Contents of Appendix

Appendix A : Functions & Function Blocks For ICP DAS PACs

Appendix A.1: Standard ISaGRAF Function Blocks

The following details the standard ISaGRAF function blocks that can be programmed with the I-8xx7, I-7188EG/XG, uPAC-7186EG & ISaGRAF WinCE PAC however labeled with "*" & "#" is not supported by I-8xx7 & I-7188EG/XG, while ISaGRAF WinCE PAC doesn't support items with "#" label only.

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

Please refer to the on-line help from the ISaGRAF workbench.
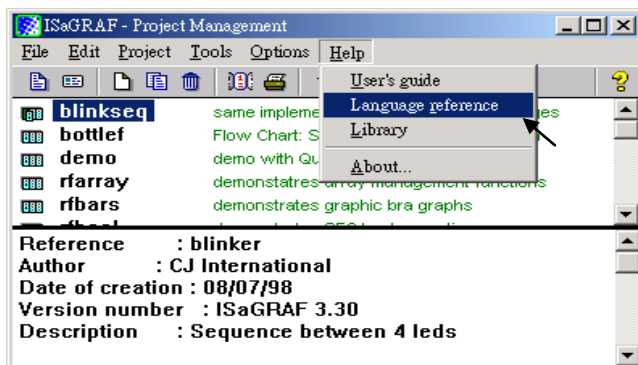
Appendix A.2: Adding New Function Blocks To ISaGRAF

Click [Start]→[Program]→[ISaGRAF3.4] or [ISaGRAF3.4], then click on "Libraries" to begin installing or updating ISaGRAF functions or function blocks.
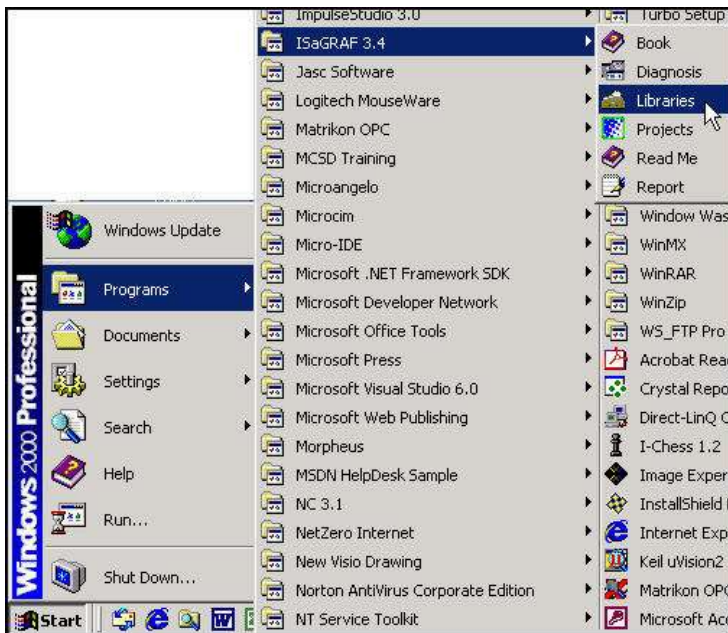
When you click on "Libraries" the "ISaGRAF Libraries" window will be open. To add a new function block or function select "Tools" from the menu bar and then click on "Archive".
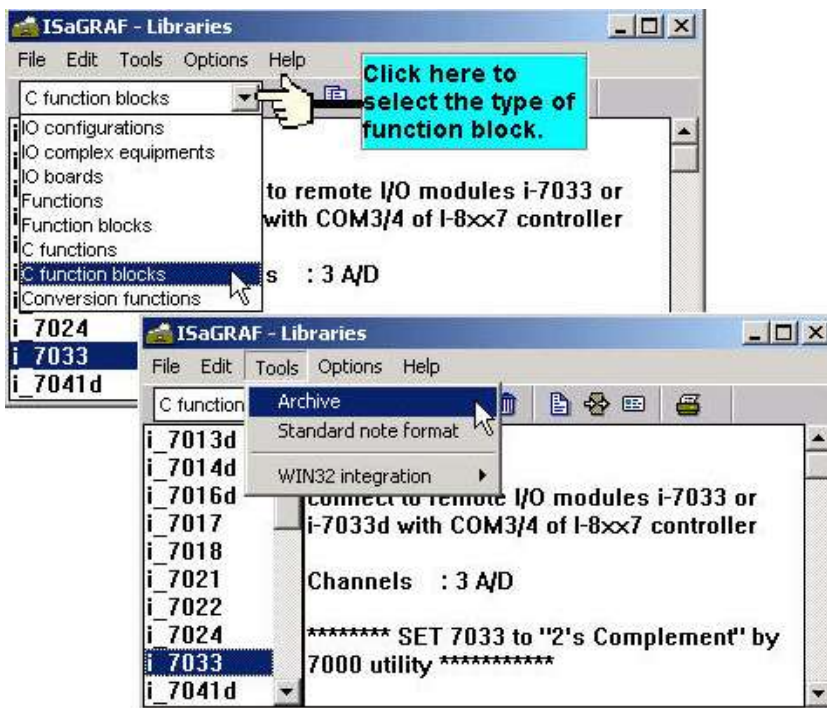
Click on the file name you want to "Archive" and then click "Browse" button to select the sub-directory where (CD_ROM: \Napdos\ISaGRAF\ARK\) you want to archive the function block library to.
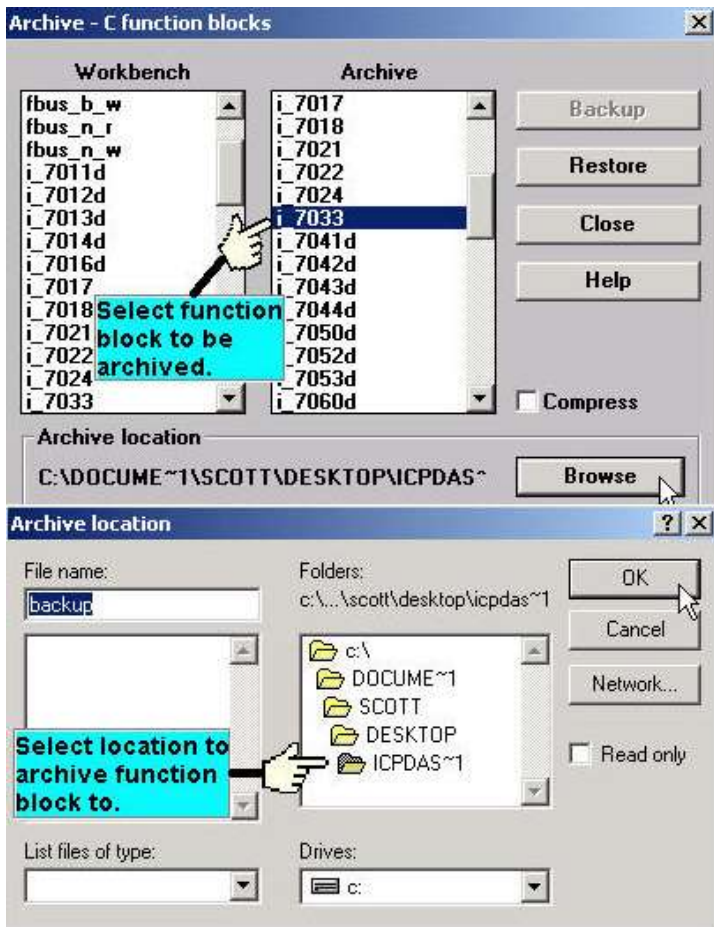
Select the new function block that you want to add from the "Archive" window, and then click on the "Restore" button. When you click on the "Restore" button the function block will be added to the ISaGRAF Workbench window.
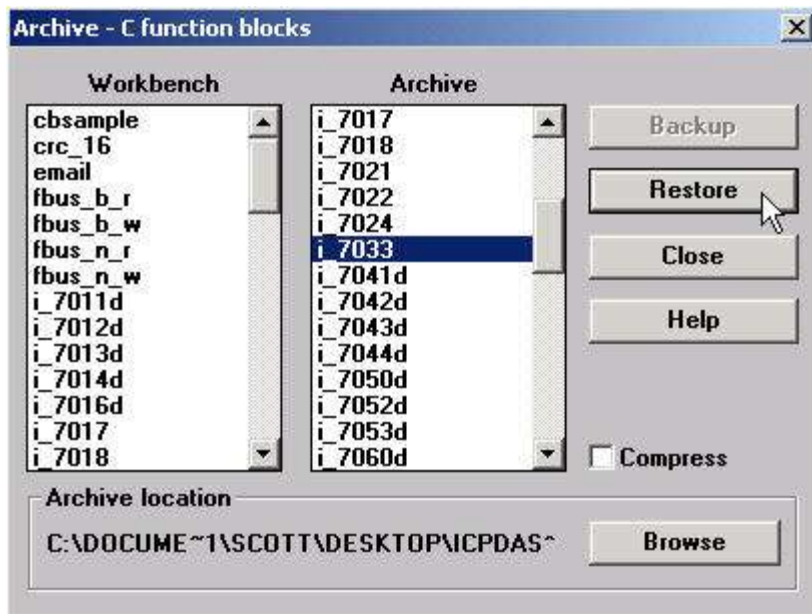
Appendix A.3: 7-Segment LED Reference Table

