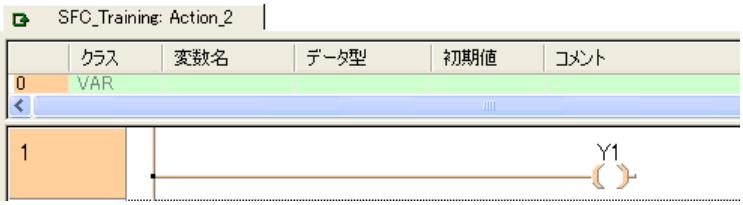
①登録 Action 名稱 Initial_Action 到 Initial STEP。



②登録 Action 名稱 Action_1 到「Step_1」STEP。



③登録 Action 名稱 Action_2 到「Step_2」STEP。

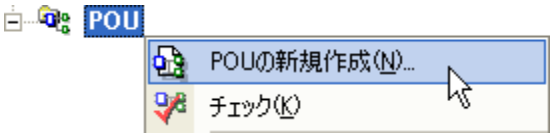


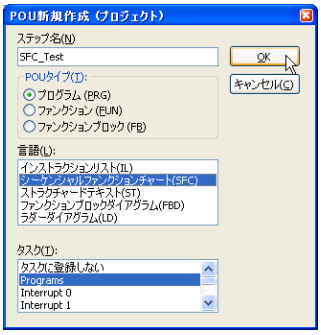
④登録 Action 名稱 Action_3 到「Step_3」STEP。

12-4 應用的 STEP 移動動作

在此介紹應用的 STEP 移動動作。

從 POU 新製作將名稱為“SFC_TEST”的 POU 用 SFC 語言製作。





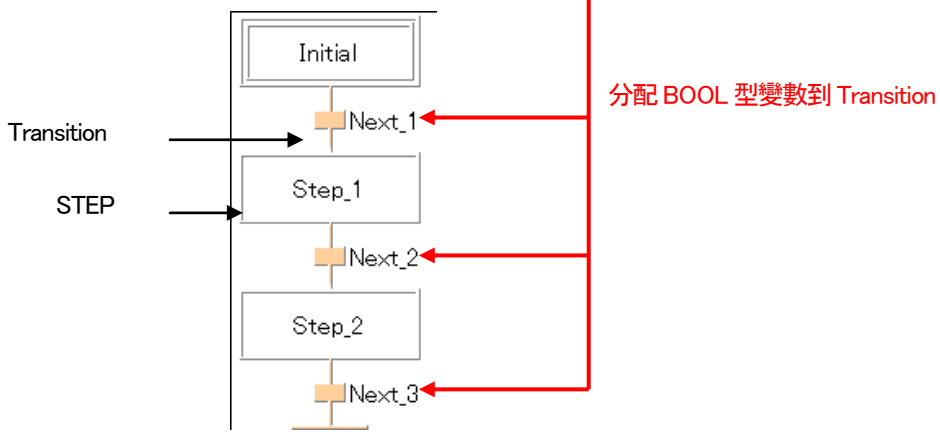
12-4-1 使用變數的 STEP 移動動作

介紹將配變數到 Transition、用其變數的 ON 移動到下個 STEP 的情況。
首先做好下圖的流程圖。

●Header

	クラス	変数名	データ型	初期値
0	VAR	Next_1	BOOL	FALSE
1	VAR	Next_2	BOOL	FALSE
2	VAR	Next_3	BOOL	FALSE

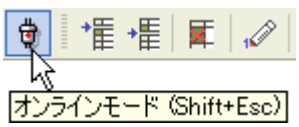
●Body



在此變數“Next_1”如果 ON 的話、會移動到「Step_1」、變數“Next_2”如果 ON 的話會移動到「Step_2」、變數“Next_3”如果 ON 的話會移動到「Initial」。

■程式的下載

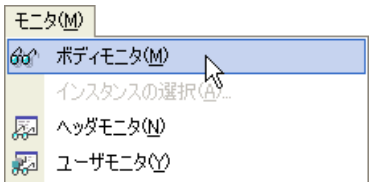
編譯後、Online 後下載到 PLC。



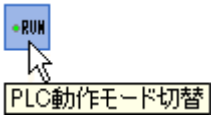
■動作確認

那麼依照移動的順序來看看。

請確認是否變為監控狀態。

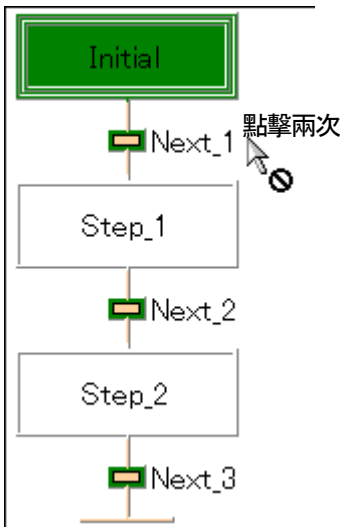


PROG.模式的時候請切換為 RUN 模式。

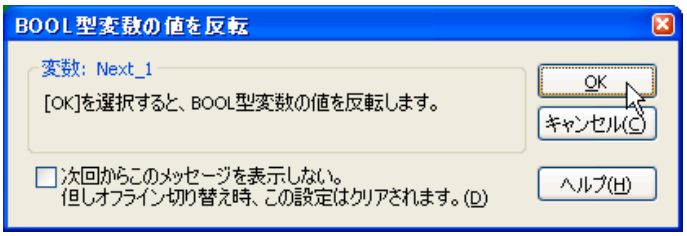


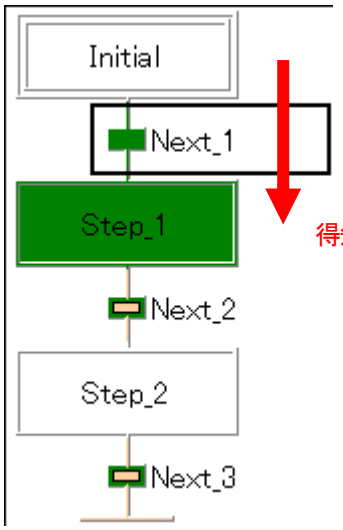
①移動到「Step_1」。

將 Transition 的“Next_1”點擊兩次後變成 ON。



點擊兩次

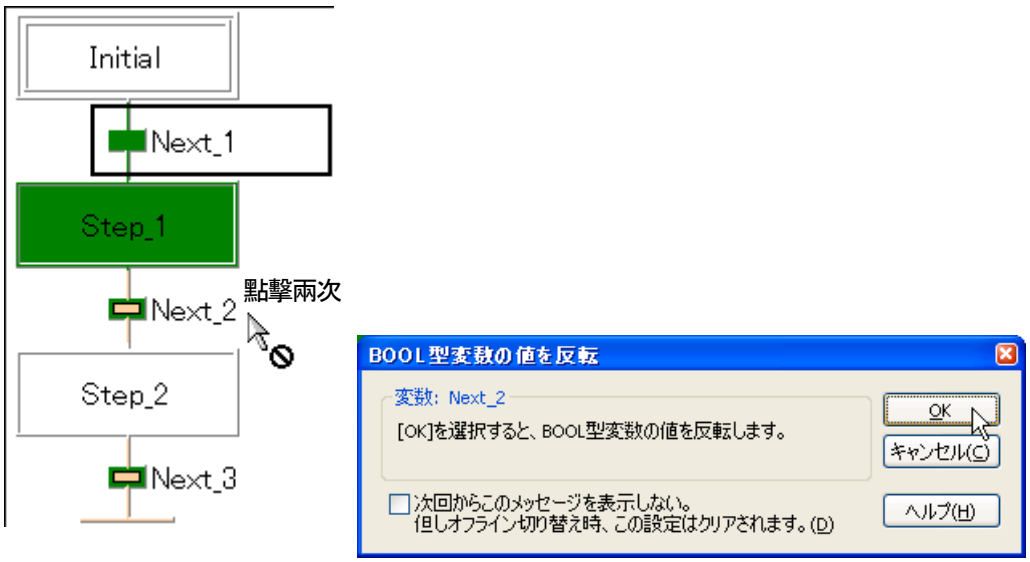


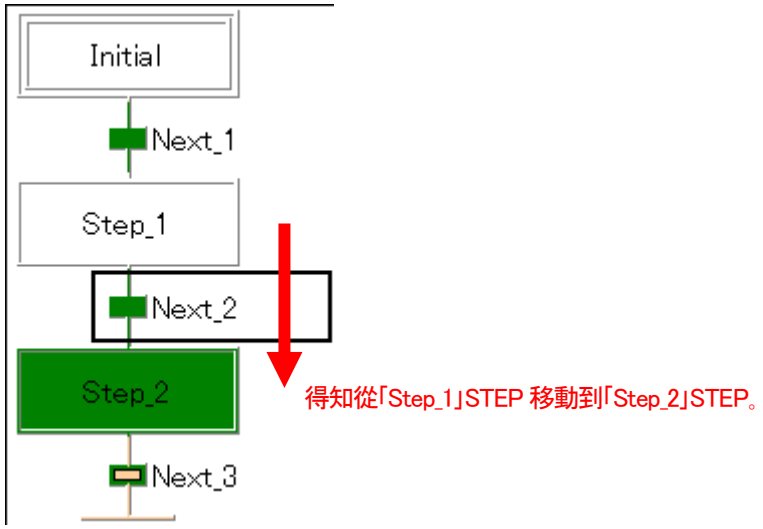


得知從「Initial」STEP 移動到「Step_1」STEP。

②移動到「Step_2」。

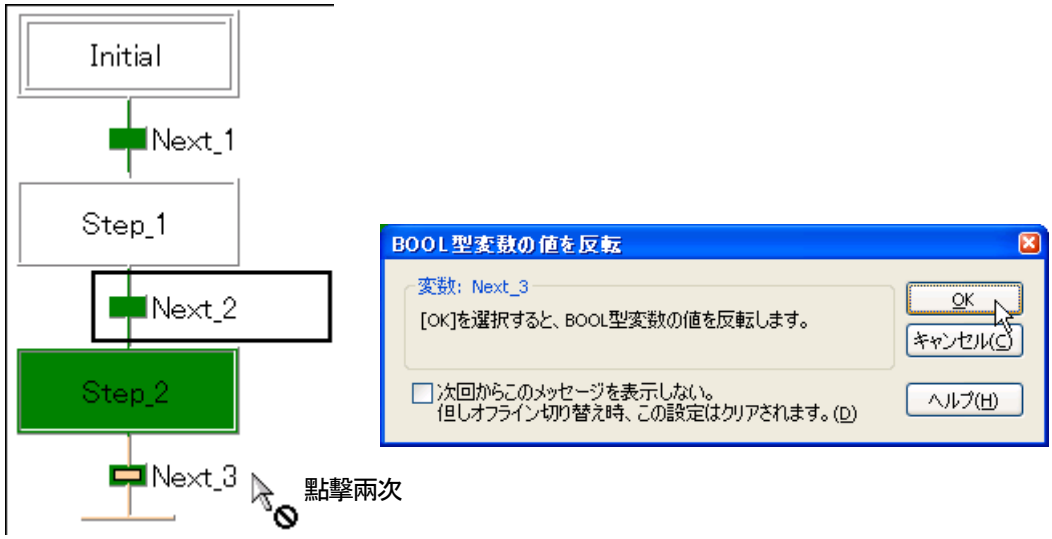
Transition 的“Next_2”點擊兩次使其 ON。



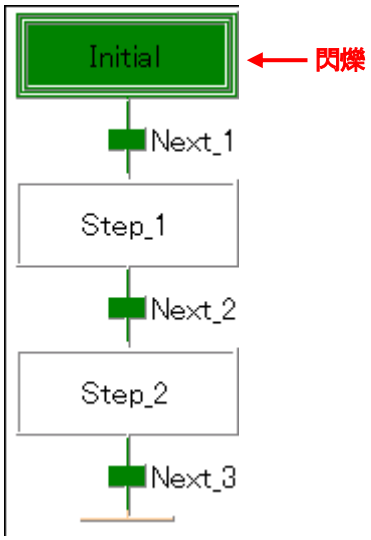


③移動到「Initial」。

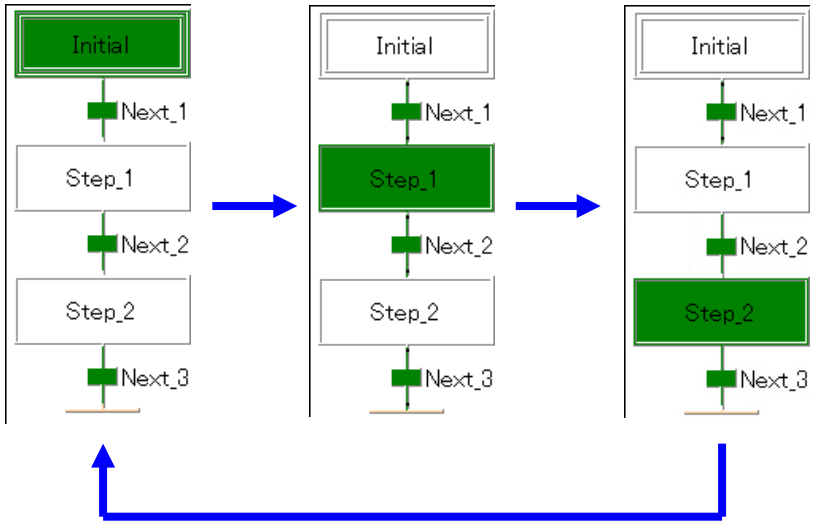
Transition 的“Next_3”點擊兩次使其 ON。

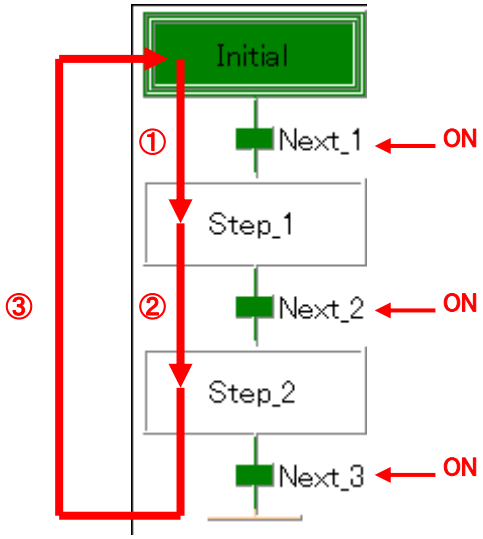


「Initial」STEP 會變成閃爍的狀態。



此為重複循環從「Initial」到「Step_2」執行的狀態。





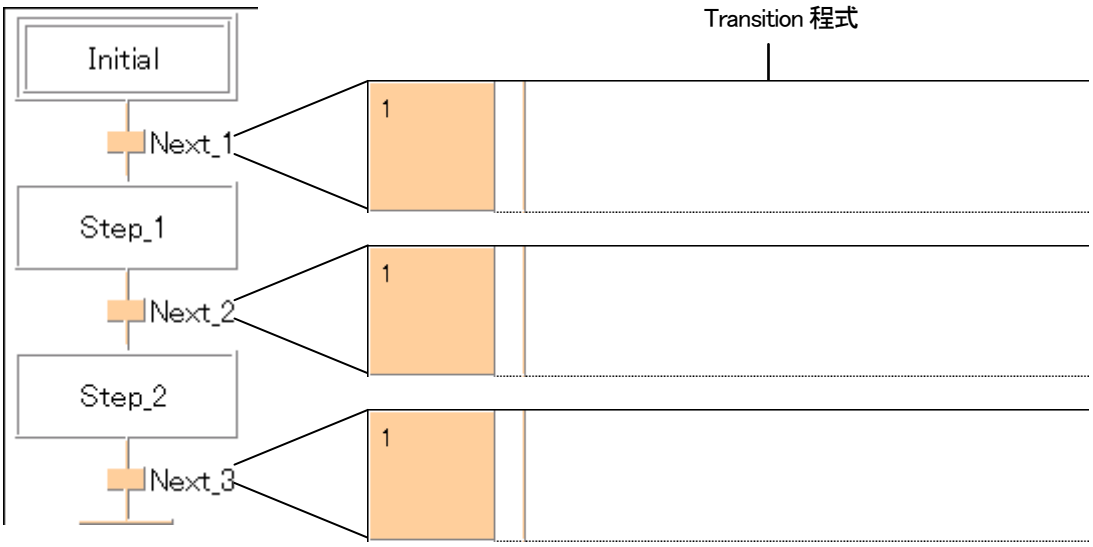
此為了讓 Transition“Next_1”、“Next_2”、“Next_3”全部變成 ON、而進行連續移動。

Transition 從 FALSE(OFF)變成為 TRUE(ON)的時候、移動到 STEP 的情況像上圖的做法是不夠的。

接下來介紹從 Transition 的啓動 ON(FALSE(OFF)到 TRUE(ON)の時)移動STEP的方法。

12-4-2 用 Transition 的 ON 移動 STEP

用 Transition 的 ON 移動 STEP、必須要在 Transition 內寫上程式。



●注意

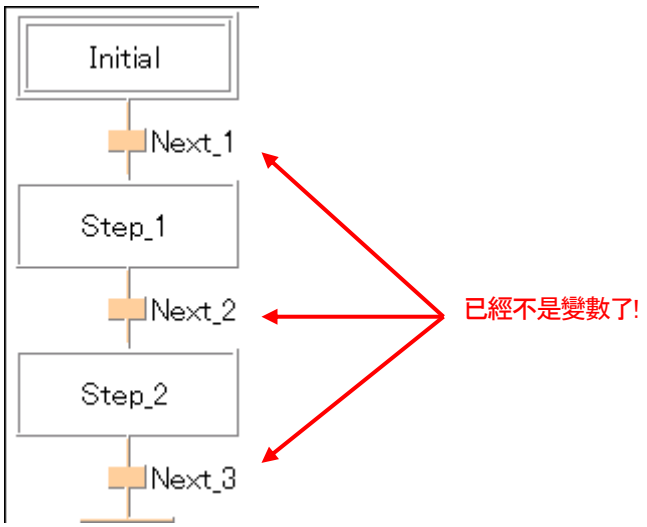
分配變數到 Transition、則不能做成 Transition 程式。
製作 Transition 程式時、請注意必須要讓 Transition 程式名稱和變數名稱有所不同。

到前項為止、已經分配變數到 Transition、首先刪除 Header 的變數。

	クラス	変数名	データ型	初期値
0	VAR	Next_1	BOOL	FALSE
1	VAR	Next_2	BOOL	FALSE
2	VAR	Next_3	BOOL	FALSE

刪除“Next_1”、“Next_2”、“Next_3”

	クラス	変数名	データ型	初期値
0	VAR			

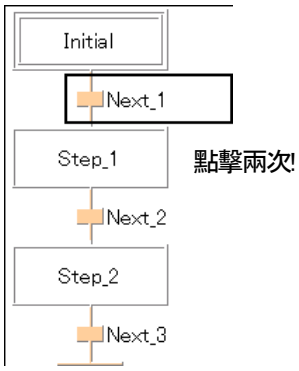


①要移動的信号作為新的變數進行登録。

	クラス	変数名	データ型	初期値
0	VAR	Signal_1	BOOL	FALSE
1	VAR	Signal_2	BOOL	FALSE
2	VAR	Signal_3	BOOL	FALSE

在此作為 BOOL 型變數宣告“Signal_1”、“Signal_2”、“Signal_3”。

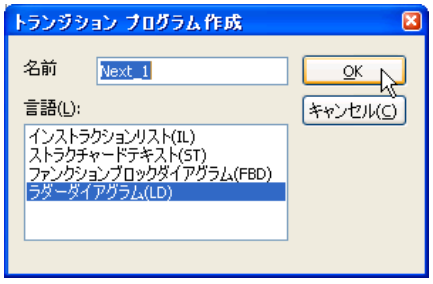
②製作 Transition 程式。
首先在“Next_1”製作。
用游標對準 Transition“Next_1”點擊兩次、
或是用右鍵選擇「打開 Object」。



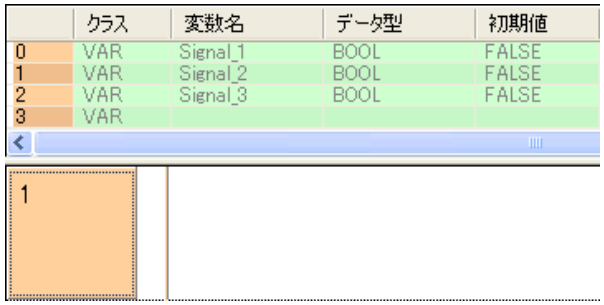
點擊兩次!