The following table provides the reference definitions for programming the 7 LED indicators on the I-8xx7 & I-7188EGD/XGD PAC system.
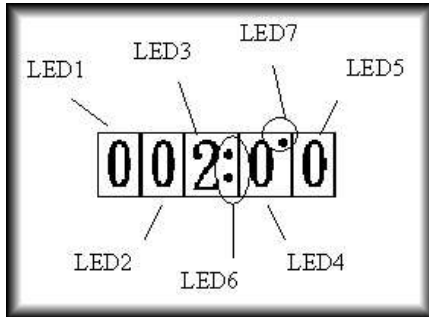
LED 6: Set to TRUE to display ":" (colon):
LED 7: Set to TRUE to display "." (period above LED 4)

Display Table: LED 1 through LED 5

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

Appendix A.4: Function Blocks For The Controller

The following function blocks have been developed specifically for the I-8xx7, I-7188EG/XG, μPAC-7186EG & ISaGRAF WinCE PAC system. The Standard_Function or Standard _Function Block is supplied by ISaGRAF. The C_Function or C_Function Block is supplied by ICP DAS PAC Company.

```
a4_20_to
Analn
Range
HiVal
LoVal      Q
```

A4_20_TO
■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
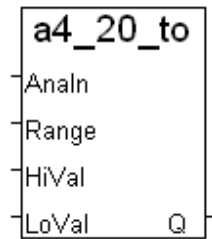
Description: C_Function
    Convert Analog Input from 4 - 20 mA to User's Engineering Value ("Real" format)
Input Parameters:

AnaIn_	Integer	the integer variable related to the Analog input board or module. The value is usually from -32768 to +32767 depending on the range setting of the IO board.
Range_	Integer	Range setting of the Analog input board or module.

> 16#6 :   -20 to +20 mA,   16#7 :   4 to +20 mA
> 16#D :   -20 to +20 mA,   16#1A :   0 to +20 mA,

HiVal_	Real	User's related High Eng. value when analog input signal is 20 mA
LoVal_	Real	User's related Low Eng. value when analog input signal is 4 mA. Ex: Convert I-8017H 's input signal from 4 - 20 mA to become 0 - 100 psi, please set HiVal_ = 100.0 , LoVal_ = 0.0 and Range_=16#6 (depeneds on range setting of the related IO board)

Return:

Q_	Real	The Engineering value after conversion. If given incorrect Range_ , returns 1.23E-20

Example:
Scale I-8017H 's current input with range setting as 6: (-20 to +20 mA) to user's engineering
    format of (0 to 100 psi). 4 mA means 0 psi , 20 mA means 100 psi

```
                    A4_20_TO
                   en        eno                          < >
I_8017_1_01─┤AnaIn_    Q_├─Psi_val
      16#6─┤Range_
     100.0─┤HiVal_
       0.0─┤LoVal_
```

> I-8017_1_01 is declared as integer input,
>   Psi_val is declared as Real internal.
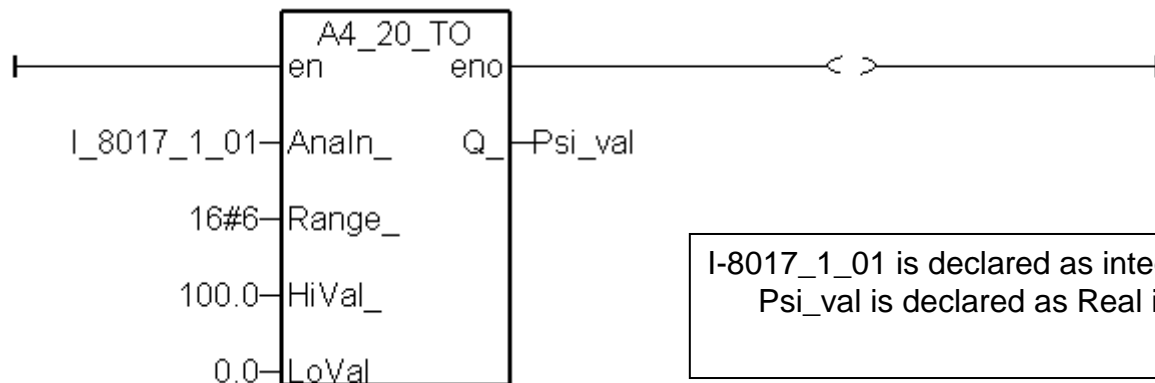
Note: 1. Please refer to similar functions: to_A4_20 , to_V0_10 , A4_20_to , V0_10_to
   2. When using A4_20_to, To_A4_20, To_V0_10, V0_10_to functions, user needs to upgrade the driver to the version listed below or latter version: i-7188EG: v2.16, i-7188XG: v2.14, i-8xx7:v3.18, so that the program can run well. (The older driver may cause the program cease after running a period of time.).

ANA

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
      Ana
IN          Q
```
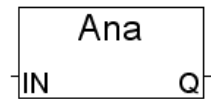
Description : Standard_Function

Convert any variable to an integer one

Input Parameters :

IN	Any	any non-integer analog value

Return :

Q	Integer	0 if IN is FALSE / 1 if IN is TRUE
		Number of milliseconds for a timer
		Integer part for real analog
		Decimal number represented by a string

Example :

(* FBD Example*)

```
  true  ───  Ana   ───  bres
             IN  Q

 t#1s46ms ── Ana   ───  tres
             IN  Q

  '0198' ───  Ana   ───  mres
             IN  Q
```
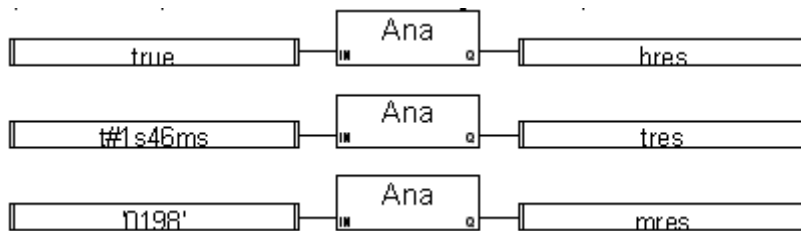
(* ST equivalence : *)

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

(* IL equivalence : *)
```
LD          true
ANA
ST          bres
LD          t#1s46ms
ANA
ST          tres
LD          '0198'
ANA
ST          mres
```

ARCREATE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
arcreate
ID
Size      ok
```
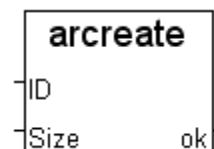
Description : Standard_Function

Creating a temporary Integer (Real) memory space for ISaGRAF program
data storing

Input Parameters :

ID	Integer	Set to 1 for ISaGRAF WinCE PAC, can not set as other number
SIZE	Integer	Could be (1 ~ 3,000,000)
		ISaGRAF WinCE PAC could set max. 3,000,000 of 32-bit Integer memory space.
		(Total are 3,000,000 x 4 = 12,000,000 bytes)

Return :

OK	Integer	1: creation success; other: fail.

Note :

1. ISaGRAF WinCE PAC can use ARcreate only once in the program, see description below:

   Create this memory space in the first PLC Scan:

   ```
   (* INIT initial as True,   TMP_v type is Internal Integer *)
   IF  INIT  THEN
        INIT  :=  False;
        TMP_v  :=  ARcreate(1, 2000000);
   END_IF;
   ```

2. Please refer to Section: 11.3.10 for example & description.
3. If the PAC is W-8xx7/W-8xx6, the driver should be ver. 3.36 or later to support ARCREATE, ARREAD and ARWRITE function.

ARRAY_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 array_r
─NUM_
─ADR    DATA─
```
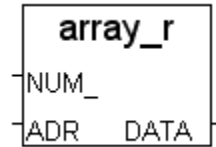
Description : C_Function
Read 1 byte from Byte Array (unsigned 8-bit)

Input Parameters :

NUM_   Integer   which array is read,
I-8xx7 & I-7188EG/XG: (1 ~ 24) , ISaGRAF WinCE PAC: (1 ~ 48)

ADR_   Integer   read which address in this array,
I-8xx7 & I-7188EG/XG: (1 ~ 256), ISaGRAF WinCE PAC: (1 ~ 512)

Return :

DATA_   Integer   the read byte value (0~255)

Example :
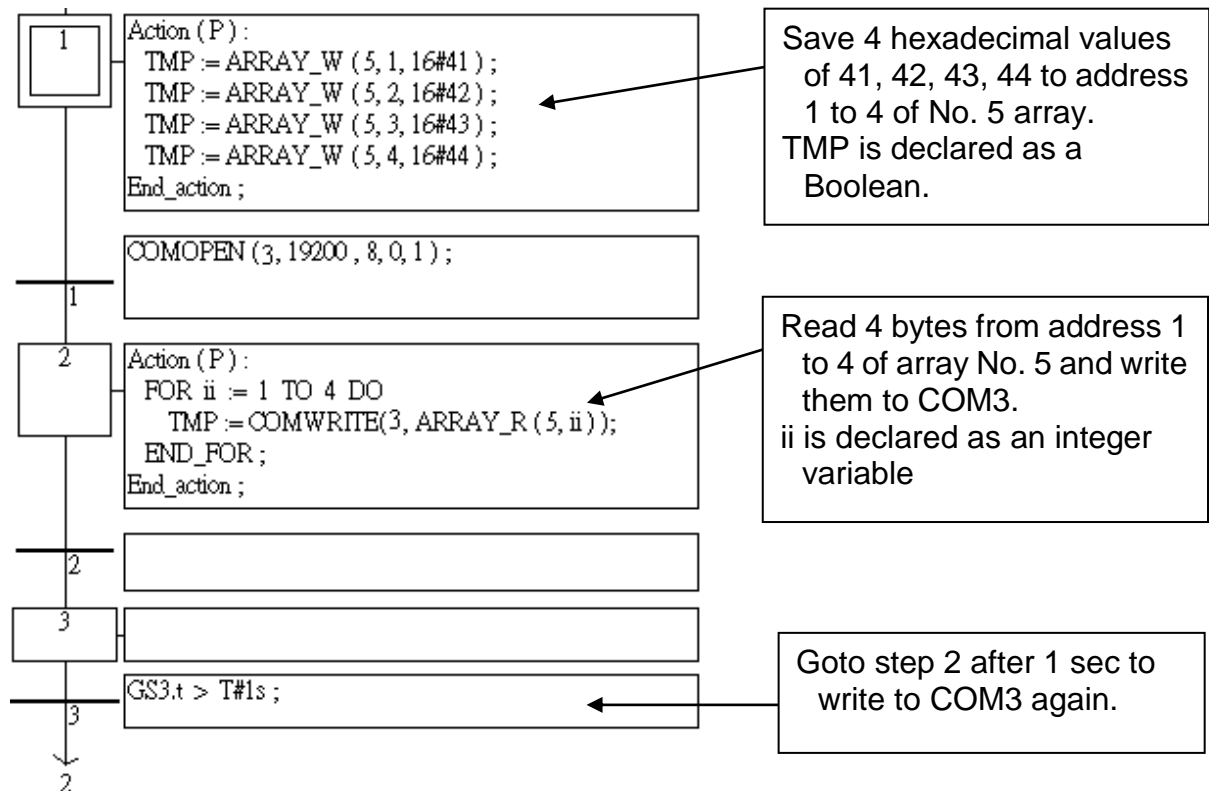
```
1   Action (P) :
     TMP := ARRAY_W (5, 1, 16#41) ;
     TMP := ARRAY_W (5, 2, 16#42) ;
     TMP := ARRAY_W (5, 3, 16#43) ;
     TMP := ARRAY_W (5, 4, 16#44) ;
    End_action ;
```

> Save 4 hexadecimal values of 41, 42, 43, 44 to address 1 to 4 of No. 5 array.
> TMP is declared as a Boolean.