因為會顯示「製作 Transition 程式」的對話框、
請點選所用的程式語言按下「OK」按鍵。

在此選擇 LD 語言。



顯示 Transition 程式的 Header 和 Body。



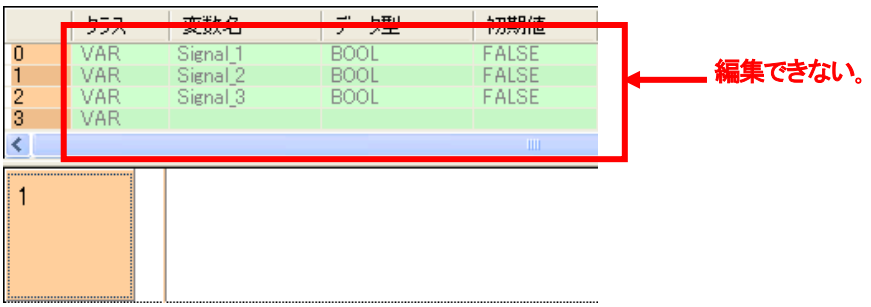
	クラス	変数名	データ型	初期値
0	VAR	Signal_1	BOOL	FALSE
1	VAR	Signal_2	BOOL	FALSE
2	VAR	Signal_3	BOOL	FALSE
3	VAR			

只有 1 個 Network

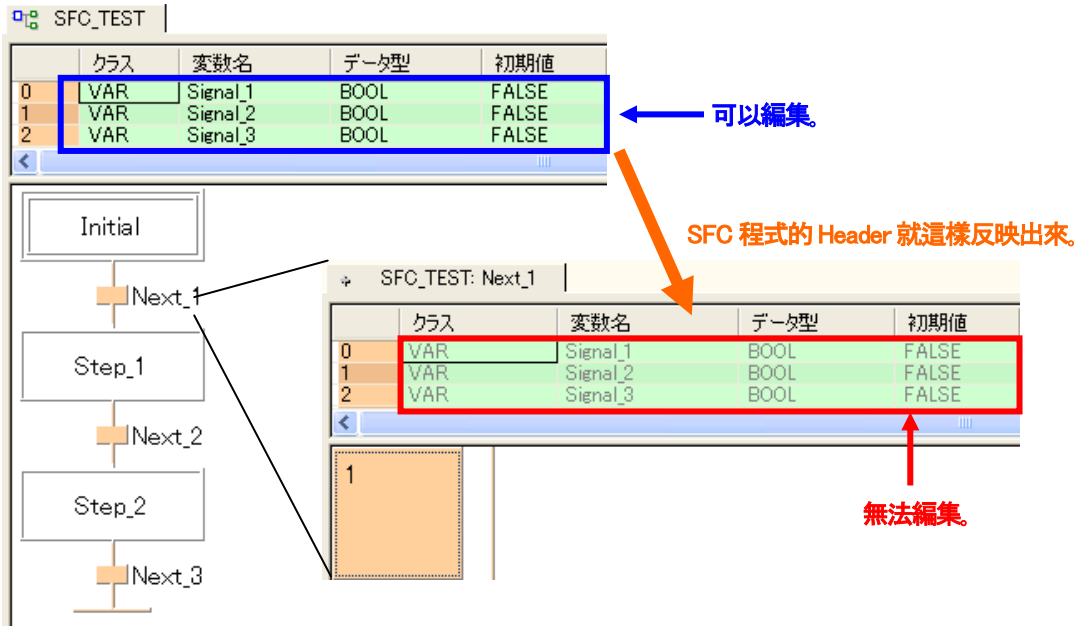
重要:用 LD 語言的 Transition 程式可以記述的只有 1 個 Network

在此請看 Transition 程式內的 Header。

Transition 程式內的 Header 無法編集。

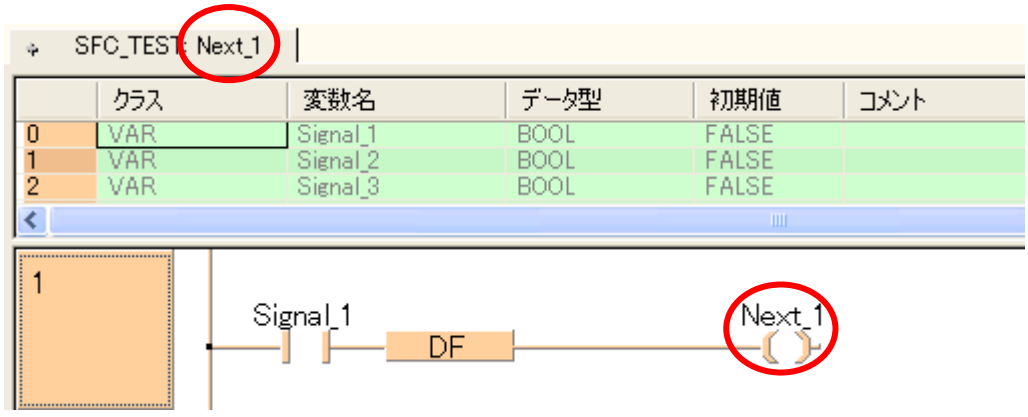


此為 POU 的母體為 SFC 畫面。



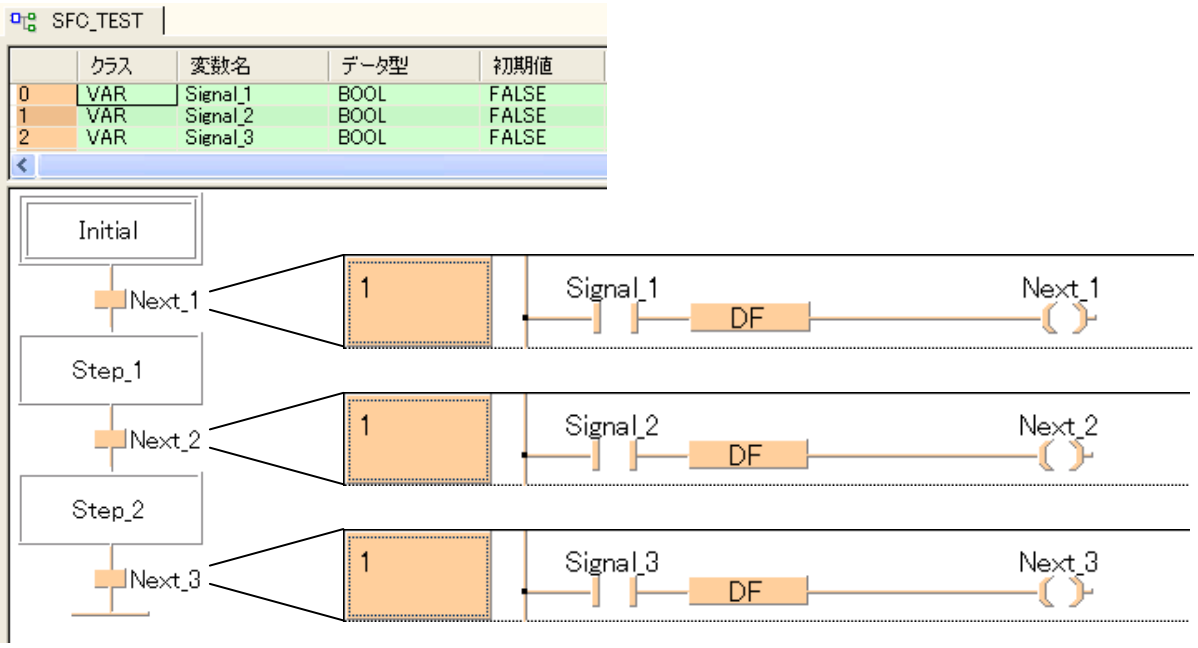
從 Header 宣告變數、或是修正・變更時從 SFC 程式的 Header 進行。

在「"Signal1"的開始信号從「Initial」STEP 移動到「Step_1」STEP」的記述程式。



輸出 Transition 名稱(Next_1)為重點

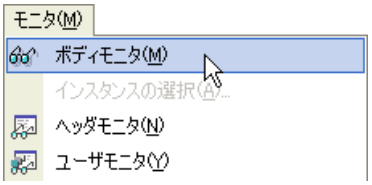
同樣的,將"Next_2"和"Next_3"的 Transition 也記述程式。



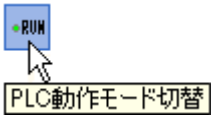
■動作確認

那麼看看移動順序。

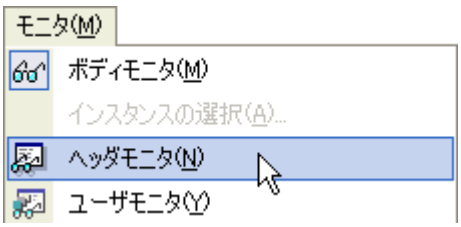
請確認是否變成監控模式執行狀態。



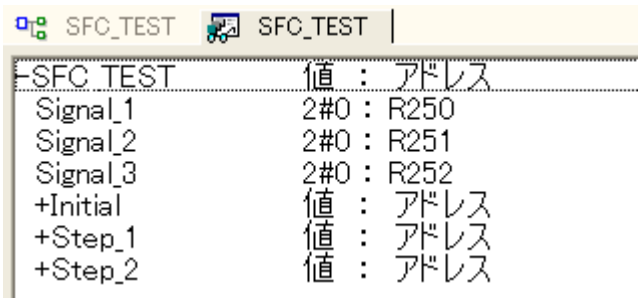
PROG.模式時請將切換到 RUN 模式。



為了讓變數"Signal_1"、"Signal_2"、"Signal_3"會 ON,要起動「Header 監控」。



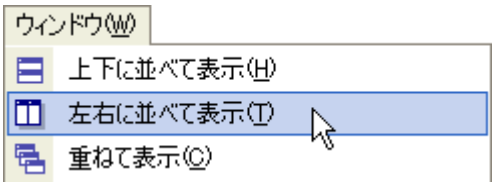
起動「Header 監控」。



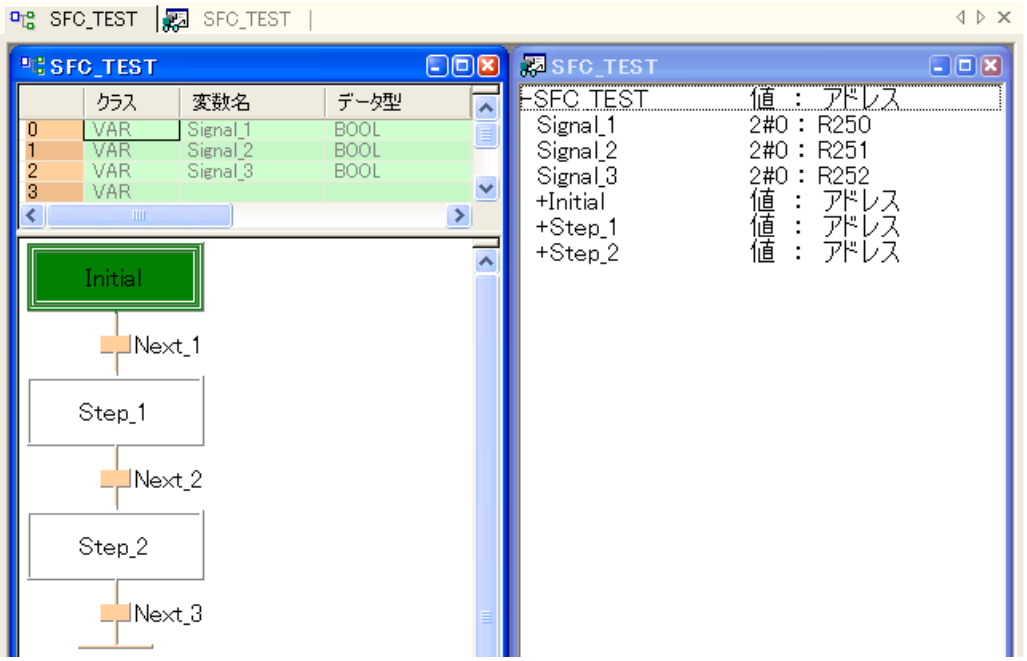
為了「SFC 畫面」和「Header 監控」可以同時監控所以要將調整視窗。

點選“SFC_TEST”、SFC 畫面會顯示到前面。

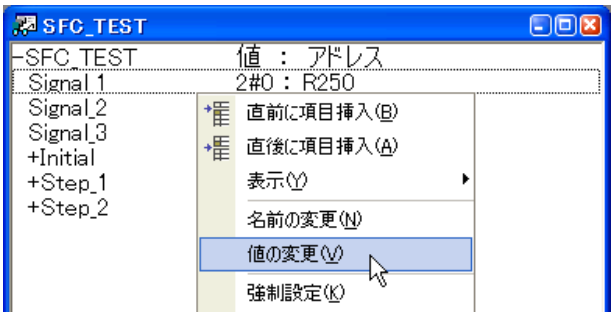
視窗左右並排顯示。



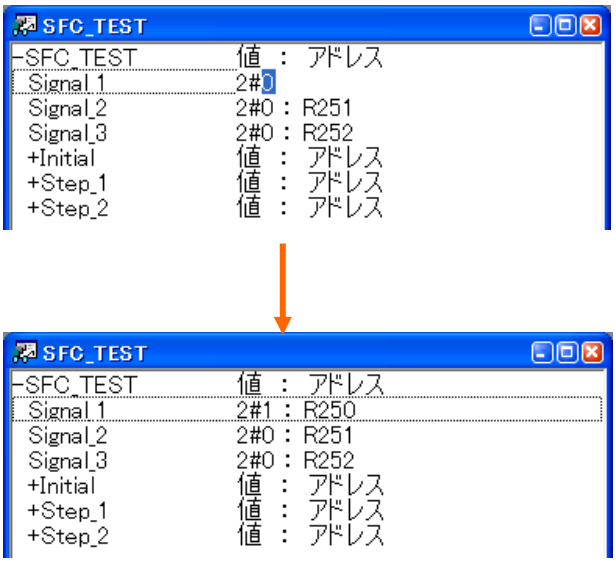
如下圖般在左右邊顯示視窗。
那麼將“Signal_1”ON 移動到「Step_1」看看。



從 Header 監控到“Signal_1”的位置按下右鍵選單點選「值的變更」。



如下圖會顯示催促值的輸入、
ON 的意思為輸入“1”請按下「Enter」鍵做確定。

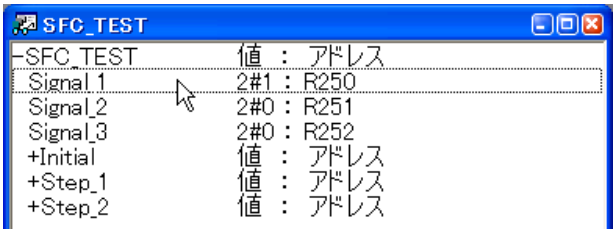


●注意
“2#”的顯示為“Signal_1”是 BOOL 型的變數。
OFF 的意思為“0”、ON 所意味的是輸入“1”以外的值會變成 ERROR。

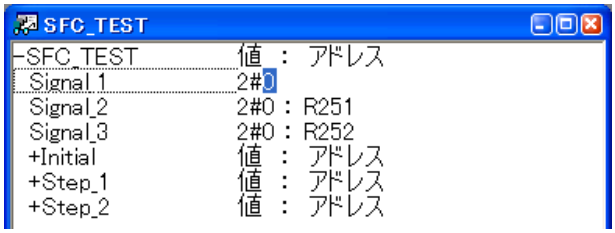
■備考 在 Header 監控的值輸入方法

從上面的右按鍵選單的變更方法以下有其他方法。

選擇目標的變數。

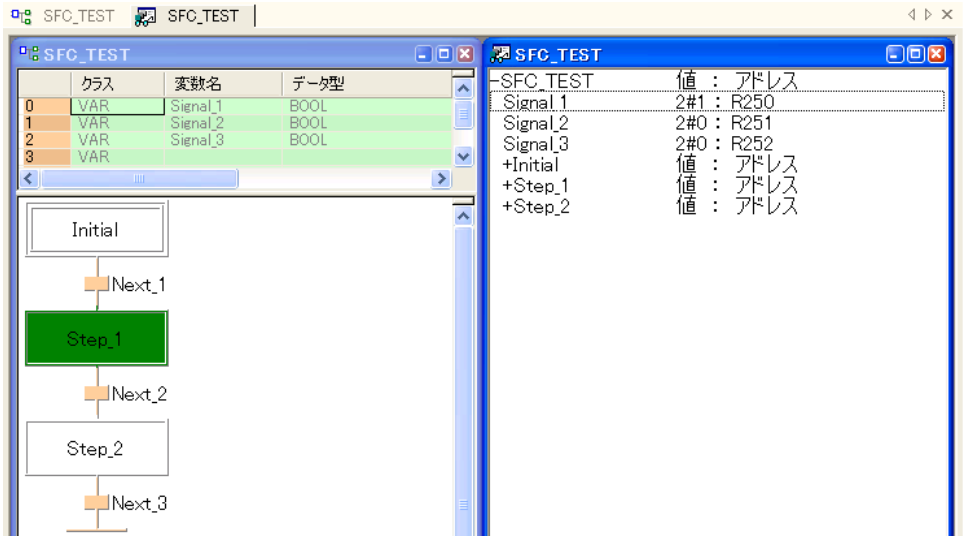


「スペース」キーを押す。

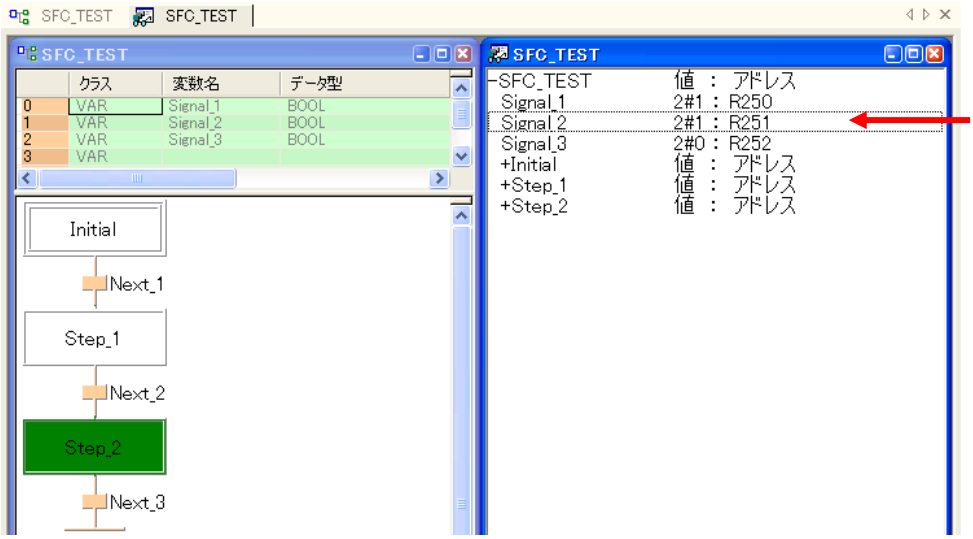


以下的上述方法為相同。

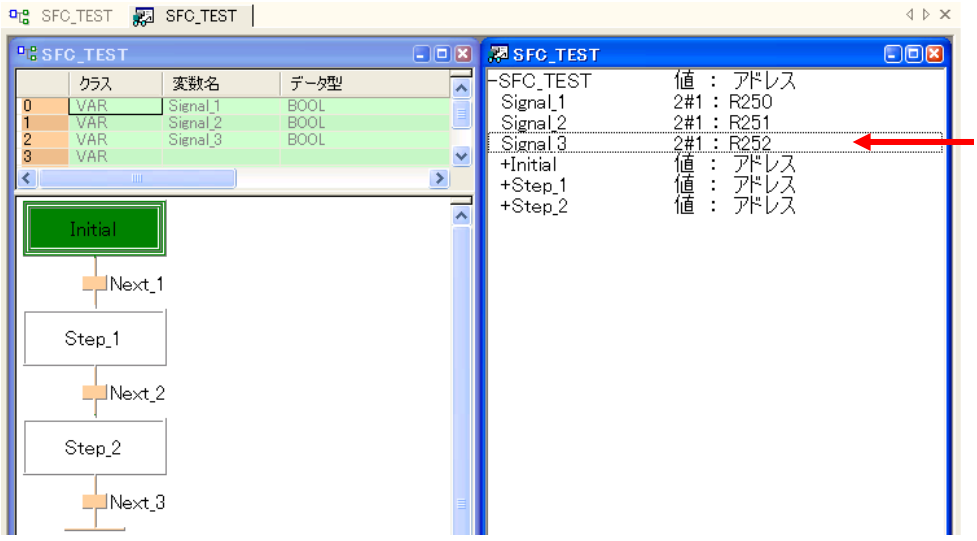
看到 SFC 畫面、得知 STEP 移動到「Step_1」。



STEP 移動到「Step_2」看看。

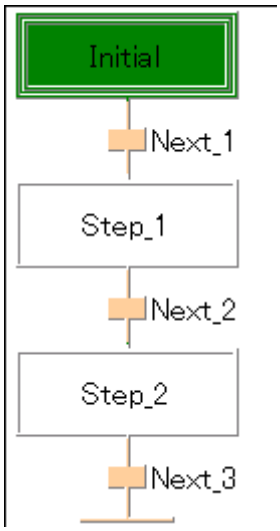


STEP 移動到「Initial」。

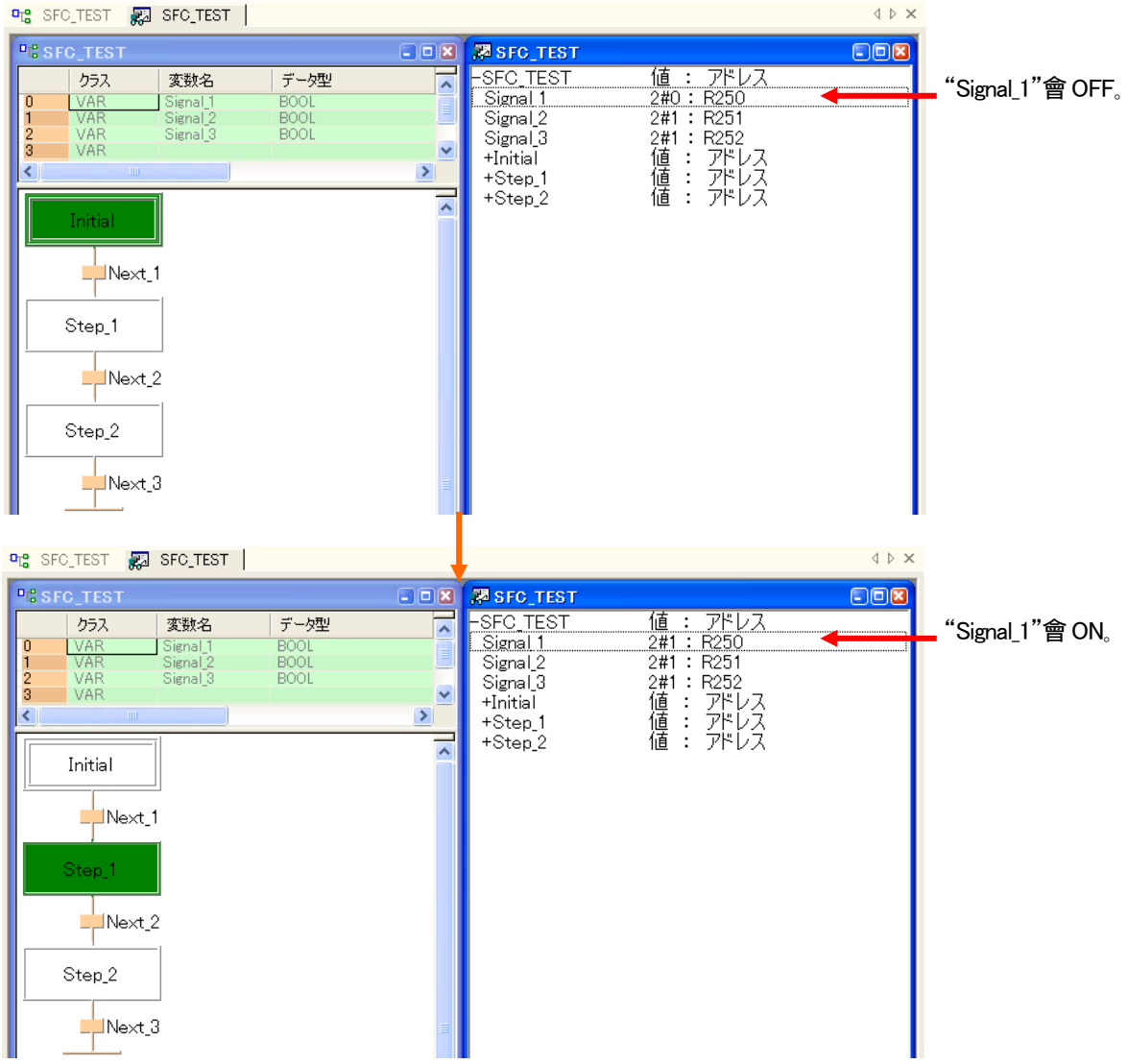


接下來怎麼辦

在「Initial」STEP 位置進行控制、無法移動到下個 STEP。

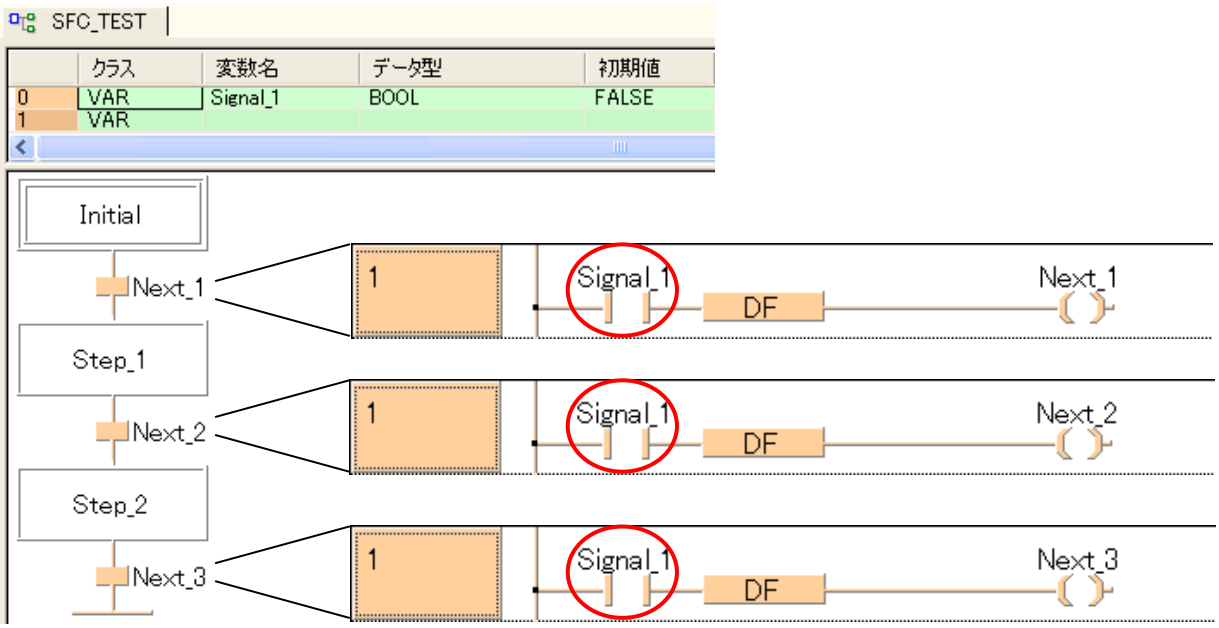


在 Transition 的“Next_1”ON 確認看看移動到下個 STEP。



得知移動“Signal_1”的 OFF→ON。

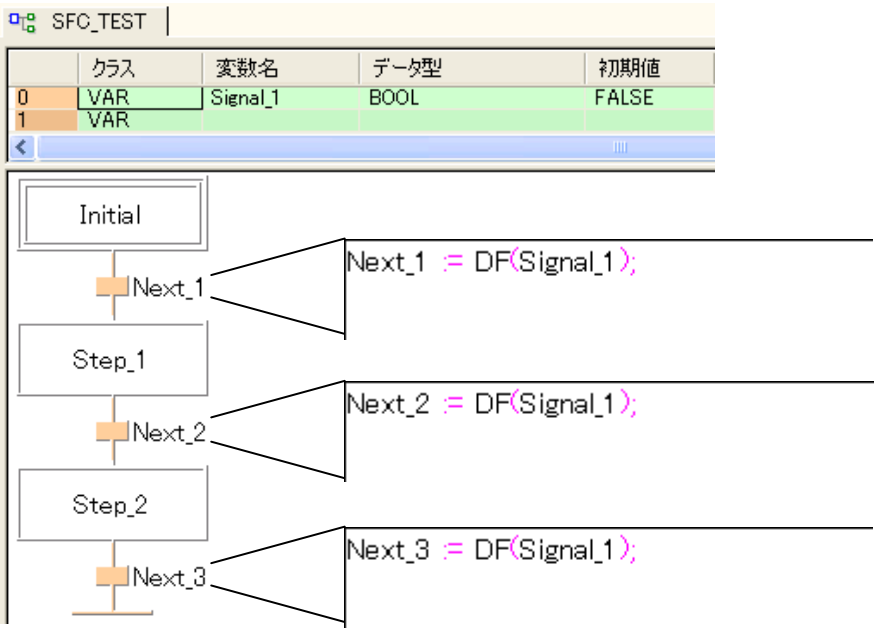
那麼作為應用例、



1 個信号的 ON 可以移動 STEP。

■備考

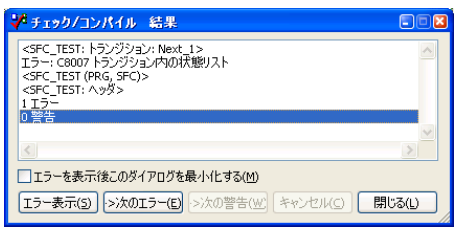
Transition 程式用 ST 語言的寫法範例



●注意!

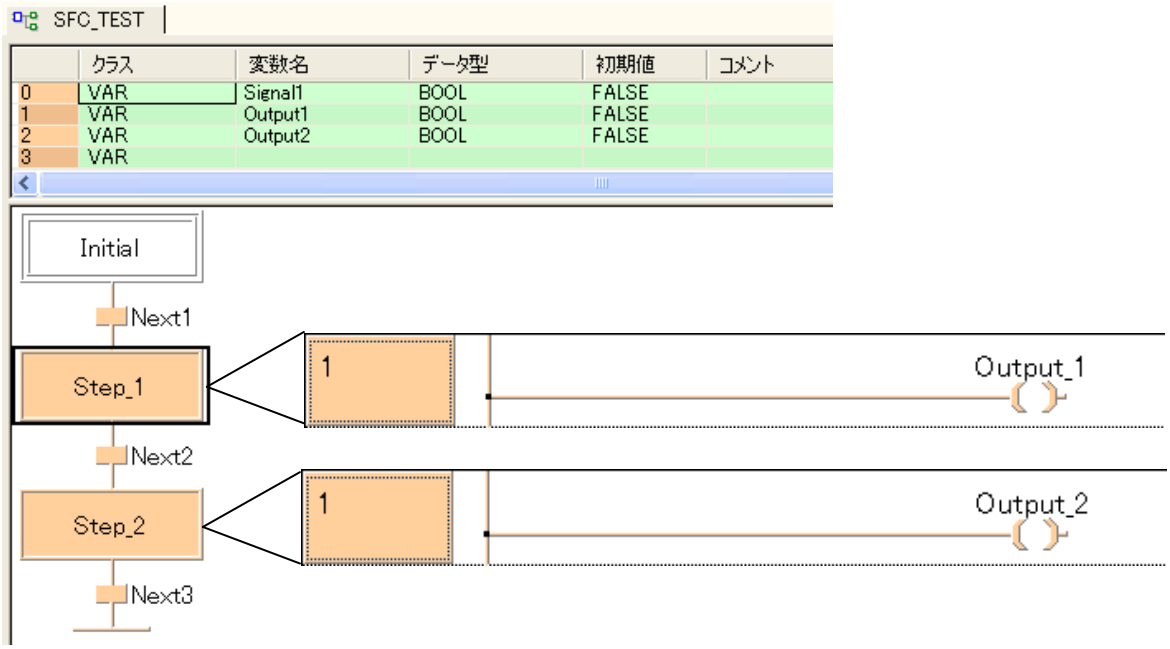
請注意在 Transition 程式內
不能使用 IF 文句。

```
IF DF(Signal_1) THEN
   Next_1 := 1;
END_IF;
```

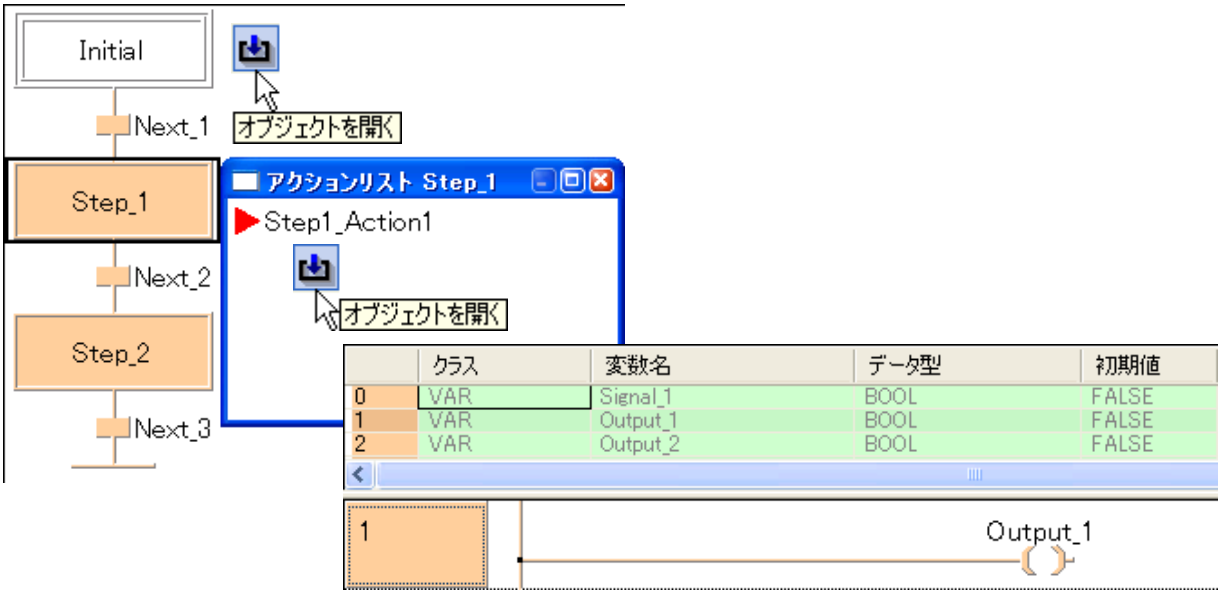


12-5 STEP FLAG 動作

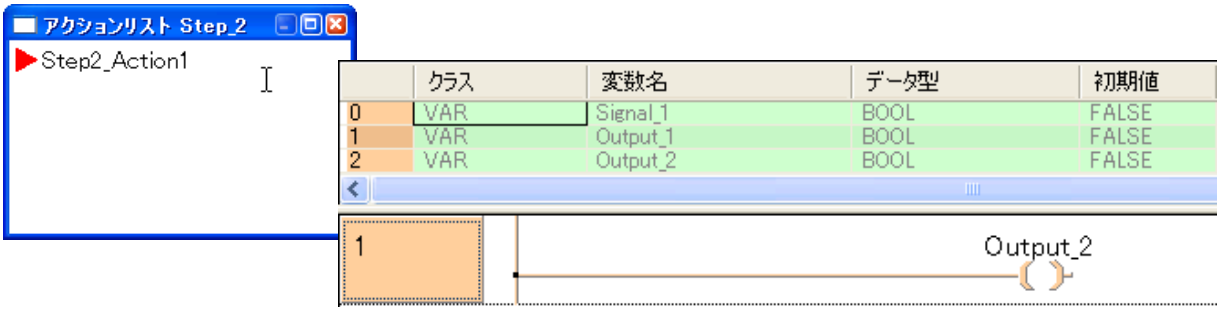
在前項的 STEP、製作以下簡單的 Action。



在此 Action 名稱為「Step_1」作為“Step_1_Action1”製作 Action。



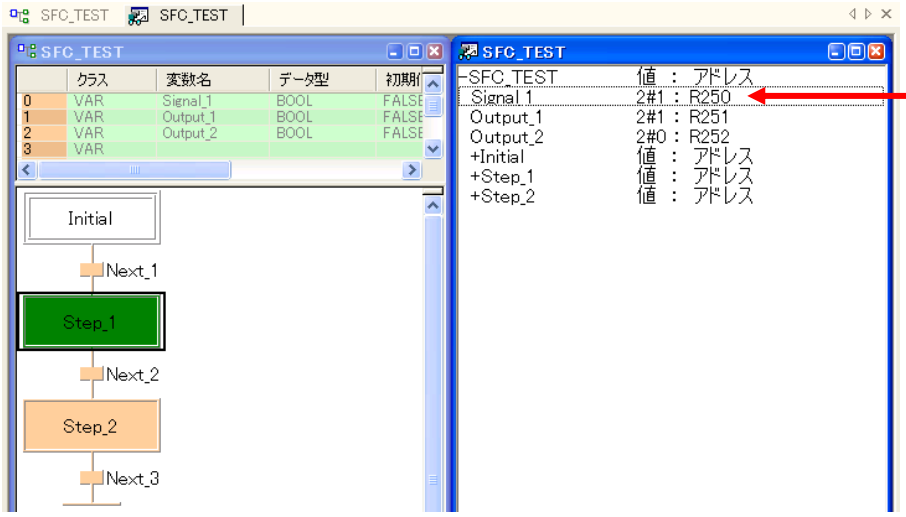
Action 名稱為「Step_2」作為“Step2_Action1”同樣的製作 Action。



■動作確認

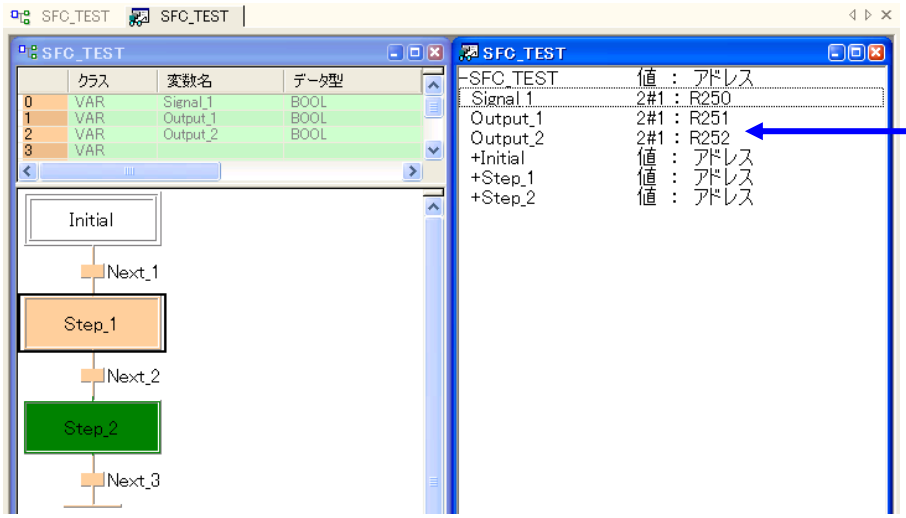
那麼依照移動的順序來看看。

STEP 移動到「Step_1」。(“Signal_1”為 ON)



移到「Step_1」的同時
“Output_1”會 ON。

STEP 移動到「Step_2」。(“Signal_1”為 OFF/ON)



“Output_2”為 ON。
而且在「Step_1」為 ON
“Output_1”會保持狀態。

※ 在 SFC 會從現在的 STEP 到接著的 Transition 條件成立(TRUE)、移動到下個 STEP 的時候、會保持現在的 STEP 輸出状態、即使移動到下個 STEP 也會被保持。

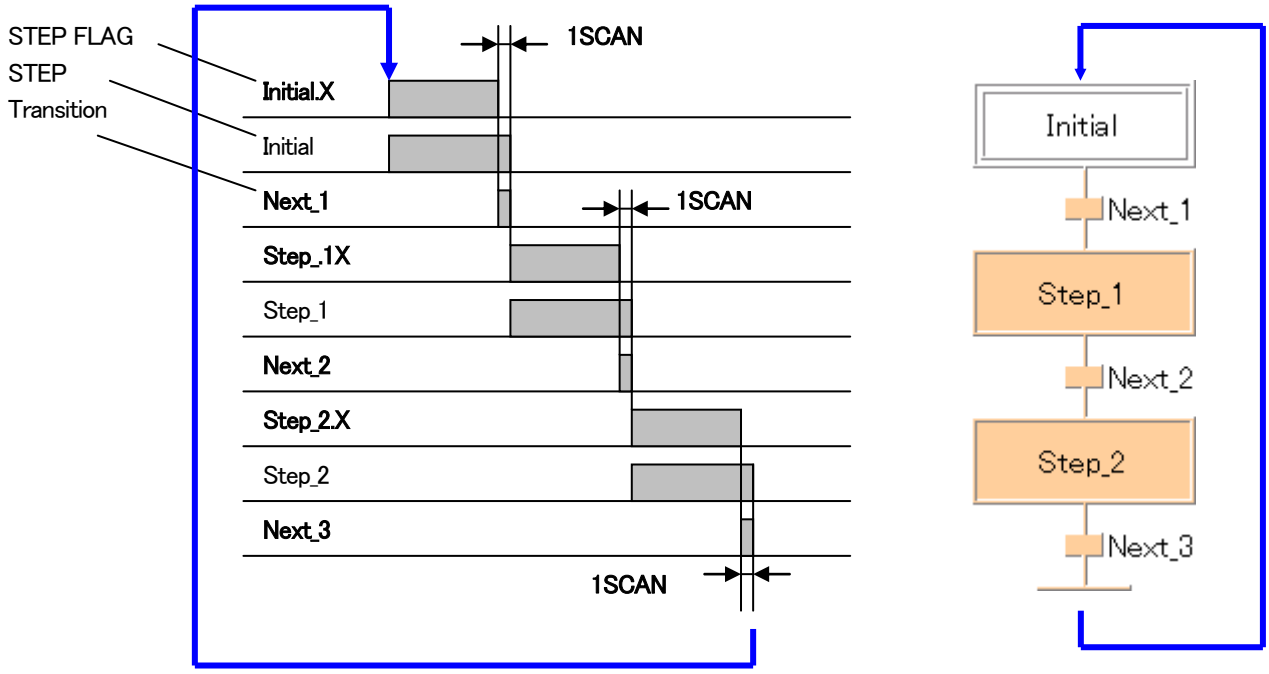
那麼移動時的現在輸出状態為 OFF(FALSE)時該如何?