```
    COMOPEN (3, 19200, 8, 0, 1) ;
1
```

```
2   Action (P) :
     FOR ii := 1 TO 4 DO
       TMP := COMWRITE(3, ARRAY_R (5, ii));
     END_FOR ;
    End_action ;
```

> Read 4 bytes from address 1 to 4 of array No. 5 and write them to COM3.
> ii is declared as an integer variable

```
2
3
    GS3.t > T#1s ;
3
2
```

> Goto step 2 after 1 sec to write to COM3 again.

Note:
The data stored in array are cleared after power off
For the W-8xx7: Recommend upgrade to ver. 3.36 or higher version

ARRAY_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        array_w
─NUM_
─ADR_
─DATA        Q─
```
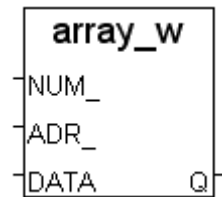
Description : C_Function
Save one byte to a byte array

Input Parameters :

NUM_	Integer	Array ID to be operated, valid range values for:
		I-8xx7 & I-7188EG/XG: (1 ~ 24); ISaGRAF WinCE PAC: (1 ~ 48).
ADR_	Integer	Address in the array where the byte is to be stored.
		I-8xx7 & I-7188EG/XG: (1 ~ 256); ISaGRAF WinCE PAC: (1 ~ 512).
DATA_	Integer	The byte value to be saved to, valid range values from 0 to 255

Return :

Q_	Boolean	If OK, return TRUE; else return FALSE

Example: Please refer to the example of "ARRAY_R".

Note:
The data stored in array are cleared after power off.
For the W-8xx7: Recommend upgrade to ver. 3.36 or higher version

ARREAD

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
       arread
  ID
  Pos          Q
```
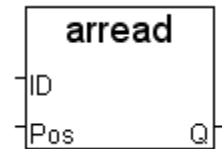
Description : Standard_Function

    Read an Integer(32-bit signed) from user created temporary memory space

Input Parameters :

ID	Integer	Set to 1 for ISaGRAF WinCE PAC, can not set as other number.
POS	Integer	The position to be read, could be (1 ~ 3,000,000)
		If POS is larger than the number created by ARcreate, the return data will be wrong.

Return :

Q	Integer	The Integer has been read.

ARWRITE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
       arwrite
  ID
  Pos
  IN          ok
```

Description : Standard_Function

    Write an Integer to user created temporary memory space

Input Parameters :

ID	Integer	Set to 1 for ISaGRAF WinCE PAC, can not set as other number.
POS	Integer	The position want to write, could be (1 ~ 3,000,000)
		If POS is larger than the number created by ARcreate, data will not be written.
IN	Integer	The written Integer.

Return :

OK	Integer	1: success; other: fail.

Note :
1. If user did not use ARcreate to create a memory space before using ARread & ARwrite, the return value will be wrong.
2. If user wants to read/write a Real value in the temporary memory space, please use Real_Int & Int_Real function to map between Integer & Real.
   (Please refer to the example of section 11.3.10 of the ISaGRAF User Manual)
3. If the PAC is W-8xx7/W-8xx6, the driver should be ver. 3.36 or later to support ARCREATE, ARREAD and ARWRITE function.

ARY_F_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
ary_f_r
NUM_
ADR    DATA
```
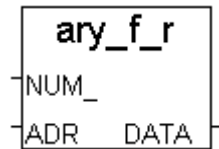
Description : C_Function
> Read one float value (32-bit format) from an float array

Input Parameters :
NUM_	Integer	Array ID to be operated, valid range:
		For ISaGRAF WinCE PAC is from 1 to 18.
		For I-7188EG/XG & I-8xx7 is from 1 to 6.
ADR_	Integer	Address in the array where the Integer is to be stored, (1 ~ 256)

Return :
DATA_	Real	The returned Float value (32-bit float)

ARY_F_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
ary_f_w
NUM_
ADR_
DATA    Q
```
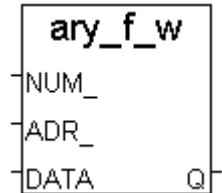
Description : C_Function
> Save one float value (32-bit format) to a float array

Input Parameters :
NUM_	Integer	Array ID to be operated, valid range:
		For ISaGRAF WinCE PAC is from 1 to 18.
		For I-7188EG/XG & I-8xx7 is from 1 to 6.
ADR_	Integer	Address in the array where the integer is to be stored. (1 ~ 256)
DATA_	Real	The float value to be saved to (32-bit float)

Return :
Q_	Boolean	If OK, return TRUE, else return FALSE

Note:
1. The data stored in array are cleared after power off.
2. For I-7188EG/XG & I-8xx7, the float array and Integer array use the same memory. Please use them carefully. If use ARY_F_R to read the REAL value, but the value in the memory address is Integer data, the return value is not correct.
3. For using ARY_F_R and ARY_F_W, the driver must be the version listed below or higher version:
> I-7188EG : since ver. 2.17
> I-7188XG : since ver. 2.15
> I-8xx7   : since ver. 3.19
> W-8xx7   : Recommend upgrade to ver. 3.36 or higher version

ARY_N_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
   ┌──────────────┐
   │   ary_n_r    │
 ──┤NUM_          │
   │              │
 ──┤ADR     DATA  ├──
   └──────────────┘
```
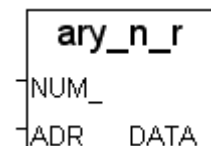
Description : C_Function
Read long integer (signed 32-bit) from Integer Array (signed 32-bit)

Input Parameters :

NUM_	Integer	Which array is read, valid range:
		I-8xx7 & I-7188EG/XG: (1 ~ 6); ISaGRAF WinCE PAC: (1 ~ 18);
ADR_	Integer	Read which address in this array (1 ~ 256)

Return :

DATA_	Integer	The Integer been read

ARY_N_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
   ┌──────────────┐
   │   ary_n_w    │
 ──┤NUM_          │
   │              │
 ──┤ADR_          │
   │              │
 ──┤DATA       Q  ├──
   └──────────────┘
```
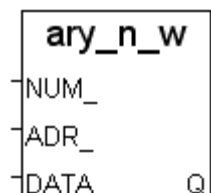
Description : C_Function
Write 1 long integer (signed 32-bit) to array

Input Parameters :

NUM_	Integer	Write to which array, valid range values:
		I-8xx7 & I-7188EG/XG: (1 ~ 6); ISaGRAF WinCE PAC: (1 ~ 18).
ADR_	Integer	Write to which address in this array (1-256)
DATA_	Integer	The Integer value to write

Return :

Q_	Boolean	True: ok; False: fail.

Note: 1. The Long Integer array use the same memory as short integer array. Be careful if using both of them at the same time.

Word Array(num,adr)	Integer Array(num,adr)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

2. The data stored in array are cleared after power off.

Example: Please refer to the example of "ARRAY_R" .