可以用 SFC 語言所準備的 STEP FLAG。

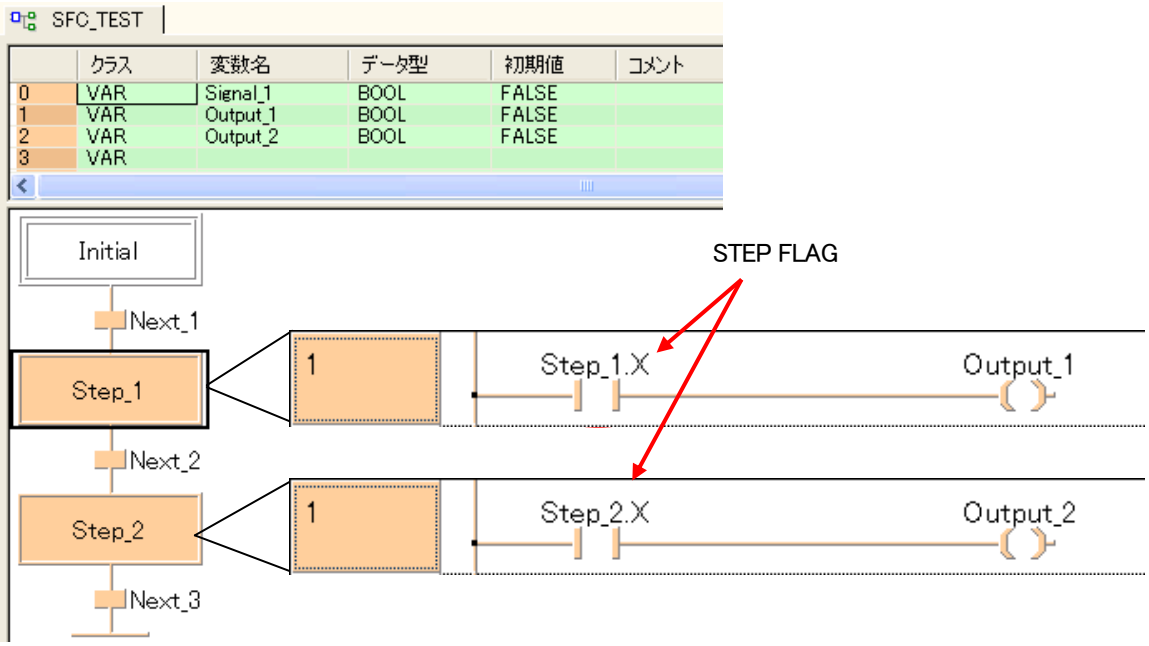
編譯器對每個 SFC 的 STEP、會將 BOOL 型的 STEP FLAG 做自動分配。
STEP FLAG 以 STEP 的狀態工作中(TRUE)，或是停止中(FALSE)來做顯示。
本例如 STEP 的名稱 Step_1.X, Step_2.X、STEP 名稱和添加字“X”來表示。

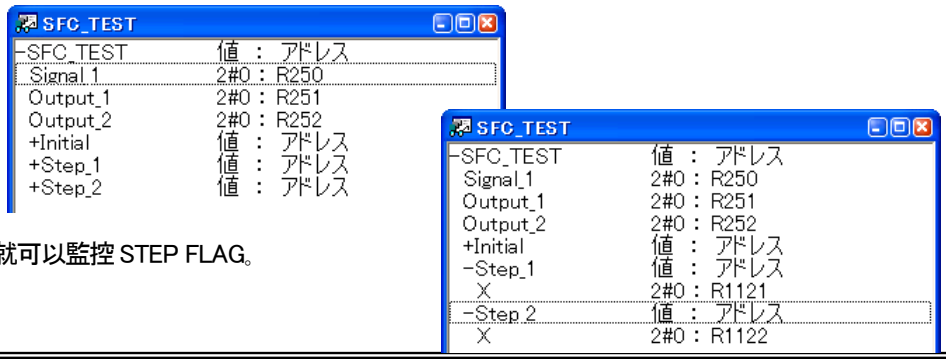
STEP(Step_1)起動、則對應的 STEP FLAG(Step_1.X)會設定為 TRUE。
STEP(Step_1)接下來的 Transition 的條件(Next_1)成立、STEP FLAG(Step_1.X)雖然會被設定為 FALSE、
現在的 STEP(Step_1)會從還有 1 個 Scan 動作開始往下個 STEP(Step_2)移動。
因此各 STEP 經常最少也會進行 2 次以上的執行。

■STEP FLAG 的時間變化



那麼用 STEP FLAG 如下圖般變更各 STEP 的 Actionr 進行動作的確認。



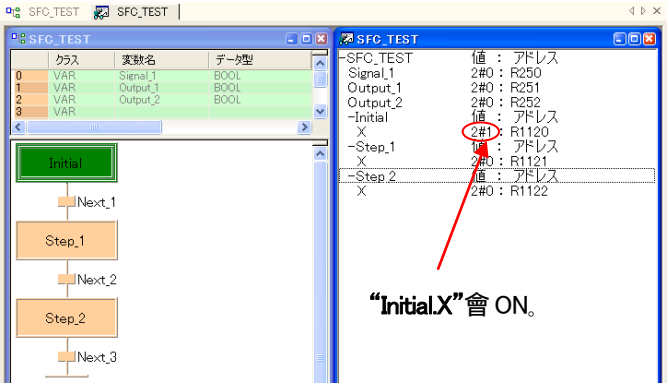


在 Header 監控、
STEP 名稱點擊兩次就可以監控 STEP FLAG。

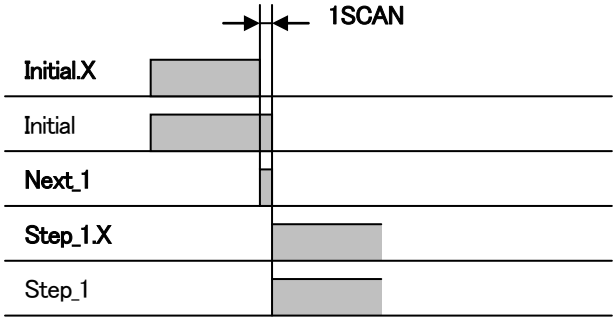
■動作確認

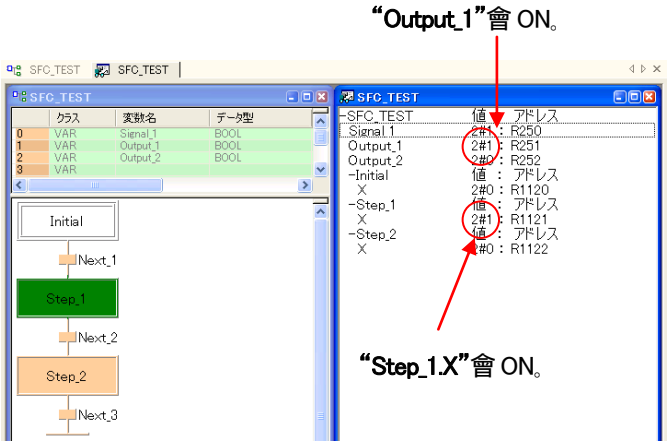
那麼依照移動的順序來看看。

①變成 RUN 模式、啓動「Initial」STEP。

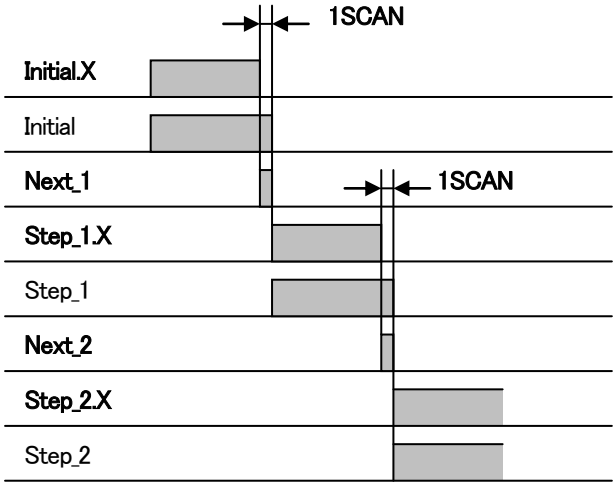


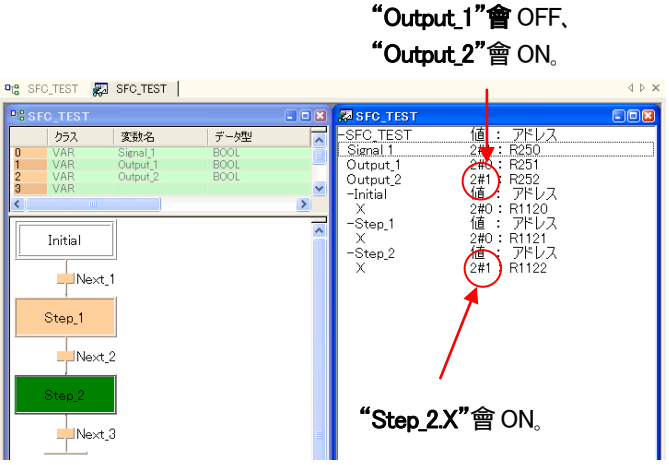
②STEP 移動到「Step_1」。("Signal_1"會 ON)





③STEP 移動到「Step_2」。("Signal_1"為 OFF/ON)

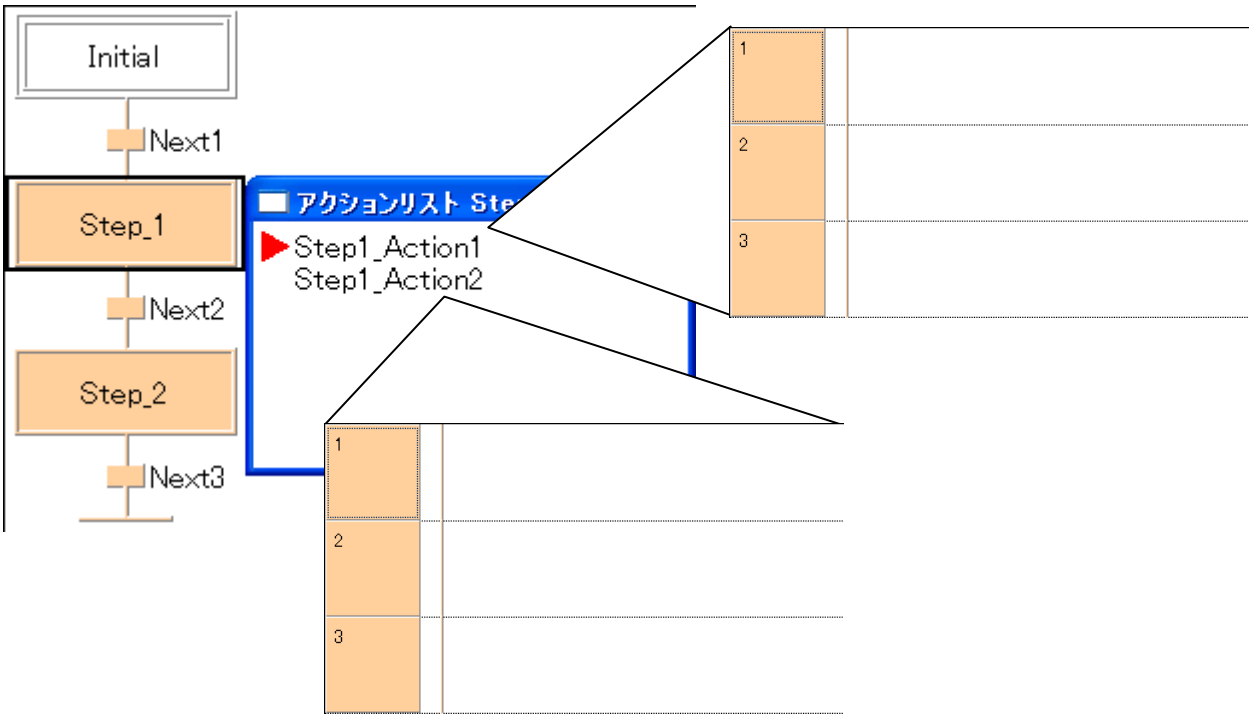




通過使用 STEP FLAG、現在的 STEP 所接下來的 Transition 成立時、
在下個 STEP 起動時、現在 STEP 的 STEP 輸出可以 OFF。

■備考

而且作為程式的 Action 可以複數寫入。



12-6 Action 動作

STEP 起動時、所謂作為程式的 Action 動作另外可以將 BOOL 型的變數作為 Action 寫入。

●Action 的動作記號

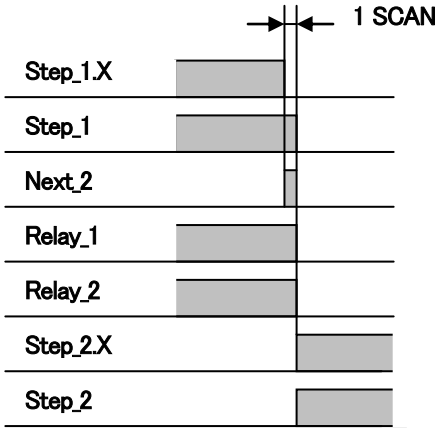
在 Action 宣告 BOOL 型變數時、在 Action 連結 List 使用下一個動作記號、
可以設定 BOOL 型變數的動作。

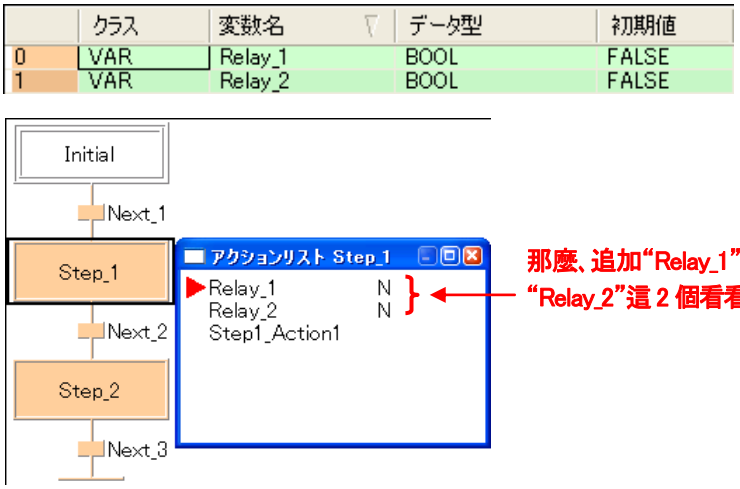
動作記號	內容
N	非保持
R	RESET(OFF 保持)
S	SET(ON 保持)
P	Pulse

12-6-1 動作記号 N

BOOL 型變數被分配到動作記號 N（非保持 Action）、則到移動到下個 STEP 時、FALSE 會被 RESET。
即是無法保持 N 記述 BOOL 型變數現在值。

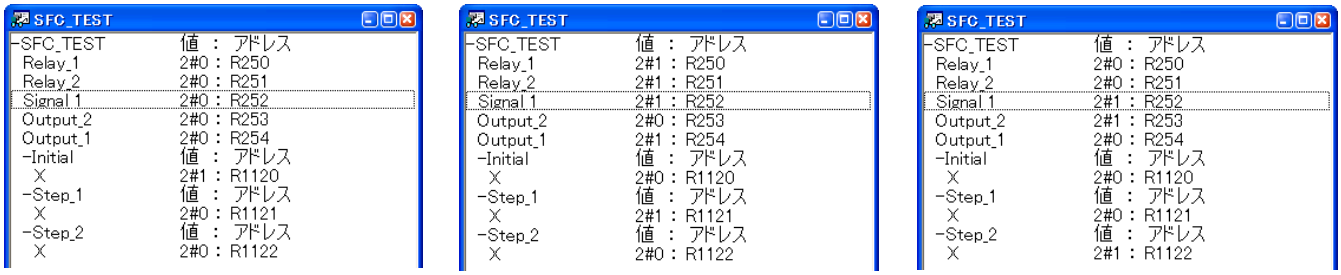
那麼到前項為止的程式「Step_1」、追加作為變數的 Action 看看。





●動作確認

「Initial」STEP → 「Step_1」STEP → 「Step_2」STEP



“Relay_1”和“Relay_2”會 ON　　“Relay_1”和“Relay_2”會 OFF

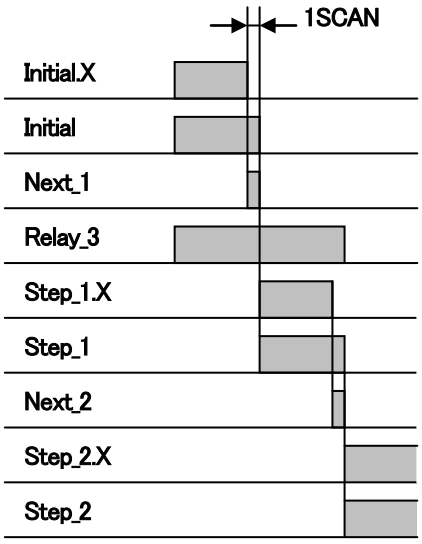
●注意

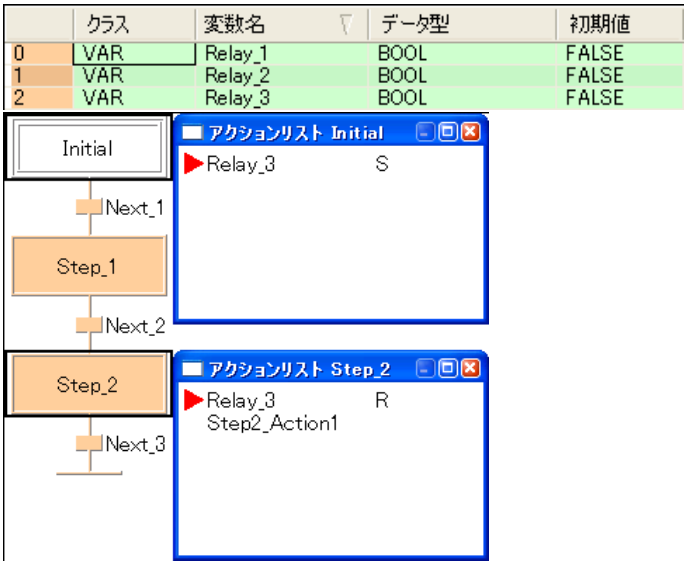
特別是沒有記述 Action 動作記號時、會被自動的認為是 N。

12-6-2 動作記號 R 和 S

分配到 BOOL 型變數的動作記號 R(RESET)、STEP 起動則 FALSE 立刻會被 SET。
分配動作記號 S(SET)、STEP 起動則 TRUE 立刻會被 SET。
即使移動到下個 STEP、變數值不會被變更、就這樣保持著。

那麼到前項為止的程式「Initial」和「Step_2」、將作為變數的 Action 追加看看。

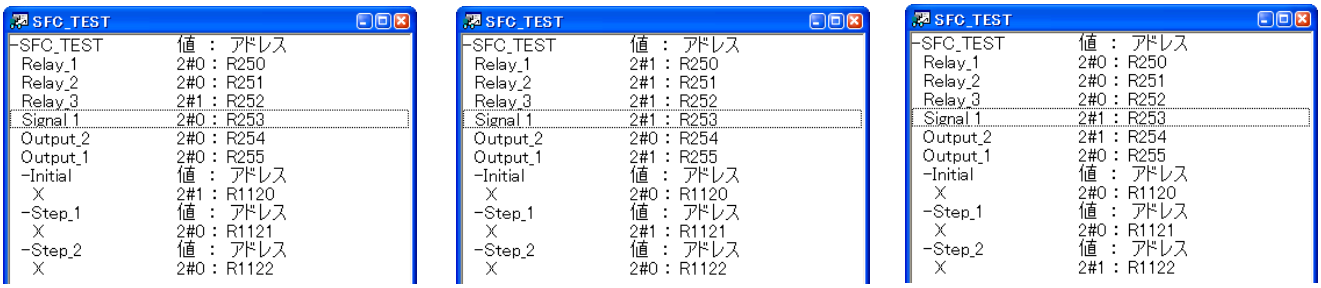




在「Initial」STEP、會將作為 Action 動作記號 S 分配到 BOOL 變數“Relay_3”。
「Initial」STEP 起動的同時、變數“Relay_3”的 TRUE 會被 SET。
此状態為在「Step_2」STEP 內、直到 RESET 為止、會被一直保持下去。