ARY_W_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  ary_w_r
┌──────────┐
┤NUM_      │
│          │
┤ADR   DATA├
└──────────┘
```

Description : C_Function
      Read Short Integer (signed 16-bit) from array

Input Parameters :

NUM_	Integer	Which Short Integer array is read,
		I-8xx7 & I-7188EG/XG: (1 ~ 12);
		ISaGRAF WinCE PAC: (1 ~ 36).
ADR_	Integer	Read which address in this array (1 ~ 256)

Return :

DATA_	Integer	The Short Integer been read (-32768 ~ +32767)
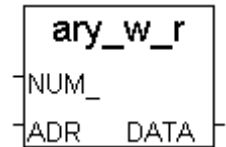
ARY_W_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  ary_w_w
┌──────────┐
┤NUM_      │
│          │
┤ADR_      │
│          │
┤DATA     Q├
└──────────┘
```

Description : C_Function
      Write 1 Short Integer (signed 16-bit) to array

Input Parameters :

NUM_	Integer	Write to which Short Integer array,
		I-8xx7 & I-7188EG/XG: (1 ~ 12);
		ISaGRAF WinCE PAC: (1 ~ 36).
ADR_	Integer	Write to which address in this array (1-256).
DATA_	Integer	The Short Integer value to write (-32768 ~ +32767)

Return :

Q_	Boolean	TRUE : ok; FALSE : fail.
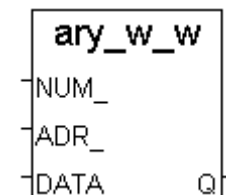
Note: 1. The Long Integer array use the same memory as the Short Integer array. Be careful if using both of them at the same time.

Word Array (num,adr)	Integer Array (num,adr)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

     2. The data stored in array are cleared after power off.

Example: Please refer to the example of "ARRAY_R"

BCD_V

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────┐
│  BCD_V  │
│         │
┤IN     Q├
└─────────┘
```

Description : C_Function
    Convert BCD value to Decimal value
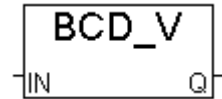
Input Parameters :
    IN_   Integer   The BCD value
Return :
    Q_   Integer   The decimal value, 0 ~ 99999999. EX:
                                    16#12345 → 12345
                                    18   → 12

BIN2ENG

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
┌──────────┐
│ bin2eng  │
┤In_       │
┤Hi_2s     │
┤Lo_2s     │
┤Hi_En     │
┤Lo En  Out├
└──────────┘
```

Description : C_Function
    Convert 2's complement to Engineering format

Input Parameters :
    IN_   Integer   The 2's complement to be converted
    HI_2S_   Integer   Max. 2's complement value of In_
    LO_2S_   Integer   Min. 2's complement value of In_
    HI_EN_   Integer   Max. Engineering value of Out_
    LO_EN_   Integer   Min Engineering value of Out_
Return :
    OUT_   Integer   The converted Engineering format value
EX:
    HI_2s_ = 32767 , LO_2s_ = -32768, HI_EN_ = 1000, LO_EN_ = -1000
                                    IN_ = 16383   → OUT_= 500
                                    IN_ = -12345   → OUT_= -377

Note: HI_2S_ can not be the same value as LO_2S_, and both value must be wrote in the
          range of (-32768 ~ +32767).

BIT_WD

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        ┌──────────┐
        │  bit_wd  │
       ─┤B1_       │
       ─┤B2_       │
       ─┤B3_       │
       ─┤B4_       │
       ─┤B5_       │
       ─┤B6_       │
       ─┤B7_       │
       ─┤B8_       │
       ─┤B9_       │
       ─┤B10_      │
       ─┤B11_      │
       ─┤B12_      │
       ─┤B13_      │
       ─┤B14_      │
       ─┤B15_      │
       ─┤B16_  VAL_├─
        └──────────┘
```

Description : C_Function
    Convert 16 Boolean values to a Word value

Input Parameters :
    B1_ ~ B16_   Boolean   The 16 Booleans to be converted
Return :
    VAL_   Integer   The converted Word value (-32768 ~ +32767)

EX: If B1_ , B2_ is TRUE , others is FALSE , the Word value is 3.
    If B1_ , B8_ is TRUE , others is FALSE , the Word value is 129.
    If all of B1_ ~ B16 is TRUE , the Word value is -1.
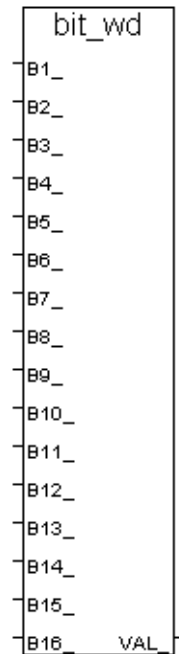
BLINK

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        ┌──────────┐
        │  blink   │
       ─┤RUN       │
       ─┤CYCLE    Q├
        └──────────┘
```

Description : Standard_Function Block
    Generate an ON_OFF blinking signal.

Input Parameters :
    RUN   Boolean   Mode: TRUE = blinking / FALSE = reset the output to false
    CYCLE   Timer   Blinking period

Return :
    Q   Boolean   Output ON_OFF blinking signal

Timing diagram:

```
Timing diagram:

RUN _____┌──────────────┐_____
            │              │
Q _____│_┌─┐_┌─┐_┌─┐__│_____
            ╎ └─┘ └─┘ └─┘  ╎
            ╎←──→╎
            ╎CYCLE
```
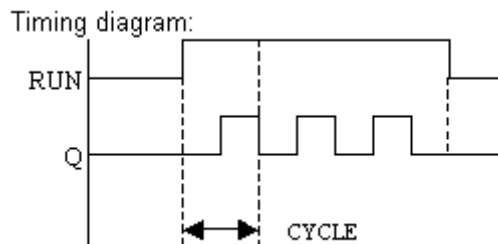
(Next page please)

Note:

BLINK Function Block could generate a Pulse True in a fixed interval. It could be used to some applications that need to do something every fixed period, such as:

(* LD Program: *)

```
        ┌─────────┐
        │ BLINK   │                    Pulse1
  ┤─────┤RUN    Q ├──────────────────────○N○───────────┤
        │         │
  T#2s──┤CYCLE    │
        └─────────┘
```
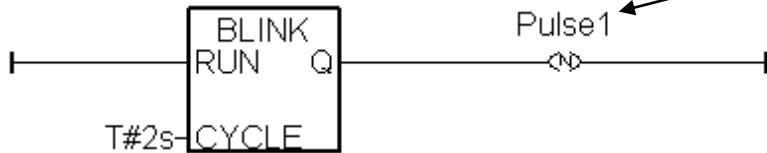
┌─────────────────────────────────┐
│ Pulse1 is declared as Boolean   │
│ Internal.   │
│ It will do something every 2s in │
│ this example.   │
└─────────────────────────────────┘

(* ST Program: *)
```
     IF  Pulse1  THEN
         (* Do some thing *)
         (*  ……        *)
     END_IF;
```

If the time interval is smaller than 200ms or the PLC Scan Time becomes larger, the time precision will be low. For example, do something every 50ms. The 50ms is a smaller interval and the value is close to the PLC Scan Time. The result will become not so precise. User can change the program as below to improve the accuracy.