●動作確認

「Initial」STEP → 「Step_1」STEP → 「Step_2」STEP

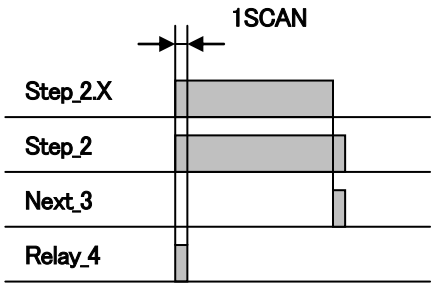


“Relay_3”會 ON　“Relay_3”繼續 ON　“Relay_3”會 OFF

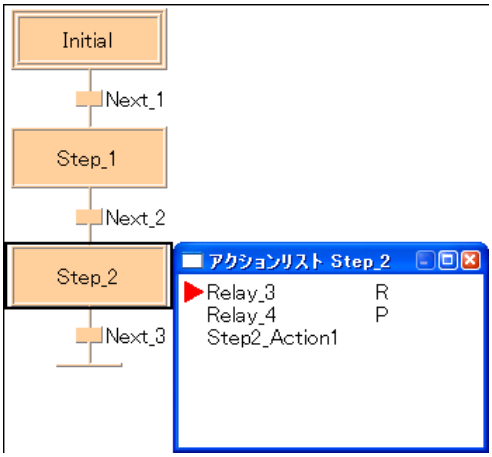
12-6-3 動作記號 P

動作記號 P(Pulse)分配到 BOOL 型變數、在 STEP 起動的同時
只有 PLC 的 1SCAN 之間、TRUE 會被 SET。

那麼到前項為止的程式「Step_2」、作為變數的 Action 也追加看看。

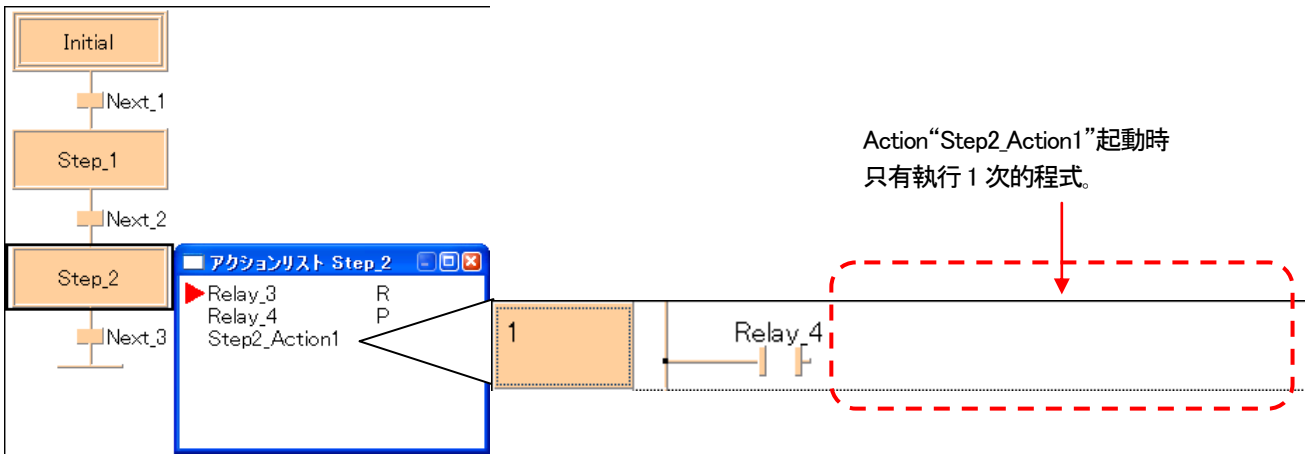


	クラス	変数名	データ型	初期値
0	VAR	Relay_1	BOOL	FALSE
1	VAR	Relay_2	BOOL	FALSE
2	VAR	Relay_3	BOOL	FALSE
3	VAR	Relay_4	BOOL	FALSE



「Step_2」會把 P 作為 Action 動作記號分配到 BOOL 型變數"Relay_4"。
「Step_2」起動的同時、變數"Relay_4"的 TRUE 會被 SET、1SCAN 掃描後則 FALSE 會被 SET。
之後、「Step_2」執行中、變數"Relay_4"則保持 FALSE。

通過如"Relay_4"這般的 1SCAN ON Pulse、
可以做出「Step_2」的 Action 起動時只有執行 1 次的程式。

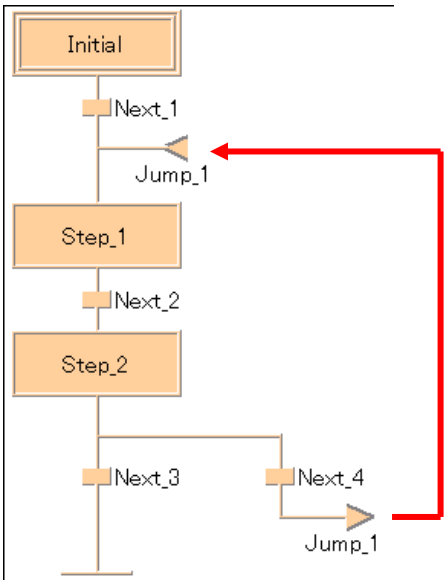


■備考

如下圖通過使用"sys_bIsFirstScanOfSfcStep"同樣可以做出來。

12-7 JUMP 和 Label

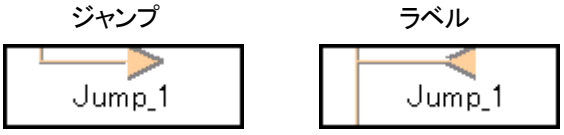
利用 Transition 條件來說明在 SFC 內的 JUMP 方法。



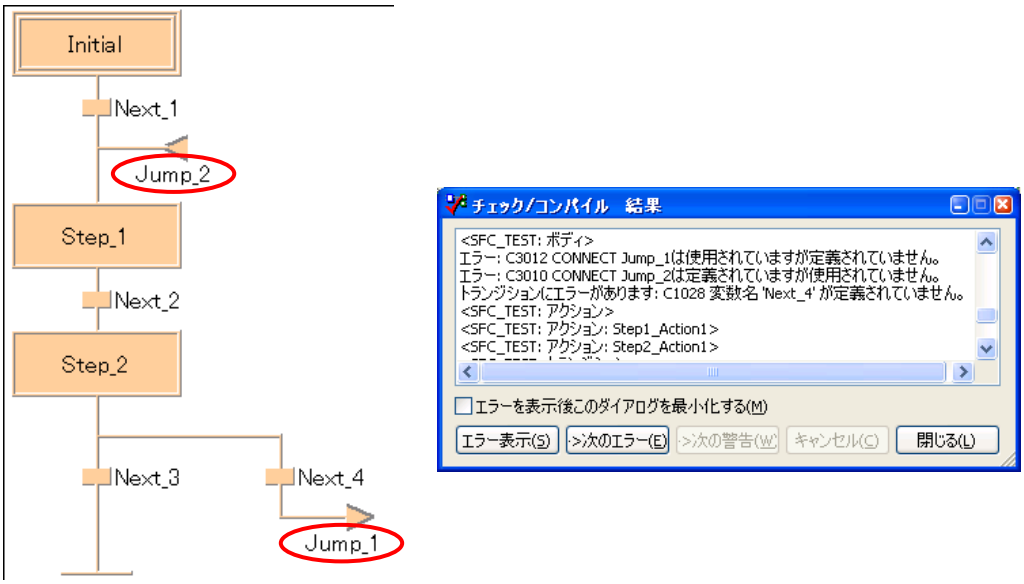
如果 Transition“Next_4”為 TRUE 的話
則 JUMP 到「Step_1」STEP。

上圖為對於到前項為止的 SFC、
「Step_2」STE 之後的 Transition“Next_4”如果是 TRUE 則 JUMP 到「Step_1」STEP
以上為追加流程範例。

為了 JUMP 需寫上「JUMP」記號和要到 JUMP 的地方「Label」記號。
「JUMP」和「Label」要用相同名稱進行動作。



例如、「JUMP」名稱和「Label」名稱不同時、編譯會發生 ERROR。



名稱相同的「JUMP」和「Label」不存在時也會發生編譯 ERROR。

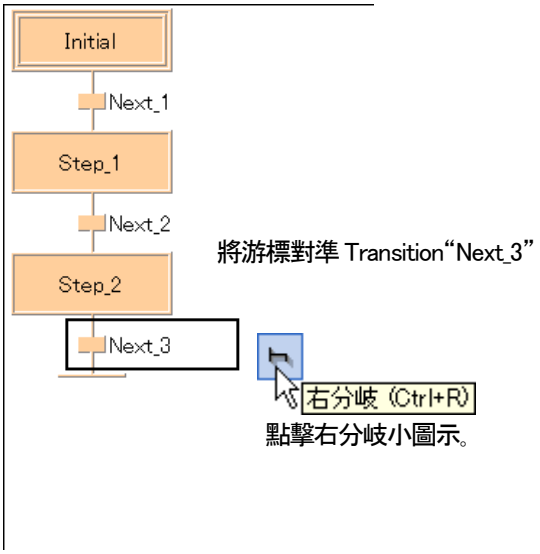
那麼將「JUMP」和「Label」追加到前項 SFC 看看。

首先宣告「JUMP」的 Transition 為“TRUE”的 BOOL 型變數。
在此稱為“Jump_Signal”。

	クラス	変数名	データ型	初期値
0	VAR	Relay_1	BOOL	FALSE
1	VAR	Relay_2	BOOL	FALSE
2	VAR	Relay_3	BOOL	FALSE
3	VAR	Relay_4	BOOL	FALSE
4	VAR	Jump_Signal	BOOL	FALSE
5	VAR	Signal_1	BOOL	FALSE
6	VAR	Output_2	BOOL	FALSE
7	VAR	Output_1	BOOL	FALSE

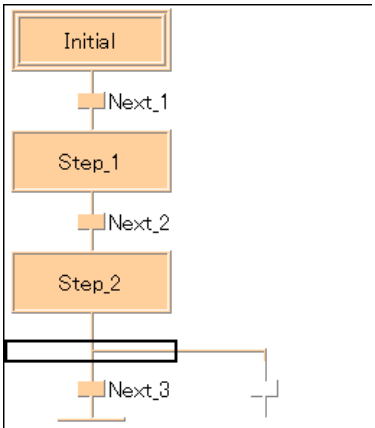
追加

位於「Step_2」之後用並列分岐製作 Transition“Next_4”。



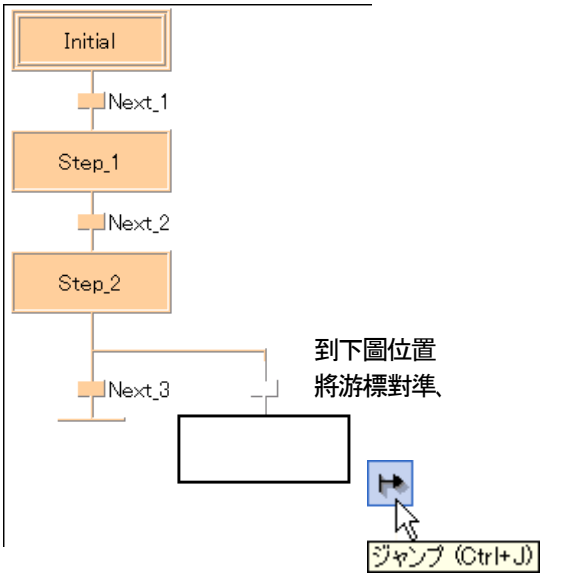
將游標對準 Transition“Next_3”

點擊右分岐小圖示。



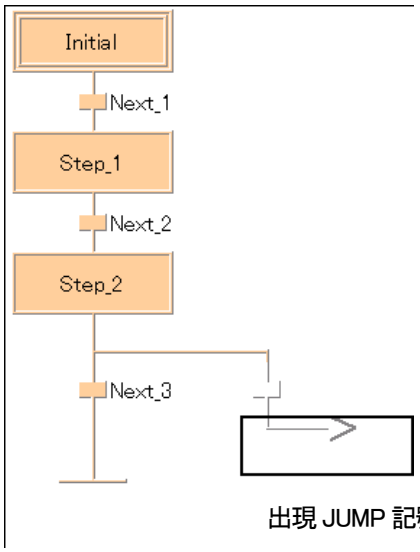
出現並列分岐的 Transition。

製作“JUMP”。



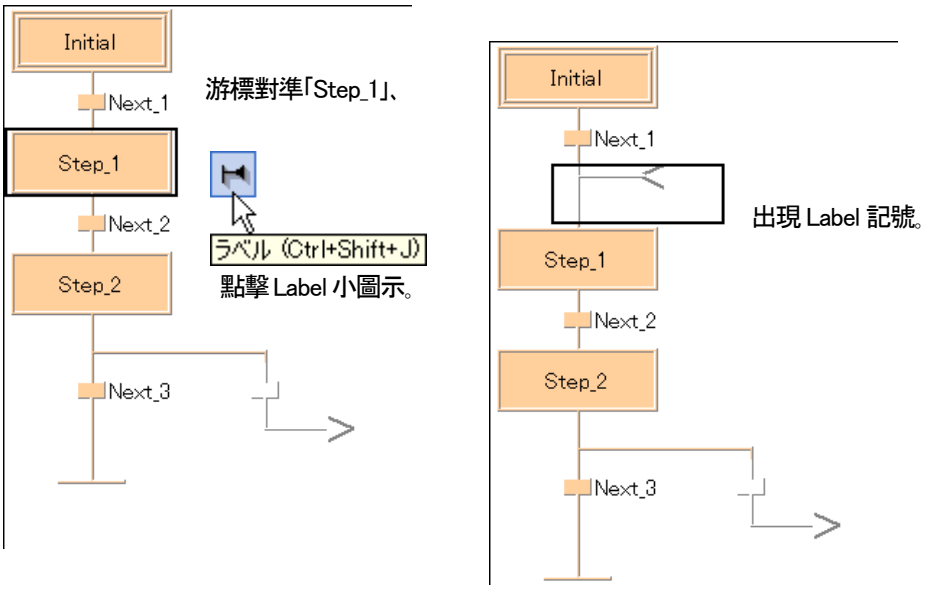
到下圖位置
將游標對準、

點擊 JUMP 小圖示。

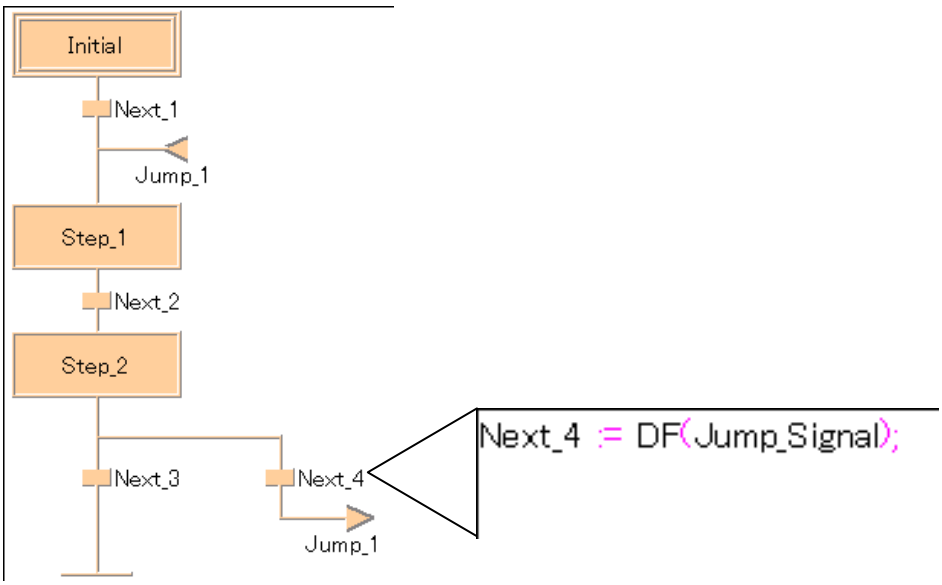


出現 JUMP 記號。

製作“Label”。



將「JUMP」和「Label」名稱寫為 Action 到 Transition 則完成。



那麼確認動作一次看看。

12-8 編集視窗的分割

使用編集視窗的捲軸列和標題列所挾的小黑列、可以將編集視窗分割為上下。
將一個程式的內容分成 2 個畫面、可以個別表示捲軸，監控。

■操作步驟

1. 將游標對準被編集視窗的標題列和捲軸列夾著的小黑色的 BAR。

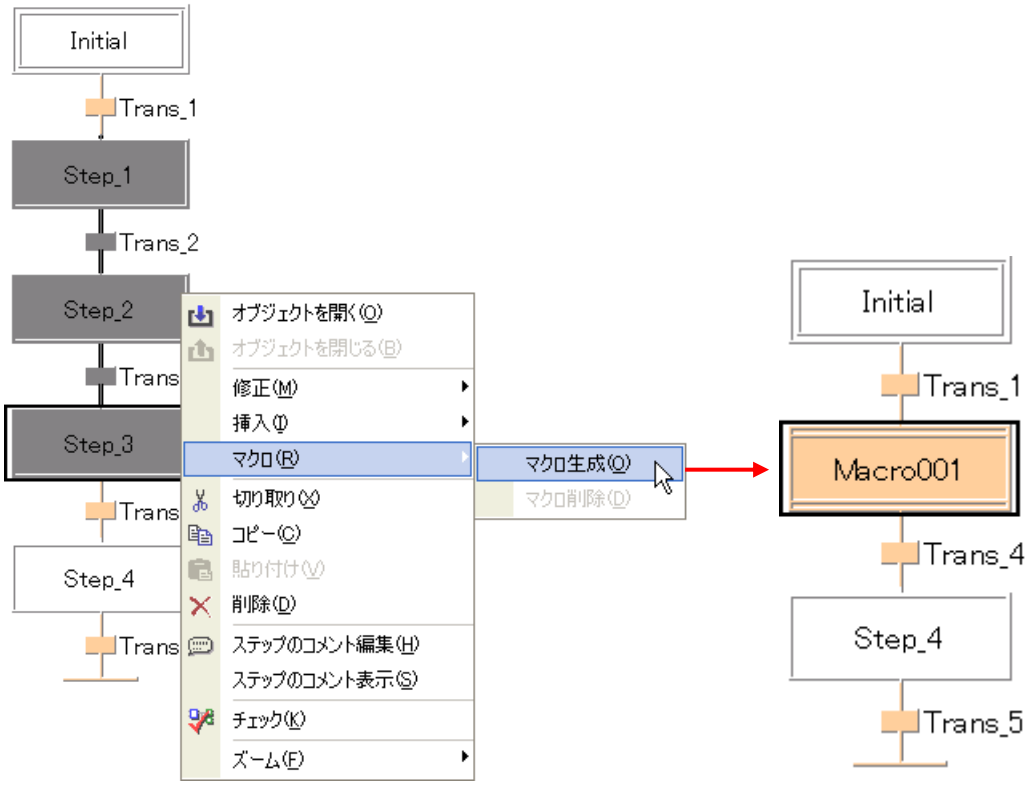
2. 按著滑鼠左鍵、將 BAR 拖曳到任意的位置。

12-9 Macro STEP

將複數的 STEP、Transition 整合為一個(Macro STEP)、如此流程會變的簡單明瞭。

■操作步驟

選擇要整合為一個的複數 STEP、Transition(按著 Shift 鍵點選各個 STEP)、
點右鍵選擇→「Marco」→「Marco 産生」。



12-9-1 Marco 的顯示

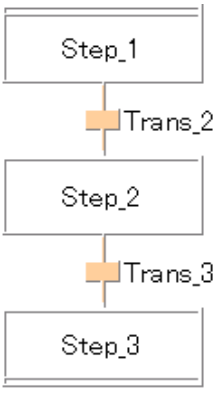
點選(選單)工具 → Marco 構成 List、打開對話框、用 POU 選擇被定義的全部 Marco 做一覽顯示。
從一覽選擇打開編集 Marco。

■操作步驟:打開 Marco

1. **點選位於 POU Body 的任一個 Marco STEP。**
2. **點選 (選單)工具 → Marco 構成 List、或是**  **。**
   打開 Marco 構成 List、用所選擇的 POU 顯示被定義 Marco 的全部。