ST Program:

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

     IF   T1 >= T1_next  THEN
         IF  T1 > T#22h  THEN
             T1 := T#0s;
             T1_next := T#0s;
         END_IF;
         T1_next := T1_next + T#50ms;
         (*Do some thing *)
         (*  ……        *)
     END_IF;
```

┌─────────────────────────────────────┐
│ INIT is declared as Boolean Internal │
│   Initial value is TRUE   │
│ T1 & T1_next is declared as Timer   │
│ Internal.   │
└─────────────────────────────────────┘

BOO

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────┐
│   Boo   │
│IN      Q│
└─────────┘
```
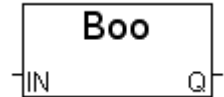
Description : Standard_Function

Convert any variable to a Boolean one

Input Parameters :

IN	Any	Any non-boolean value

Return :

Q	Boolean	TRUE for non-zero numerical value
		FALSE for zero numerical value
		TRUE for 'TRUE' message
		FALSE for 'FALSE' message

Example :

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

```
┌────────────┐   ┌─────────┐   ┌────────────┐
│     10     │───┤IN Boo  Q├───│    ares     │
└────────────┘   └─────────┘   └────────────┘

┌────────────┐   ┌─────────┐   ┌────────────┐
│    t#0s    │───┤IN Boo  Q├───│    tres     │
└────────────┘   └─────────┘   └────────────┘

┌────────────┐   ┌─────────┐   ┌────────────┐
│  'false'   │───┤IN Boo  Q├───│    mres     │
└────────────┘   └─────────┘   └────────────┘
```
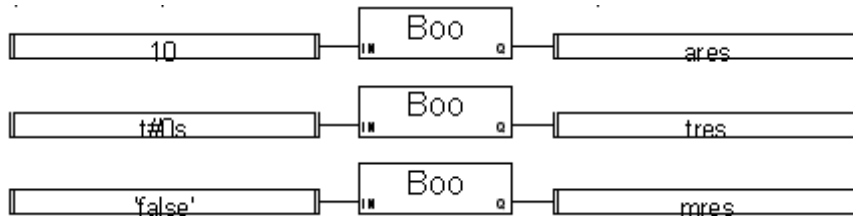
(* ST Equivalence: *)

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

(* IL equivalence: *)

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

CJC
■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
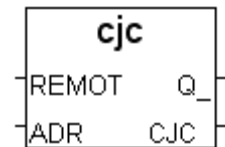
```
        cjc
 REMOT      Q_
 ADR      CJC
```

Description : C_Function Block
Read Cold-Junction Compensation (CJC) temperature of module I-7018 / 7019 & I-87018 / 87019

Input Parameters :
REMOTE_   Boolean   Must be a constant value, not a variable
If I-7018/7019 & I-87018/87019 are used as remote I/O,
REMOTE_ is TRUE.
If I-87018/87019 is plugged in Main Control Unit,
REMOTE_ is FALSE.
ADR_   Integer   Must be a constant value, not a variable
If REMOTE_ is TRUE, ADR_ means the remote I/O module's address (1 ~ 255)
If REMOTE_ is FALSE, ADR_ means slot No., 0 ~ 7 (W-8xx7/8xx6 ADR_ means slot No., 1 ~ 7)

Return :
Q_   Boolean   TRUE: work ok.
FALSE: communication is bad, the following return value has no meaning.
CJC_   Integer   The Analog input value (2's complement format)

D3B4   --->   0000   --->   7FFF (hex.)
Val: -11340 --->   0   ---> 32767 (dec.)
Temperature: -45 --->   0   ---> +130 (degree Celsius)

Note : (VERY IMPORTANT)

1. If I-87018 / 87019 is plugged in the Main Control Unit, please connect the 87018 / 87019 I/O board first, or the "CJC" function block will not work.

2. If I-7018 / 7019 and I-87018 / 87019 are used as RS-485 remote I/O, pleas use DCON utility to set format as 2's compliment.

3. For the W-8xx7/8xx6 should use the driver version 3.21 or latter version.

CJC_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function Block

Read/Write CJC compensation of the I-7018, I-7019, I-87018
(CJC: Cold-Junction Compensation)

```
 ┌──────────────┐
 │   cjc_sts    │
 │─REMOT        │
 │              │
 │─ADR_      Q_─│
 │              │
 │─SET SNOW S─  │
 └──────────────┘
```

Input Parameters :

REMOTE_	Boolean	Must be a constant value, not a variable
		If I-7018/7019 and I-87018/87019 are used as remote I/O, REMOTE_ is TRUE.
		If I-87018/87019 is plugged in Main Control Unit, REMOTE_ is FALSE.
ADR_	Integer	Must be a constant value, not a variable
		If REMOTE_ is TRUE, ADR_ is the remote I/O module's address (1 ~ 255).
		If REMOTE_ is FALSE, ADR_ is the slot No., (0 ~ 7).
SET_STS_	Boolean	If SET_STS_ is FALSE, disable CJC compensation. If SET_STS_ is TRUE, enable CJC compensation.

Return :

Q_	Boolean	TRUE: work ok.
		FALSE: communication is bad, the following return value has no meaning.
NOW_STS_	Boolean	If NOW_STS_ is FALSE, disable CJC compensation now
		If NOW_STS_ is TRUE, enable CJC compensation now

Note (VERY IMPORTANT):

1. If the I-87018/87019 is plugged in the Main Control Unit, please connect the 87018/ 87019 I/O board first, then "CJC" function block can work.

2. If the I-87018/87019 is used as a RS-485 Remote I/O, please operate the following steps before connecting them.

    A. At I/O module side, please use "DCON Utility" to setup:
       *1. Set "address" to a unique No. (1~255)
       *2. Set proper "checksum disabled" or "checksum enabled" and set Analog Input module to be "2's complement".
       *3. Set communication parameter to "baud rate" and "8,N,1"
       *4. Set "Range Type" if using Analog board

    B. At I-8xx7/I-7188EG/I-7188XG side:
       *1. Connect I/O complex equipment "bus7000B" and set its "baud rate" and "Checksum" equal to the "baud rate" & "Checksum" setting of the I/O modules.
       *2. I-8xx7 & 7188EG/XG can connect up to 64 remote I/O modules, W-8xx7 can up to 255.
       *3. Please use i_7xxx function block or i_87xxx Function Block when programming RS-485 Remote I/O module.