3. **點任意的 Marco 名稱。**
4. **點 OK 按鍵。**

   將被選擇的 Marco 的內容顯示到編集視窗。
   可以將 STEP, Transition,分岐插入到想被顯示的地方。



5. **點選任意的 STEP。**
6. **關閉(選單)工具 → 打開 Object 或是點**  **。**
   回到原來的 SFC 編集視窗。

第13章

監控功能

13-1 概要

程式編譯並下載之後、要監控其動作有幾個方法。
在監控模式下、可以變更變數值, 顯示經過值, 設定(強制輸出入等)。
使用 Control FPWIN Pro 可以進行以下各種類型的監控。

・User 監控
・Header 監控
・程式監控
・控制監控

更可以支援以下所示的状態監控。

・PLC 状態
・PC Link 状態(MEWNET-P/W)
・Network 的状態(MEWNET-P)
・特殊内部 Relay
・特殊資料暫存
・共通記憶體

●注意

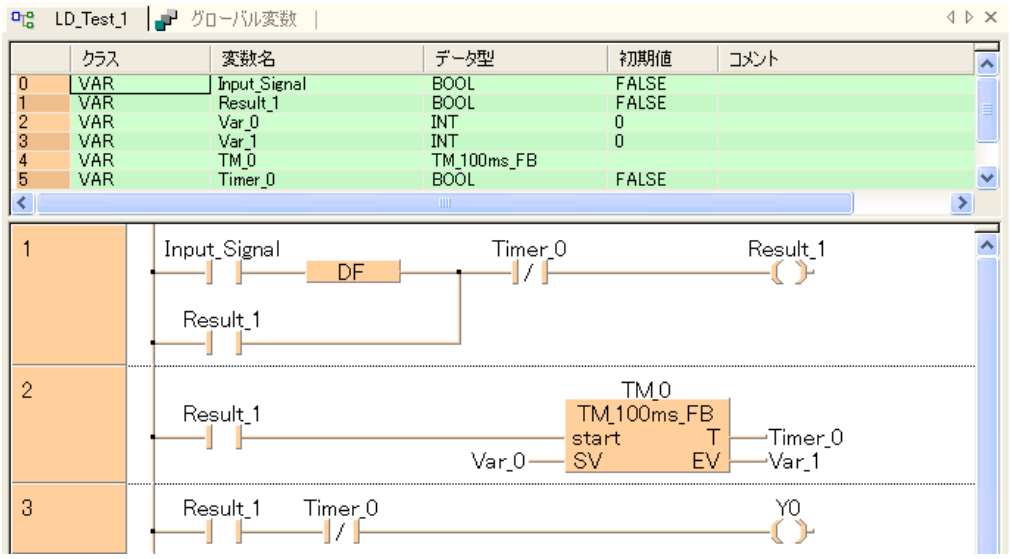
監控必須在 Online 模式下進行。

13-2 資料監控・強制輸出入

13-2-1 資料監控

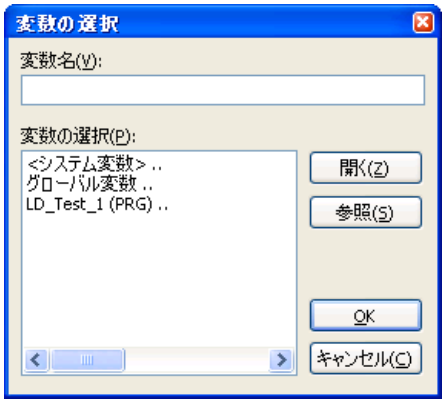
點選(選單)監控 → USER 監控、可以顯示 PLC 程式在工作中的變數值。
依據需要、可以進行變更變數值和強制設定。

使用第 7 章所作成的 POU、實際進行資料的監控。



■操作步驟

1. (選單)Online → Online 模式、或是點 。
2. (選單)Online → PLC 動作模式變更、或是點 RUN 、PLC 切換到 RUN 模式。
3. (選單)監控 →USER 監控、或是點 。
   打開以下的對話框。

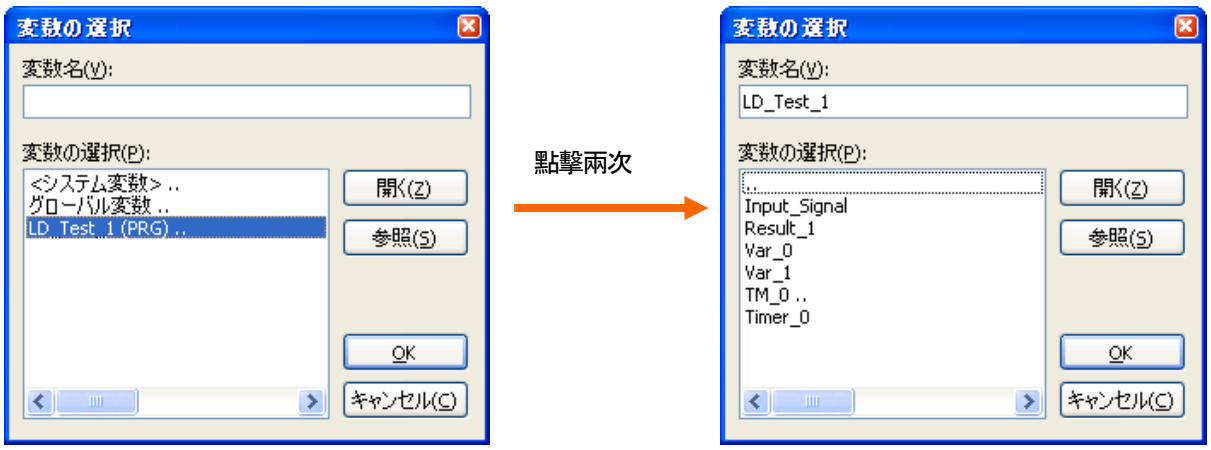


4. 點選「全域變數」、或是點選想顯示的 POU 名稱。

5. 點選任意的 Component 兩次。

顯示所選擇的 Component 變數名稱。

在此選擇“LD_Test_1”。



6. 選擇任意的變數(可以複數選擇)。

之後所表示的東西請直接輸入到「變數」區域。

POU 名

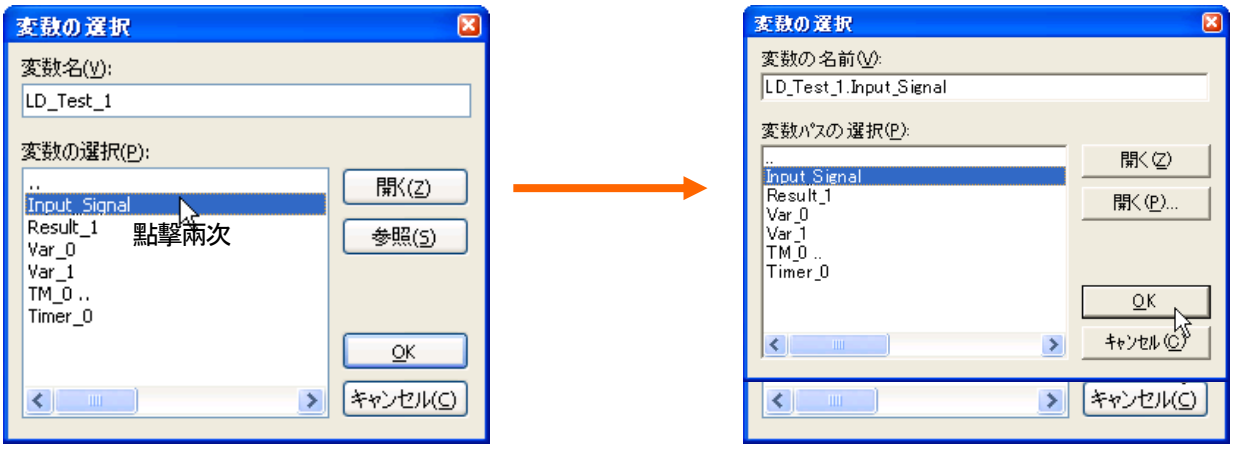
全域變數:

- 全域變數 List 的變數名稱(例 ： Key_1)
- PLC 位址(例 ： X0, Y2)
- IEC 位址(例 ： %IX0.0)

區域變數:

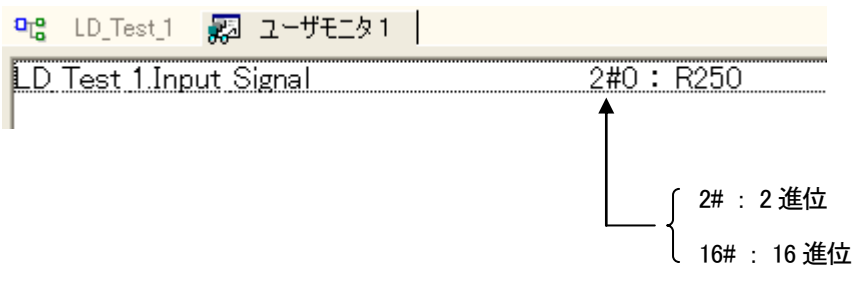
- POU 名稱. 變數名稱(依照情況有 POU 名稱. FB 名稱. 變數名稱)