CJC2

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC
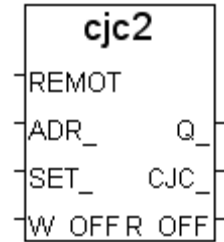
```
       cjc2
─REMOT
─ADR_        Q_─
─SET_      CJC_─
─W OFF R OFF─
```

Description : C_Function Block

　　　　Read Cold-Junction Compensation (CJC) temperature
　　　　of module I-7018/7019 & I-87018/87019 with Offset

Input Parameters :

REMOTE_	Boolean	Must be a constant value, not a variable.
		If I-7018/7019 & I-87018/87019 are used as remote I/O,
		REMOTE_ is TRUE.
		If I-87018/87019 is plugged in Main Control Unit,
		REMOTE_ is FALSE.
ADR_	Integer	Must be a constant value, not a variable
		If REMOTE_ is TRUE,
		ADR_ is the remote I/O module's address (1 ~ 255).
		If REMOTE_ is FALSE,
		ADR_ is the slot No., (0 ~ 7).
SET_	Boolean	Start to set CJC temperature offset if TRUE
W_OFFSET_	Integer	CJC temperature offset in decimal format,
		from -4096 to +4096, each count is 0.01 degree Celsius

Return :

Q_	Boolean	TRUE: work ok.
		FALSE: communication is bad, the following return value has no meaning.
CJC_	Integer	The Analog input value (2's complement format)
		D3B4　--->　0000　--->　7FFF　(hex.)
		Val: -11340　--->　　0　--->　32767 (dec.)
		Temperature:　-45　--->　0　--->　+130　(degree Celsius)
R_OFFSET	Integer	Read CJC temperature offset

Note (VERY IMPORTANT):

1. If the I-87018/87019 is plugged in the Main Control Unit, please connect the 87018/ 87019 I/O board first, then "CJC2" function block can work.
2. If the I-87018/87019 is used as a RS-485 Remote I/O, please operate the following steps before connecting them.
   A. At I/O module side, please use "DCON Utility" to setup:
   　*1. Set "address" to a unique No. (1~255)
   　*2. Set proper "checksum disabled" or "checksum enabled" and set Analog Input module to be "2's complement".
   　*3. Set communication parameter to "baud rate" and "8,N,1"
   　*4. Set "Range Type" if using Analog board
   B. At I-8xx7/I-7188EG/I-7188XG side:
   　*1. Connect I/O complex equipment "bus7000B" and set its "baud rate" and "Checksum" equal to the "baud rate" & "Checksum" setting of the I/O modules.
   　*2. I-8xx7 & 7188EG/XG can connect up to 64 remote I/O modules, W-8xx7 can up to 255.
   　*3. Please use i_7xxx function block or i_87xxx Function Block when programming RS-485 Remote I/O module,

COM_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
com_sts
PORT_      Q_
RTS_      CTS_
DTR       DSR
```
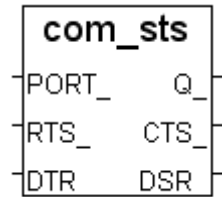
Description : C_Function Block
        Set RTS, DTR and Get CTS, DSR of COM PORT

Input Parameters :

PORT_   Integer   COM port. 3:COM3: 4:COM4; 5:COM5…
                            I-8417/8817/8437/8837: Valid ComPort No.= 3, 4 & 5
                            I-7188XG & I-7188EG: Valid ComPort No.= 4
RTS_   Boolean   TRUE: set RTS active, FALSE: set RTS inactive
DTR_   Boolean   TRUE: set DTR active, FALSE: set DTR inactive

Return :

Q_   Boolean   TRUE: Ok , FALSE: Fail
CTS_   Boolean   Get CTS status
DSR_   Boolean   Get DSR status

Note :
    1. Using Comopen( ) to open port.

COMARY_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
comary_r
─PORT_
─ARY_N  NUM─
```

Description : C_Function
 Read all bytes ready in the COM PORT buffer into a byte array
 (unsigned 8-bit)
Input Parameters :
 PORT_   Integer   Port number. I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                W-8xx7:2, 3, or …
 ARY_NO_   Integer   Byte array number (I-8xx7 & I-7188EG/XG:1~24;
                ISaGRAF WinCE PAC:1~48),
                to save bytes been read from COM
Return :
 NUM_   Integer   The number of bytes been read from COM port (0~256)
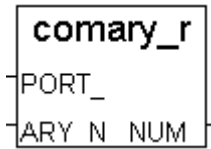
Note : Serial communication data need time to transfer. When COMARY_R is called, it will
 save the same number of bytes that have been received into the buffer at that time to the
 assigned array. The data are not received yet, of course are not saved into the array.

COMARY_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comary_w
─PORT_
─ARY_N
─NUM      Q─
```

Description : C_Function
 Write some bytes from one byte array to one COM PORT
 (unsigned 8-bit)
Input Parameters :
 PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                W-8xx7:2, 3, or …
 ARY_NO_   Integer   The byte array No. to write (I-8xx7 & I-7188EG/XG:1~24) ,
                (ISaGRAF WinCE PAC:1~48)
 NUM_   Integer   Number of bytes to write from the 1st unit of byte array (0~256)
Return :
 Q_   Boolean   TRUE: ok
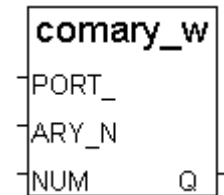
Note:
1 Before using COM1 of I-8xx7 & I-7188EG, please set COM1 as a non-Modbus-RTU port
   (Please refer to Appendix C.1)
2 Before using COM2 or COM3 of W-8xx7, please set COM2/3 as a non-Modbus-RTU port
   (Please refer to "Getting started of W-8xx7")
3   I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : not supported
Example:
   Please refer to Chapter 11 - Demo_21, 22 & 23.
   Refer to Appendix A.4 : "ARRAY_R" & "ARRAY_W"

COMAY_NW

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comay_nw
PORT_
ARY_N
NUM_
POS        Q
```
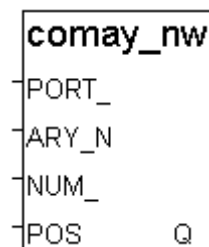
Description : C_Function

Write one Long Integer (signed 32 bit) array to COM PORT
Each Long Integer is composed of 4 bytes, and the format is a signed Long.

Each Integer written is composed of 4 bytes in the below INTEL format.
   [lowest byte] [ ] [ ] [highest byte]
For ex., if there are 3 Integers to write, the first one is 16#04030201 (67,305,985), the second one is 16#08070605 (134,678,021) and the last one is 16#FFFFFFFE (-2).
The 12 bytes been written will be [01] [02] [03] [04] [05] [06] [07] [08] [FE] [FF] [FF] [FF]

Input Parameters :

PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                       W-8xx7:2, 3, or …
ARY_NO_ Integer   The array No. to write (I-8xx7 & I-7188EG/XG:1~6) ,
                       (ISaGRAF WinCE PAC:1~18)

NUM_   Integer   Number of Integers to write (0~256)
POS_   Integer   Start position inside the array to write (1~256)
                       If POS_+NUM_ > 257, only (257~POS_) Long Integer will be written
                       Ex. If POS_=255, NUM_=3, only 2 integers written.
                       They are Pos. 255 & Pos. 256.

Return :

Q_   Boolean   TRUE means Ok, FALSE means fail.