登録“Input_Signal”。

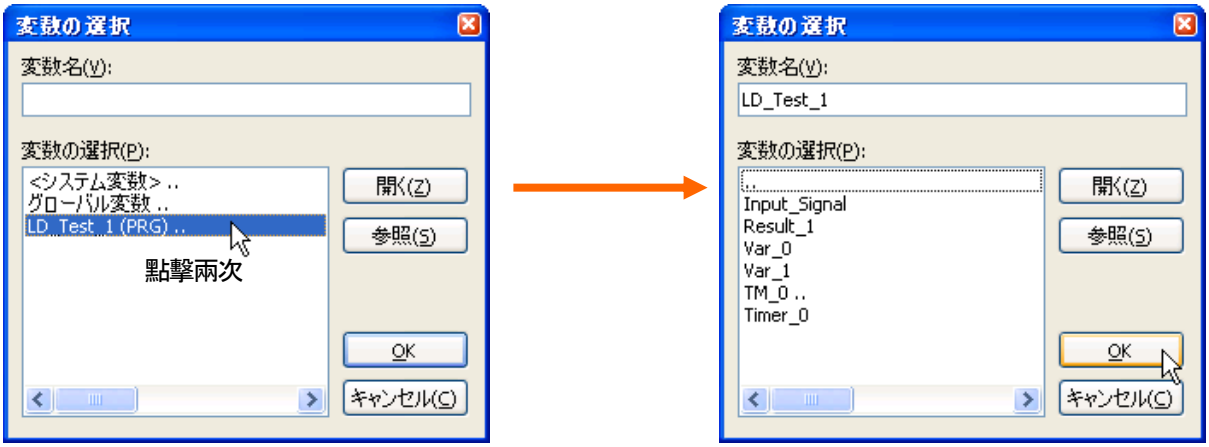


點 OK 鍵。

監控畫面如以下所示。

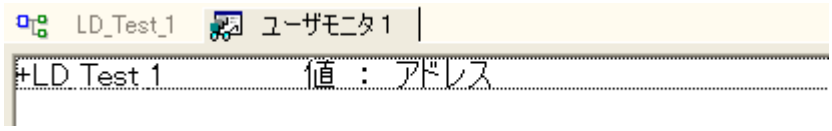


輸入 POU 名稱到「變數名稱」後、在 POU 的全部變數都可以監控。

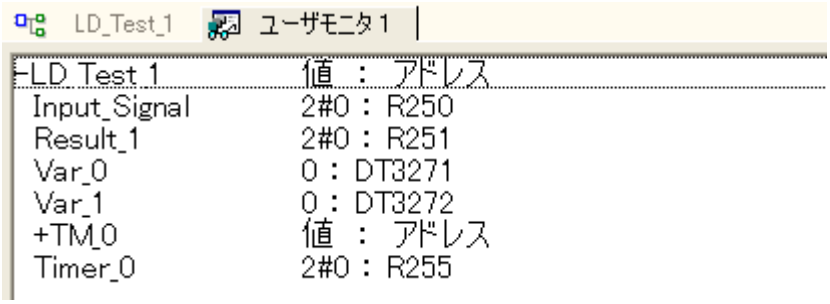


點 OK 鍵。

監控畫面如下所示。

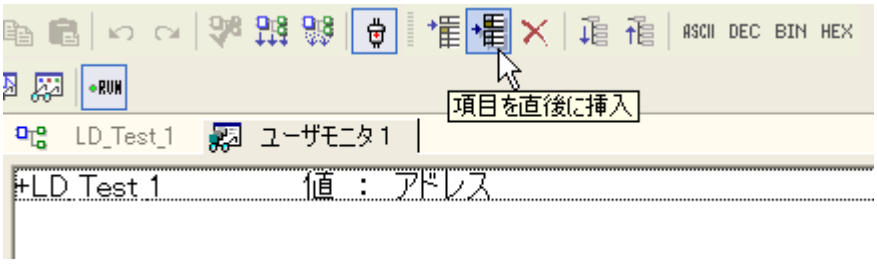


點擊 POU 名稱兩次、則可以顯示全部變數的內容。

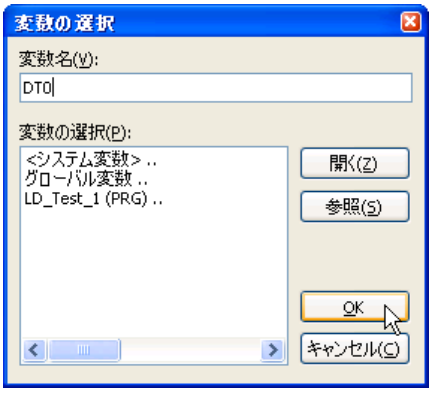


7. 直接登録元件 No.。

(選單)編集 → 追加項目 → 之前／之後、或是點 或是點 。

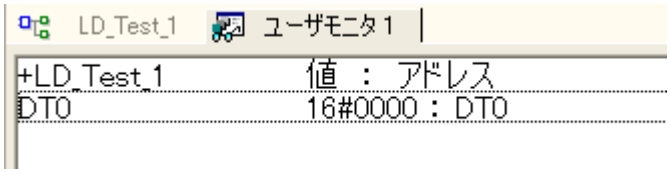


出現「變數的選擇」對話框、請輸入“DT0”。



點 OK 鍵。

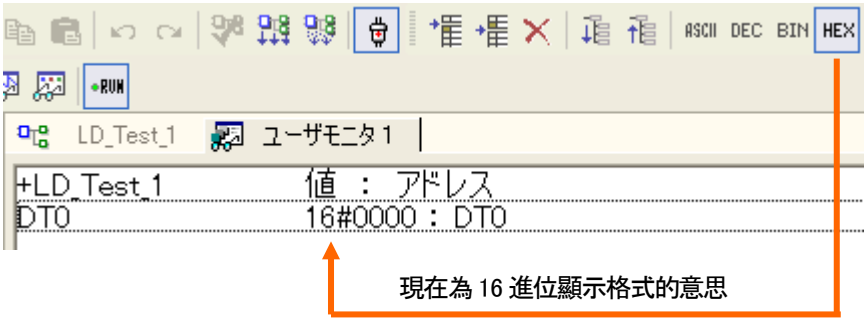
監控畫面如以下所示。



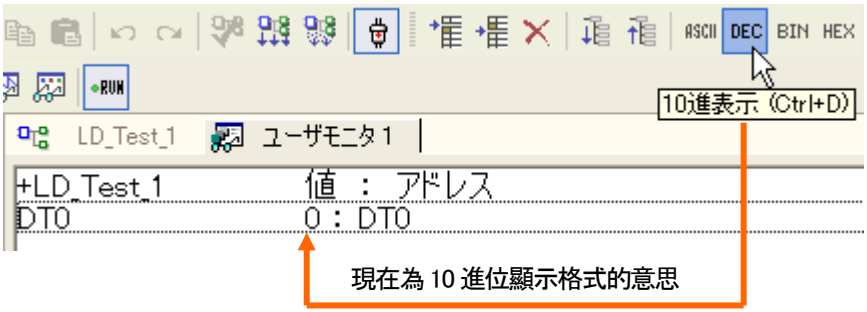
■變數值的顯示格式的變更

■操作步驟

1. **在資料監控的視窗裡、點選想更改顯示格式的地方(監控欄)。**
   在此變更“DT0”的顯示格式。



2. **從 16 進位顯示變更為 10 進位顯示。請點** DEC **。**



變更為 ASCII 顯示 → 點 ASCII 。

變更為 10 進位顯示 → 點 DEC 。

變更為 2 進位顯示 → 點 BIN 。

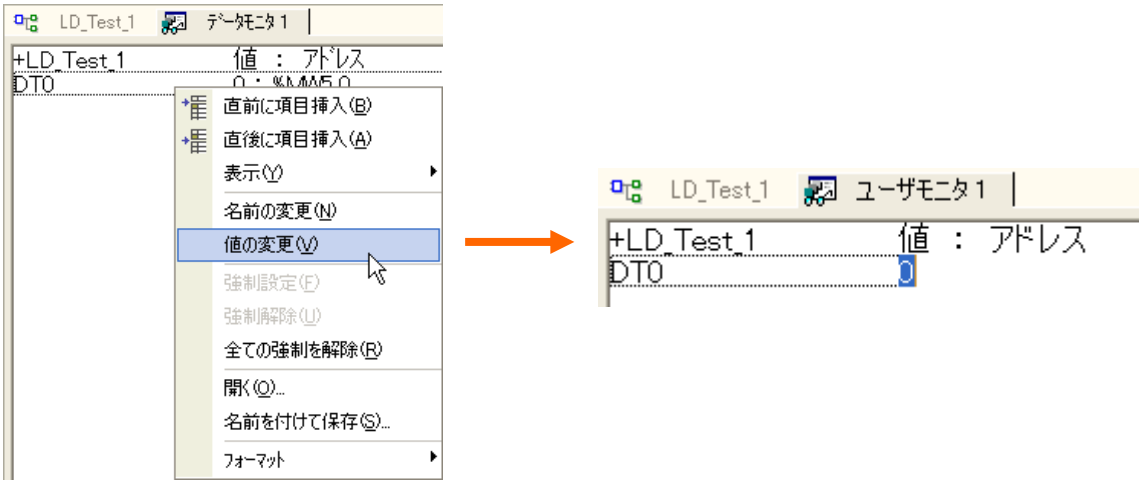
變更 16 進位顯示 → 點 HEX 。

■值的變更

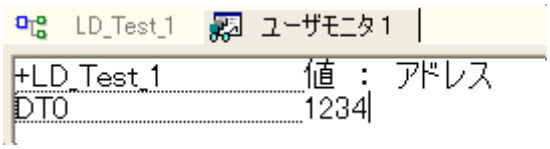
■操作步驟

1. **點選想變更的變數或是元件。**
   在此變更“DT0”的值。

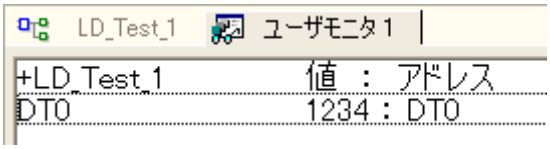
2. **點選右鍵從選單選擇「值的變更」。**



3. **輸入新的值。**
   在此輸入“1234”。



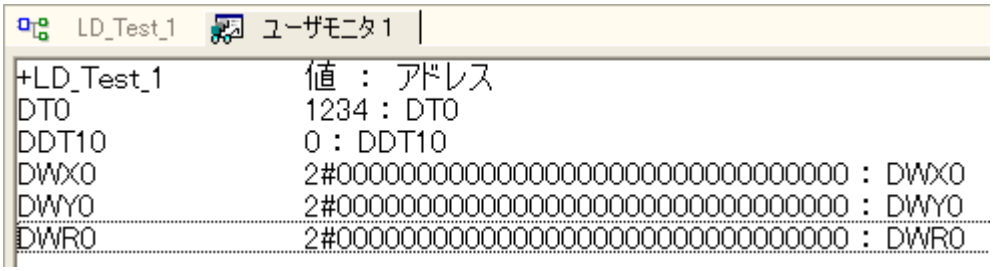
4. **按<Enter>鍵。**



●備註

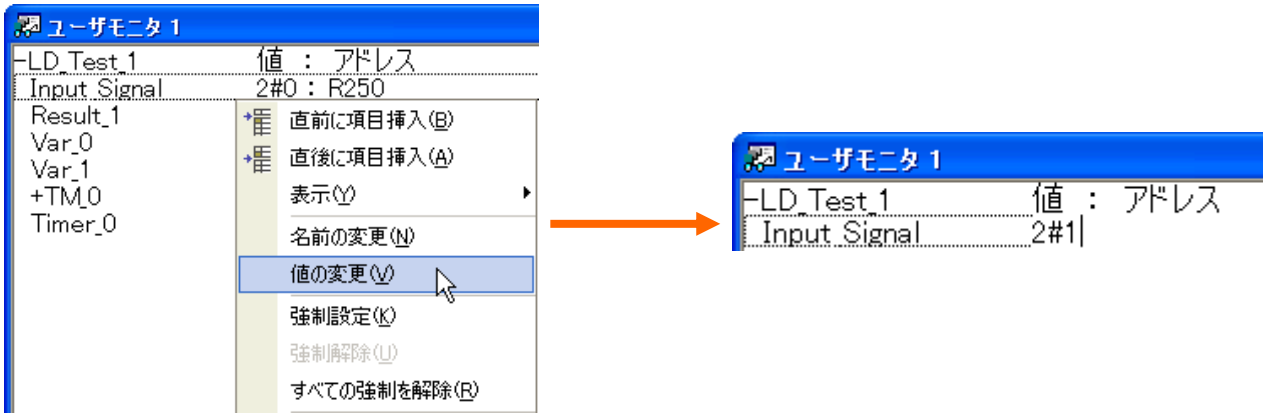
<u>登錄 2 個 WORD 資料時。</u>

元件的前面加上“D”。

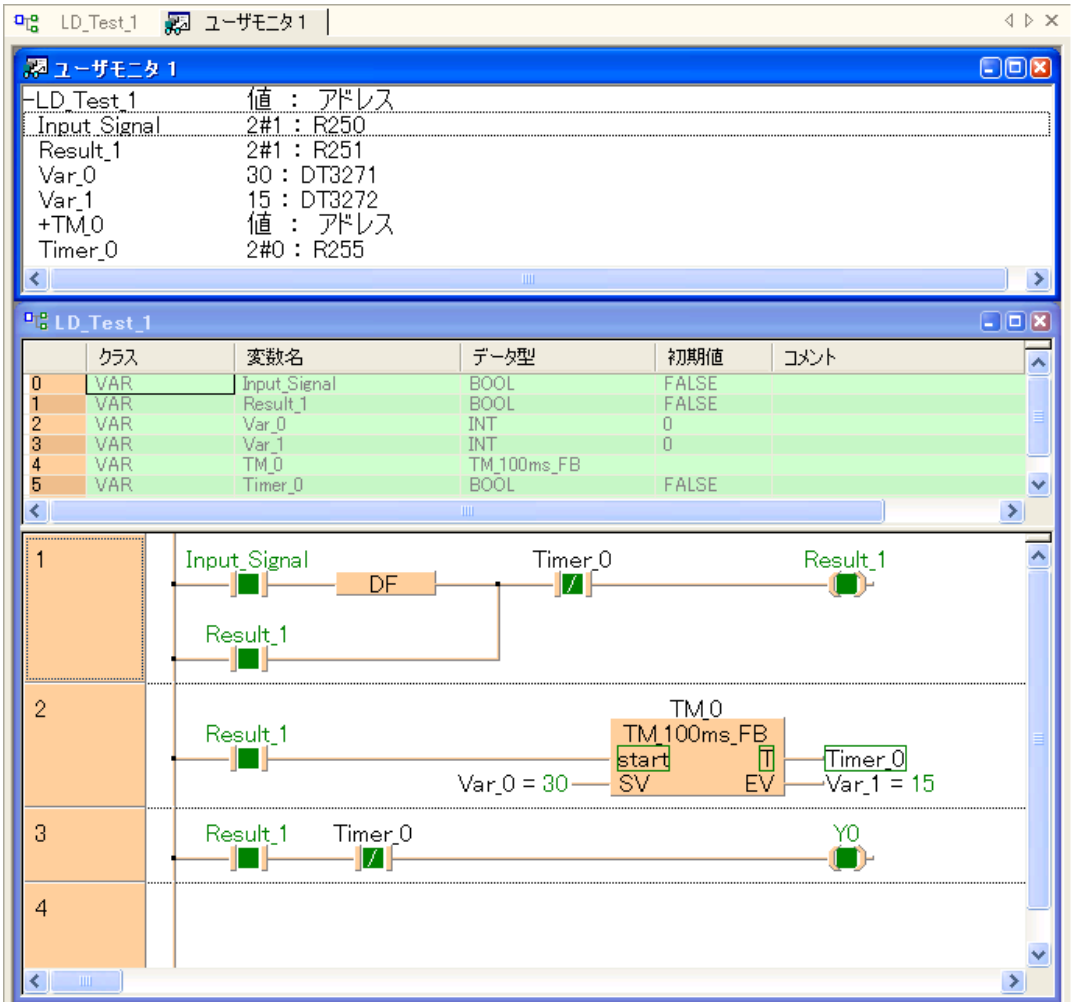


■資料監控和 POU 監控

連動並來看看資料監控和 POU 的程式上的監控。
變數“Var_0”(計時器(0.1S)的設定值)”30”設定之後、“Input_Signal”會 ON。



可以知道 POU 上的監控有連動。

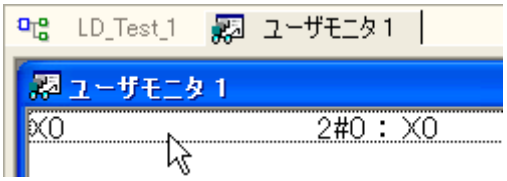


13-2-2 強制輸出入

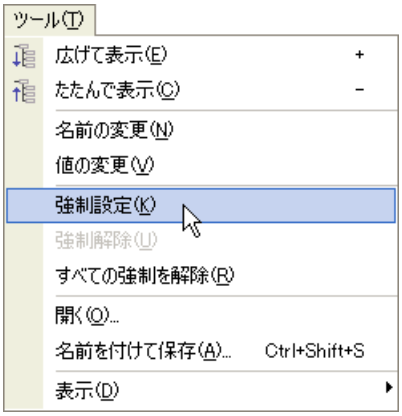
強制輸出入 ：(只有 Boolean 變數可以)

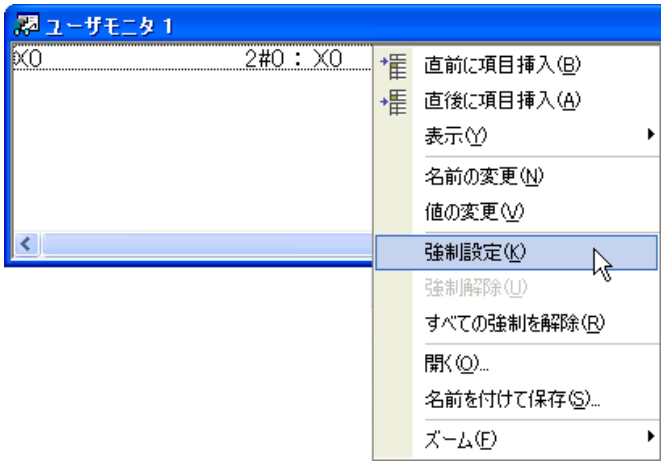
■操作步驟

1. **點選監控中的監控欄。**
   在此點選“X0”。



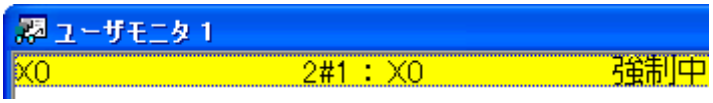
2. **(選單)工具 → 強制設定、或是點右鍵從選單點「強制設定」。**





3. **輸入強制值(0 或是 1)。**

4. **按<Enter>鍵。**
   變數顏色變成指定色。
   要變更已經強制設定的變數值時、請依照前述的變更步驟。



強制解除

■操作步驟

1. **點擊監控中的監控欄。**

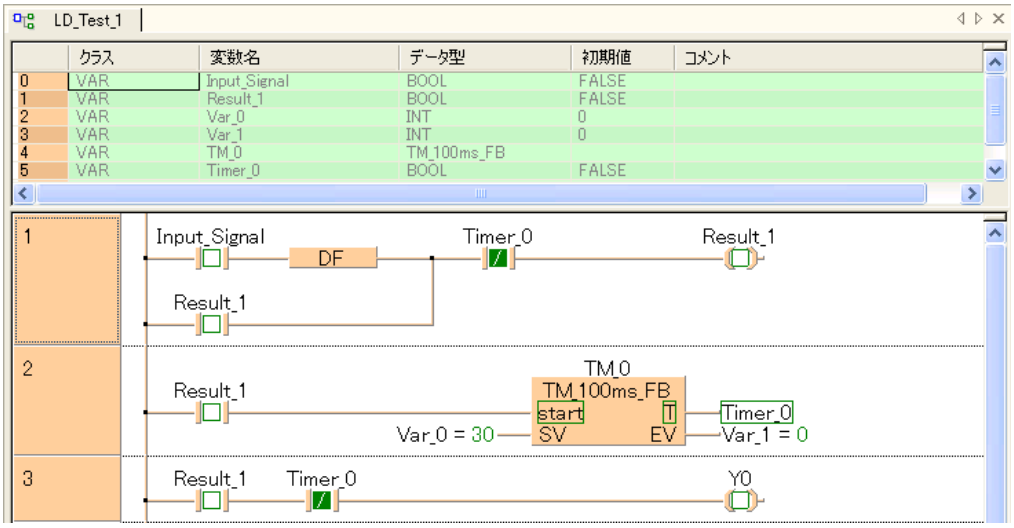
2. **(選單)工具 → 強制解除 、或是按右鍵從選單點「強制解除」。**
   變數的顏色會變為白色。

13-3 POU Header 監控

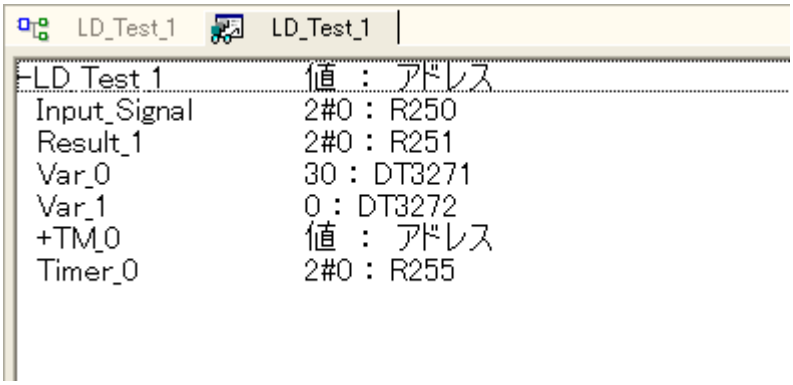
在畫面上打開 POU Body、則可以用對應此 Body 的 POU Header 監控被宣告的變數。

■操作步驟

1. 點擊兩次成為對象的 POU、打開 Header 和 Body。



2. <u>選單(監控) → Header 監控</u> 、或是點  **。**



第14章

USER Library

14-1 概要

可以將常使用的 POU、Function(FUN)、Function Block(FB)、儲存到 USER Library 再利用。
在 Control FPWIN Pro、最多可以做成到 50 種類的 USER Library。
USER Library 可以儲存到硬碟。其他 USER 也可以呼叫出 Library 到 Project File。
修改 Library Object 則可以同步對應全部的 USER。

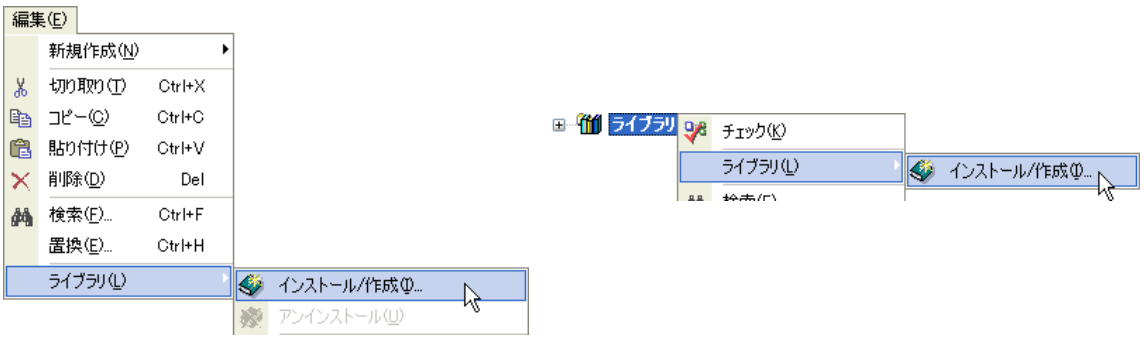
在 USER Library 有「OPEN」「變更」「安裝」等 3 個狀態。
無論那個狀態下、都顯示於 Project Navi。
依照狀態可以進行 Library 的編集和使用。

14-2 USER Library 製作步驟

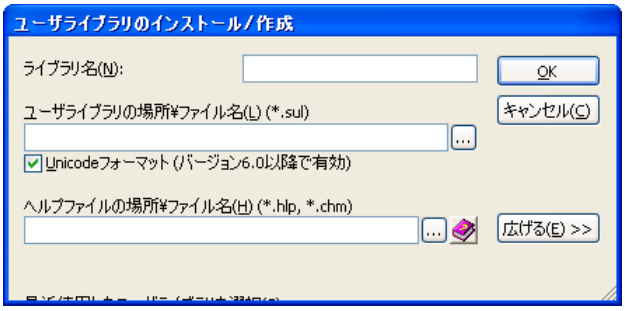
在此例子以第 8 章所作“Clock_Pulse“的 FB、以“Main_Clock_Pulse“的 Library 名稱登録做説明。

■操作步驟

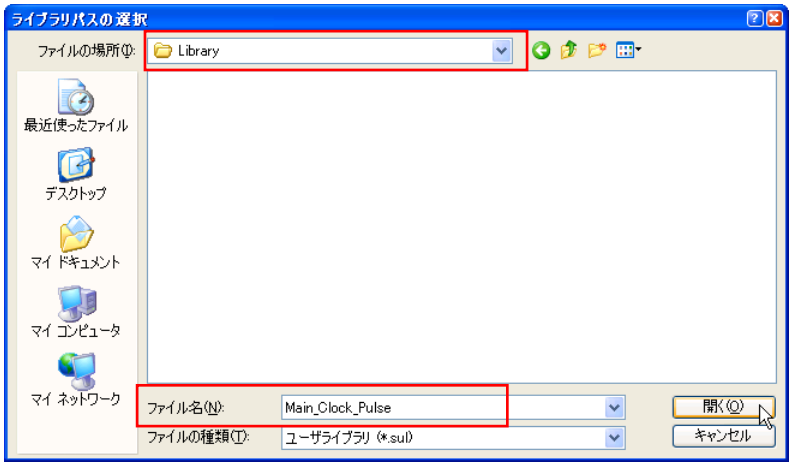
1. **打開第 8 章所作的“Clock_Pulse”的 Project。接著將 Project Navi 設定為 Active、點(選單)編集 → Library → Install／作成。(「Project 視窗」中的「Library」點右鍵、「Library」→「Install／作成」。)**



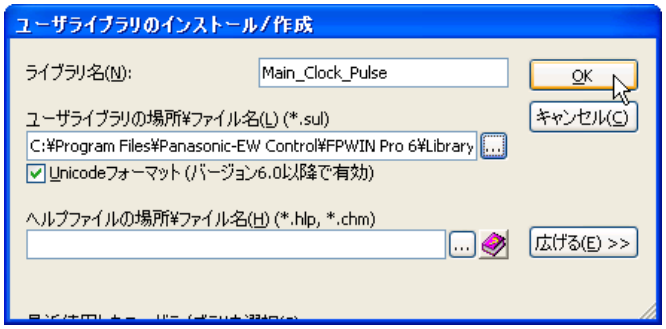
打開以下視窗。



2. **點選「USER Library 的地方¥檔案名稱(L) 」的項目的參照鍵 ... 、指定現在開始製作的 USER Library 儲存的檔案資料夾, 並輸入“Main_Clock_Pulse”到檔案名稱後、請點選「打開」。**

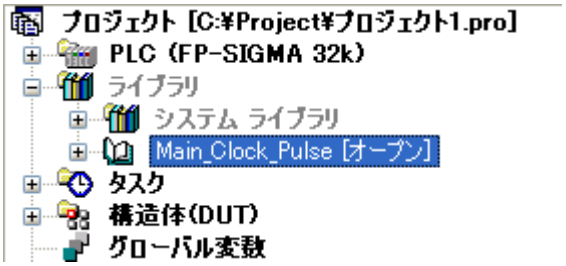


3. 切換到以下的畫面。



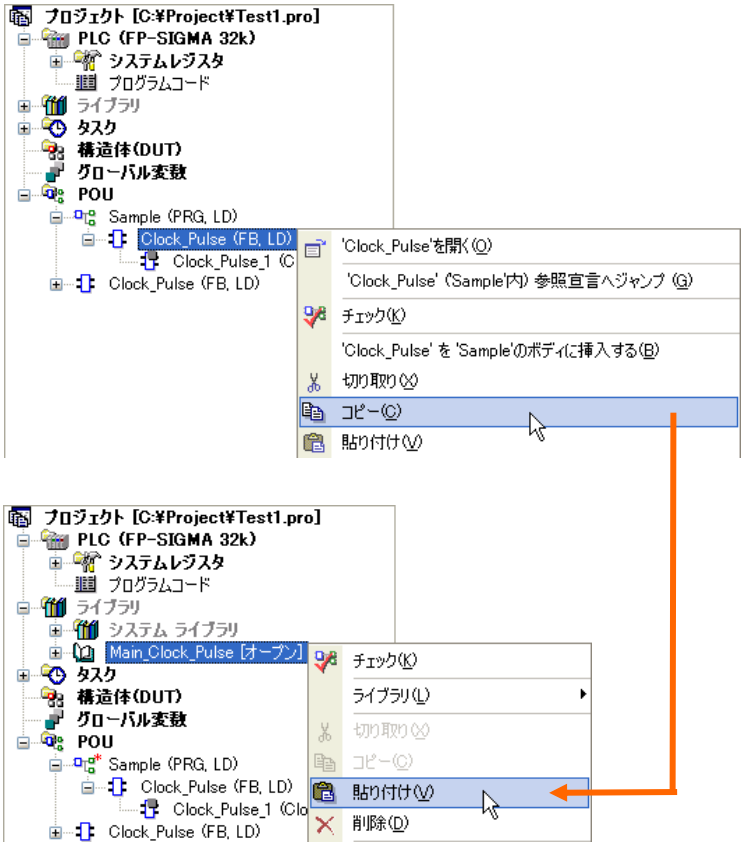
Help File 另外作成的情況時和上記同様的請指定 Help File 儲存的檔案資料夾。
沒有 Help 的情況時、請按下[OK]按鍵。

4. 仔細看 Project Navi、會發現有個名為 Main_Clock_Pulse 的 Library。

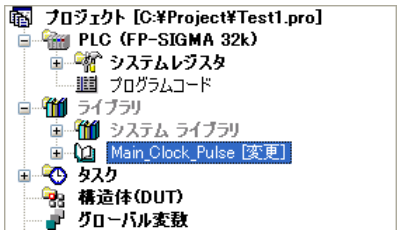


現在的状態為 Main_Clock_Puls 的 Library 為空的状態です。
接著、將 Function Block "Clock_Pulse"登録到 Library Main_Clock_Pulse。

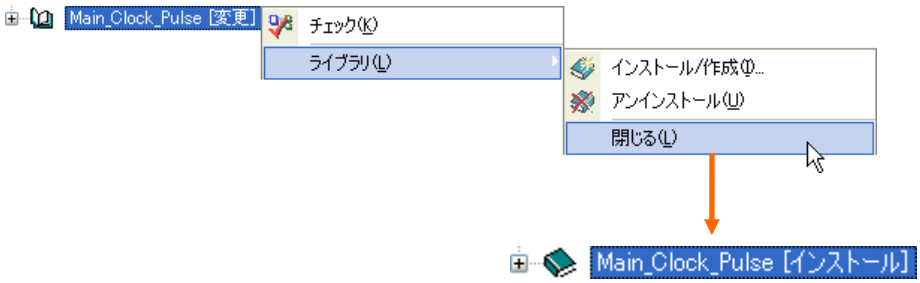
5. 在 Project Navi 內的 POU 內的 Clock_Pulse 點右鍵選擇「Copy」後、
請同樣的將 Project Navi 內的 Library 內的 Main_Clock_Pulse 點右鍵貼上。



Main_Clock_Pulse [OPEN] 的顯示會切換到 Main_Clock_Pulse [變更] 。



6. 在 Main_Pump[變更]按右鍵「Library」→選擇「關閉」、在儲存。
顯示會變更為 Main_Pump[Install]。



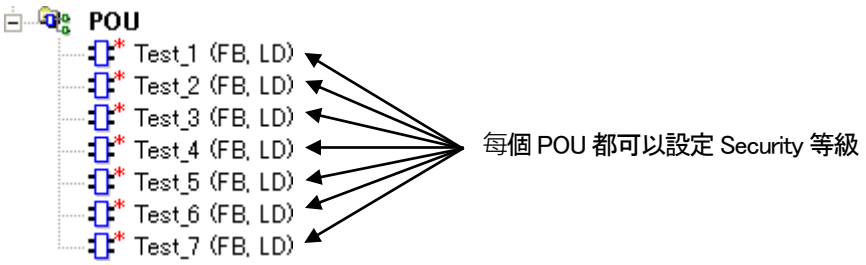
之後、新製成的 Project、即可以使用此 Library。

第15章

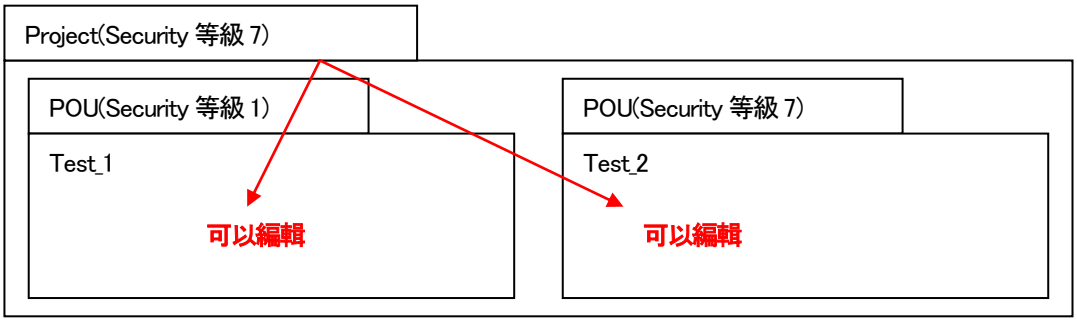
Security

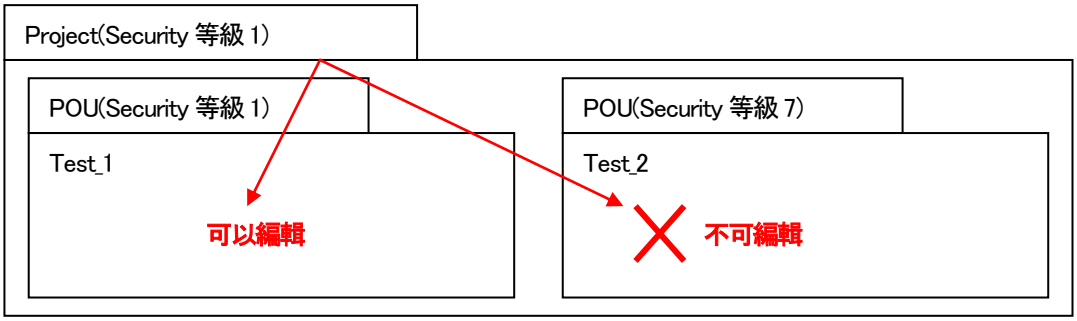
15-1 概要

在 FPWIN Pro 作為連接到一般的 PLC Security(Password 管理、禁止程式上傳)、
對於 Project(程式) Security 等級可以設定到 8 階段、依照 Password 的設定可以限制連結 POU 單位。



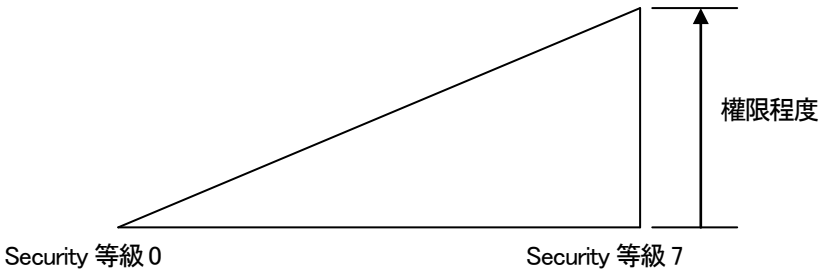
“Test_1” 是誰都可以編輯、“Test_2” 則是只有責任者可以編輯
還有無法編輯的、可以設定成唯讀。





●注意

對於比現在等級還高的 Security 等級的 Object(POU)、是無法更改 Security 等級。



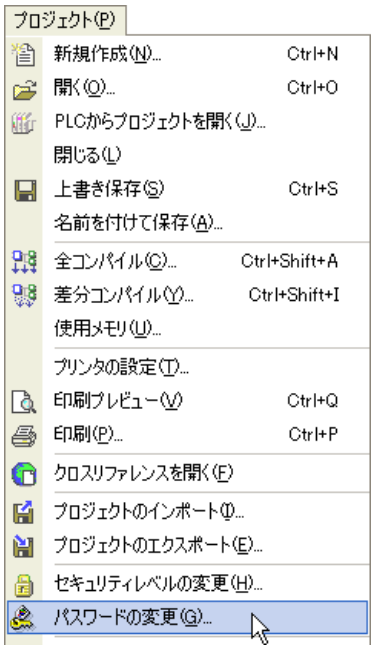
15-2 設定 Project 的 Security 等級

設定 Project 的 Security 等級。

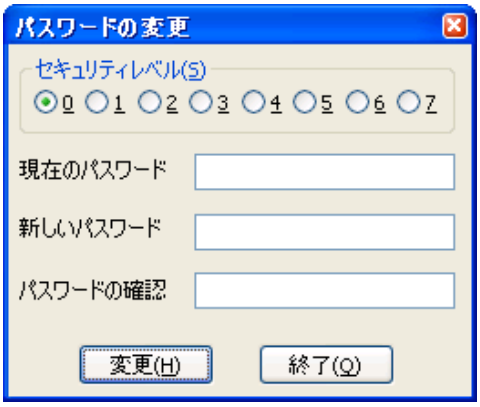
■操作步驟

1. **Password 的變更**

點(選單)Project → Password 的變更。

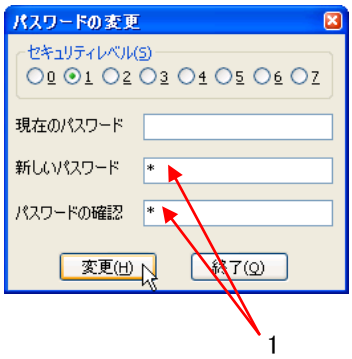


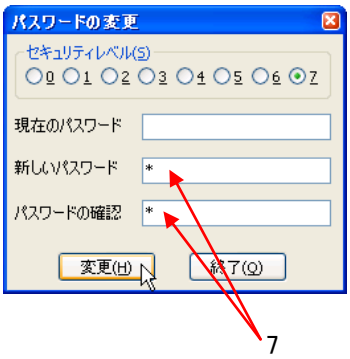
打開以下的視窗。



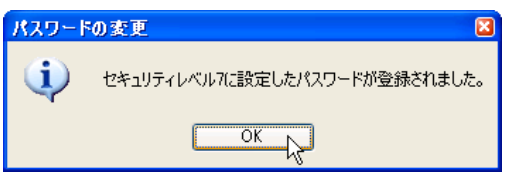
2. **設定各個 Security 等級的 Password**

新的 Project 並沒有設定 Password。
舊 Password 是空白的地方需要設定新的 Password。
用 8 階段設定想使用的 Security 等級進行 Password 設定。
本次設定 Security 等級【1】和【7】這 2 個。
為了容易了解、
設定“Security 等級 1”的 Password 是 “1”、“Security 等級 7” 的 Password 是 “7”。





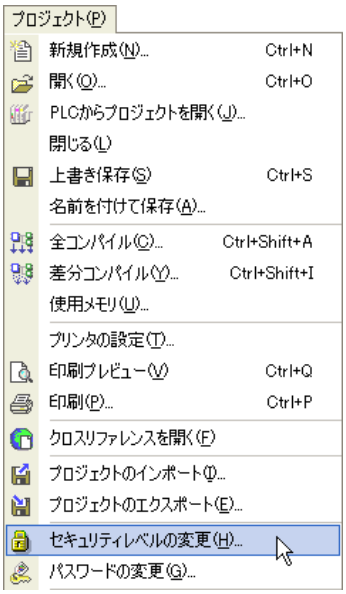
※設定好的 Password 變更時必須要在「舊 Password」欄輸入。



按下 変更(H) 鍵、
顯示左圖視窗、Password 登録結束。

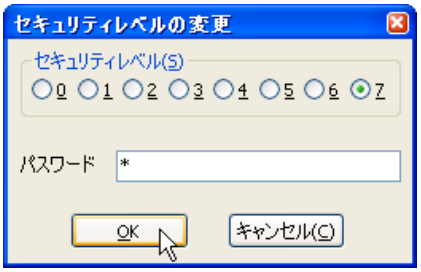
3. **切換 Project 的 Security 等級。**
   以上為止、設定 Security 等級分別為【1】和【7】。
   在此切換到 Security 等級 7。

   點(**選單**)**Project → Security 等級變更**。



   會顯示下圖的對話框、

   並將"7" 寫入到 Password、按下 [OK] 按鍵。



   ※在此 Project 的 Security 等級變成【7】。

   在此 Project 的 Security 的 Password 由管理者設定進行管理。
   請注意如果忘記密碼則無法再次呼叫出來。

15-3 POU 的 Security 等級設定

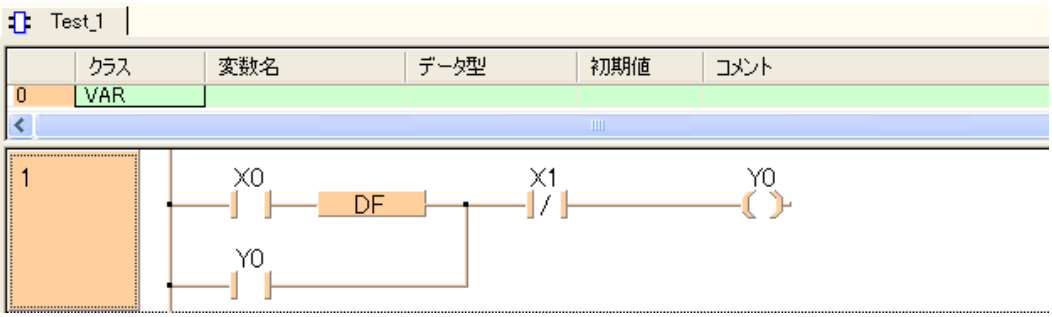
設定 POU 的 Security 等級。

■操作步驟

1. 輸入確認用的程式

在此使用 Test_1 的 POU。

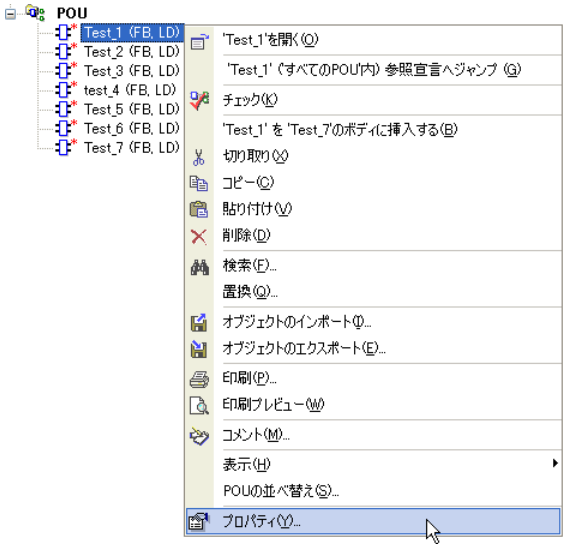
寫入簡單的程式、作為確認用。

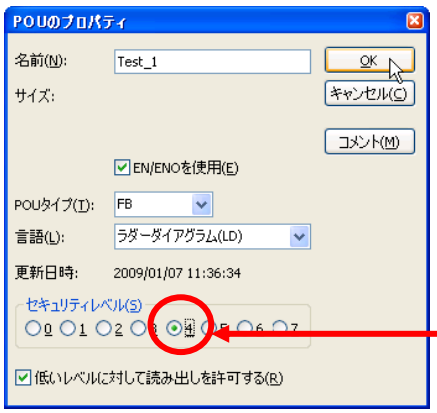


2. Security 的設定

Test_1 的 Security 等級設定為到“4”。

在 Test_1 按右鍵從選單選擇「内容」。





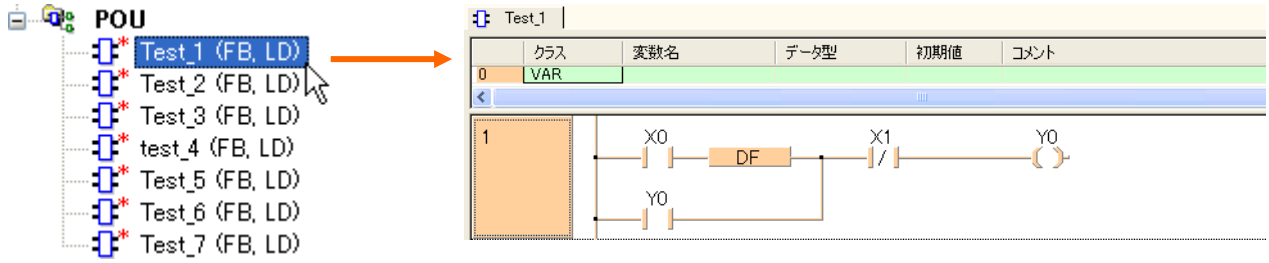
設定 Security 等級到 “4”。

OK 按鍵按下後、Security 等級的設定就結束。

15-4 讀出附有 Security 的 POU

到上述為止的操作、
Project 的 Security 等級是【7】
POU(Test_1)的 Security 等級是【4】

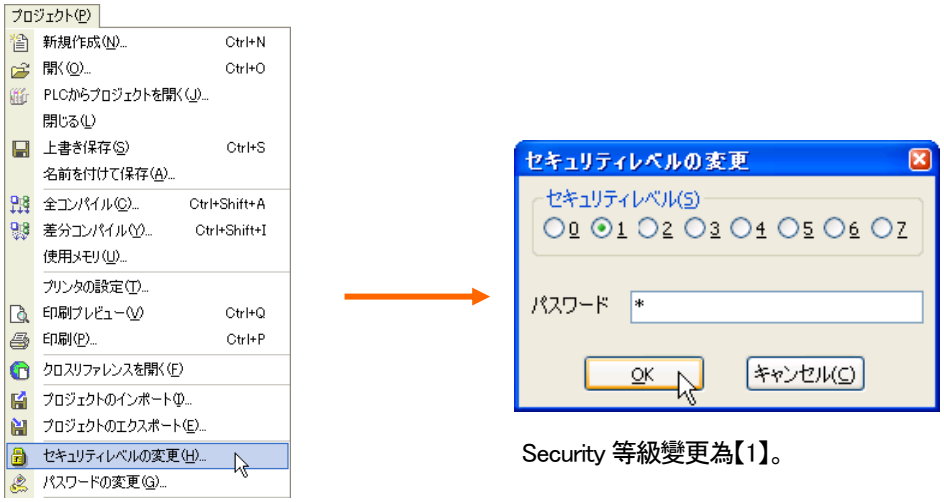
1. Project 的 Security 等級比 POU 的 Security 等級還高的時候



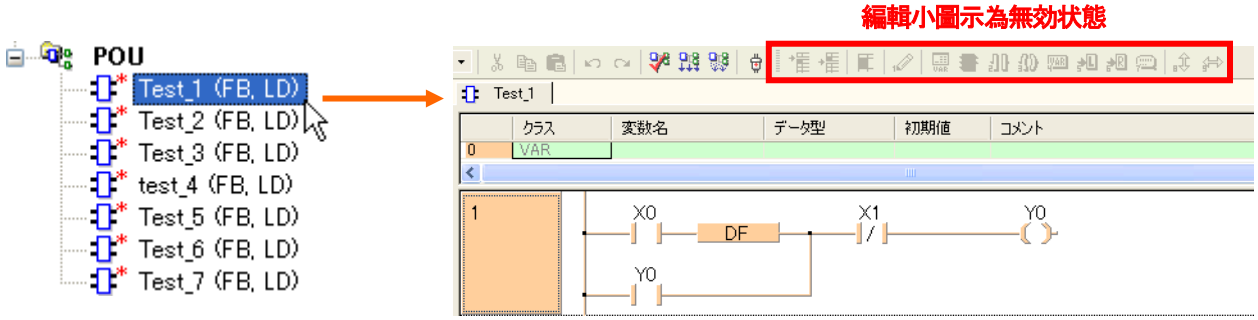
在 Project 的 Security 等級是【7】、所以比 POU 的 Security 等級【4】還高時、
可以進行普通的讀出和編輯。

2. Project 的 Security 等級比 POU 的 Security 等級還低的時候

Project 的 Security 等級變更為【1】。



Security 等級變更為【1】。

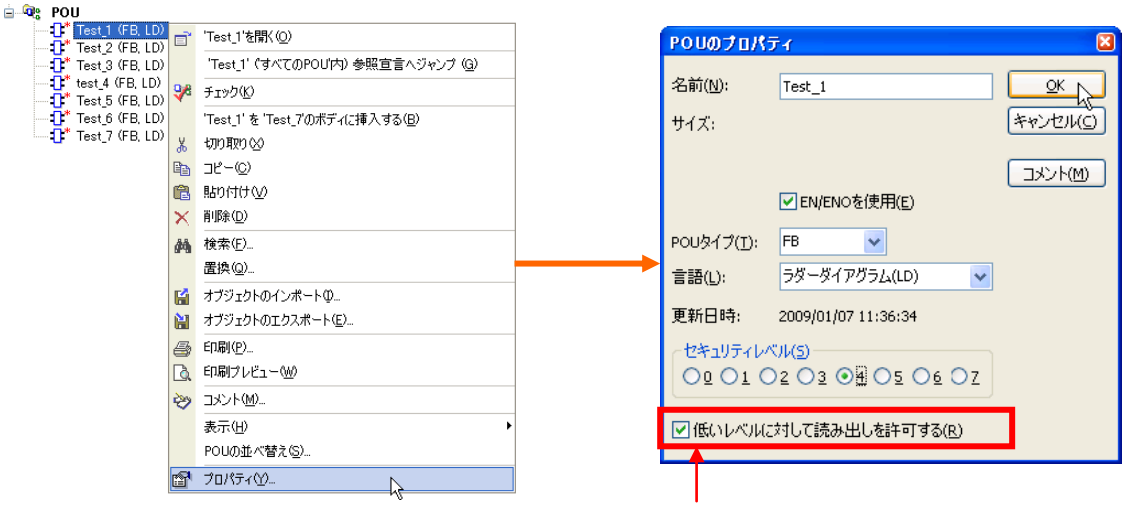


Project 的 Security 等級為【1】比 POU 的 Security 等級【4】還低時、
雖然程式可以讀出、不過可以得知上圖的「編輯小圖示為無效状態
所以無法編輯。

■備註

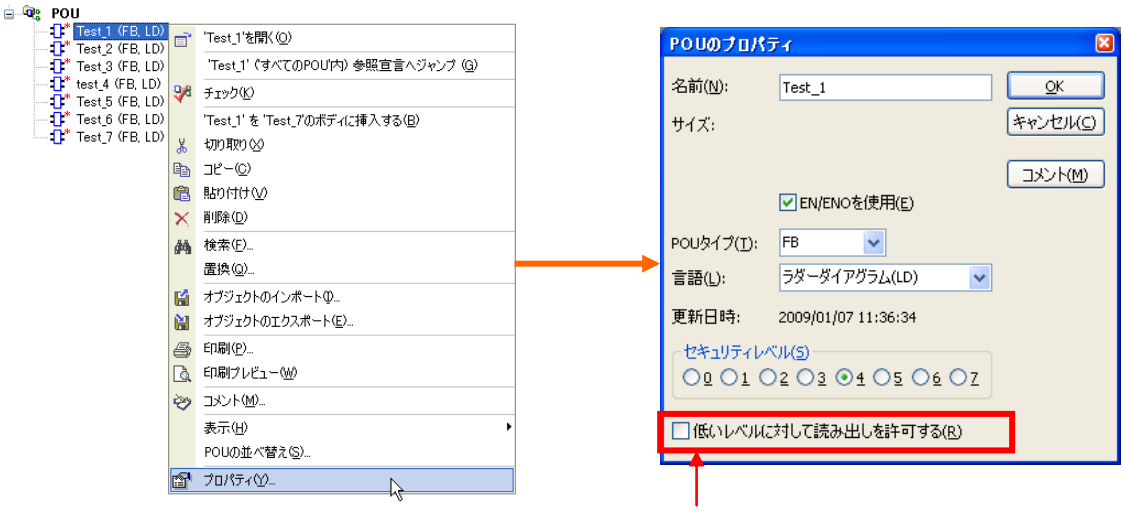
從 Security 等級低的 Project 讀出 Security 等級高的 POU 時、
①無法編輯、但是為可以唯讀的状態。
②無法唯讀的状態。
的兩個情況状態、在 POU 的 Security 等級設定時、可以預先設定。

① 雖然無法編輯、但是為可以参考的状態。



"對於低等級也可以讀出的許可"打勾。

② 無法唯讀的状態。



"對於低等級也可以讀出的許可"不打勾。

經過此 Security 設定、可以掌握 Object(POU)的編輯・閲覧的權限、
能夠防範對於程式的外漏以及錯誤操作等導致的程式變更相當有幫助。