Note :

1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Please refer to Appendix C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports.
   (Please refer to "The Getting started" of W-8xx7)
3. I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : do not support.
4. Integer array and word array use the same memory. Be careful if use both of them at the same time. (Refer to the description of Ary_N_W)

COMAY_WW

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comay_ww
PORT_
ARY_N
NUM_
POS      Q
```

Description : C_Function

Write some Words (signed 16 bit) of a Word array to COM PORT
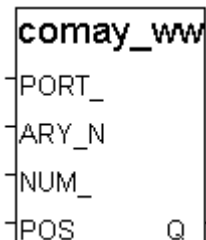
Each Word is composed of 2 bytes and as signed format (-32768 ~ +32767).
The word is wrote to COM PORT in the below INTEL format.
    [low byte] [high byte]
Ex. if there is 3 words to write, the 1st one is 16#0403 (1,027),
    the 2nd one is 16#0807 (2,055) and the last one is 16#FFFE (-2).
The 6 bytes been written will be [03] [04] [07] [08] [FE] [FF]

Input Parameters :

PORT_	Integer	Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8; W-8xx7:2, 3, or …
ARY_NO_	Integer	The word array No. to write (I-8xx7 & I-7188EG/XG:1~12); (ISaGRAF WinCE PAC:1~36)
NUM_	Integer	Number of Words to write
POS_	Integer	Start position inside the Word array to write (1~256) If POS_+NUM_ > 257, only (257~POS_) words will be written Ex: If POS_=255, NUM_=3, only 2 words will be written, They are Pos. 255 & Pos. 256.

Return :

Q_	Boolean	TRUE: OK; False: fail.

Note:

1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
    (Please refer to Appendix C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports.
    (Please refer to "The Getting started" of W-8xx7)
3. I-8xx7:
    ComPort No. on slot 0: Com5 ~ Com8
    ComPort No. on slot 1: Com9 ~ Com12
    ComPort No. on slot 2: Com13 ~ Com16
    ComPort No. on slot 3: Com17 ~ Com20
    ComPort No. on slot 4 ~ 7 : do not support.
4. Integer array and word array use the same memory. Be careful if using both of them at the same time. (Refer to the description of Ary_W_W)

COMCLEAR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 comclear
PORT_    Q
```
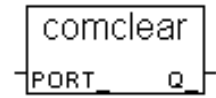
Description : C_Function
Clear the receiving buffer of COM PORT

Input Parameters :
PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                         W-8xx7:2, 3, or …
Return :
Q_   Boolean   TRUE: ok, False: fail.

COMCLOSE

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 comclose
PORT     Q
```

Description : C_Function
Close COM port

Input Parameters :
PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                         W-8xx7:2, 3, or …
Return :
Q_   Boolean   TRUE: ok; FALSE: fail.

Note:
1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Please refer to Appendix C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports.
   (Please refer to "The Getting started" of W-8xx7)
3. I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : do not support.

Example:
Please refer to the example of COMOPEN.

COMOPEN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  comopen
─PORT_
─BAUD_
─CHAR_
─PARI_
─STOP_    Q─
```

Description : C_Function
Open COM port
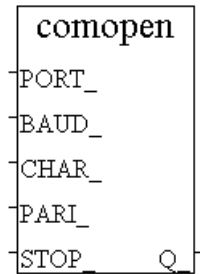
Input Parameters :

PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8;
                                   I-7188XG:2~8;   W-8xx7:2, 3, or …
BAUD_   Integer   Baud rate, range: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400,
                                   57600, 115200
CHAR_   Integer   Character size, 7 or 8
PARI_   Integer   Parity, 0: none; 1: even; 2: odd; 3: Mark; 4: Space.
                       (Parity of 3 & 4 is only for I-8xx7: COM3 to COM20,
                       I-7188EG/XG: COM3 to COM8 & W-8xx7:COM2, or ...)
STOP_   Integer   Stop bit, 1 or 2

### Return :
Q_   Boolean   TRUE: ok, FALSE: fail.

Note:
1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Please refer to Appendix C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports.
   (Please refer to "The Getting started" of W-8xx7)
3. I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : do not support.

Example:
Please refer to Chapter 11 - Demo_21, 22 & 23.

Open COM3. If success, it will return TRUE.

COMOPEN (3, 19200 , 8, 0, 1) ;

Write 1 byte (16#35) to COM3

Action (P) :
 TMP := COMWRITE (3, 16#35) ;
End_action ;

Close COM3, To next step.

COMCLOSE (3) ;

COMOPEN2

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comopen2
PORT_
BAUD_
CHAR_
PARI_
STOP_
FLOW      Q
```

Description : C_Function
Open COM PORT with flow control (RS232 only)

Input Parameters :

PORT_	Integer	Port number, I-7188EG/XG:3~8; W-8xx7:2, or …
BAUD_	Integer	Baud rate, range: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
CHAR_	Integer	Character size, 7 or 8
PARI_	Integer	Parity, 0: none, 1: even, 2: odd, 3: mark, 4: space. (Parity of 3 & 4 is only for I-8xx7: COM3 to COM20, I-7188EG/XG: COM3 to COM8 & W-8xx7:COM2, or ...)
STOP_	Integer	Stop bit, 1 or 2
FLOW_	Boolean	True: hardware flow control (CTS / RTS) (7188EG/XG 3 ~ 5), False: software flow control (XON / XOF) (7188EG/XG 3 ~ 8)

Return :

Q_	Boolean	TRUE: ok, FALSE: fail.

Note:
If use COM2 of W-8xx7, please preset it as a non-Modbus-RTU port (Refer to "The Getting Started" of W-8xx7)
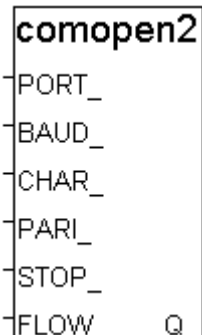
COMREAD

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comread
PORT   DATA
```

Description : C_Function
Read 1 byte (unsigned 8-bit) from COM port

Input Parameters :
PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
W-8xx7:2, 3, or …

Return :
**Q_**   Integer   The value of read byte (0~255)

Note:
1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Refer to section of C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports. (Refer
   to "The Getting started" of W-8xx7)
3. I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : do not support.

* Before using COMREAD to read from COM, please use COMREADY to test if there is data from COM.   If no data coming, use this function may cause Dead Lock.

Example:
Please refer to the example of "COMREADY".

COMREADY

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 comready
─PORT    Q─
```

Description : C_Function

Test if any data coming from COM PORT

Input Parameters :

PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                            W-8xx7:2, 3, or …

Return :

Q_   Boolean   TRUE means there is data coming (Even just 1 byte)

Note:

1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Refer to section of C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports. (Refer
   to "The Getting started" of W-8xx7)
3. I-8xx7:
   > ComPort No. on slot 0: Com5 ~ Com8
   > ComPort No. on slot 1: Com9 ~ Com12
   > ComPort No. on slot 2: Com13 ~ Com16
   > ComPort No. on slot 3: Com17 ~ Com20
   > ComPort No. on slot 4 ~ 7 : do not support.

* Before using COMREAD to read from COM, please use COMREADY to test if there is data from COM.   If no data coming, use this function may cause Dead Lock.

Example:

Please refer to Chapter 11 - Demo_21, 22 & 23.

Open COM3

`COMOPEN ( 3, 19200, 8, 0, 1 ) ;`

Test if any data coming from COM3

`COMREADY ( 3 ) ;`

Read 1 byte from COM3

```
Action ( P ) :
   VAL := COMREAD ( 3 )
End_action ;
```

To step 2
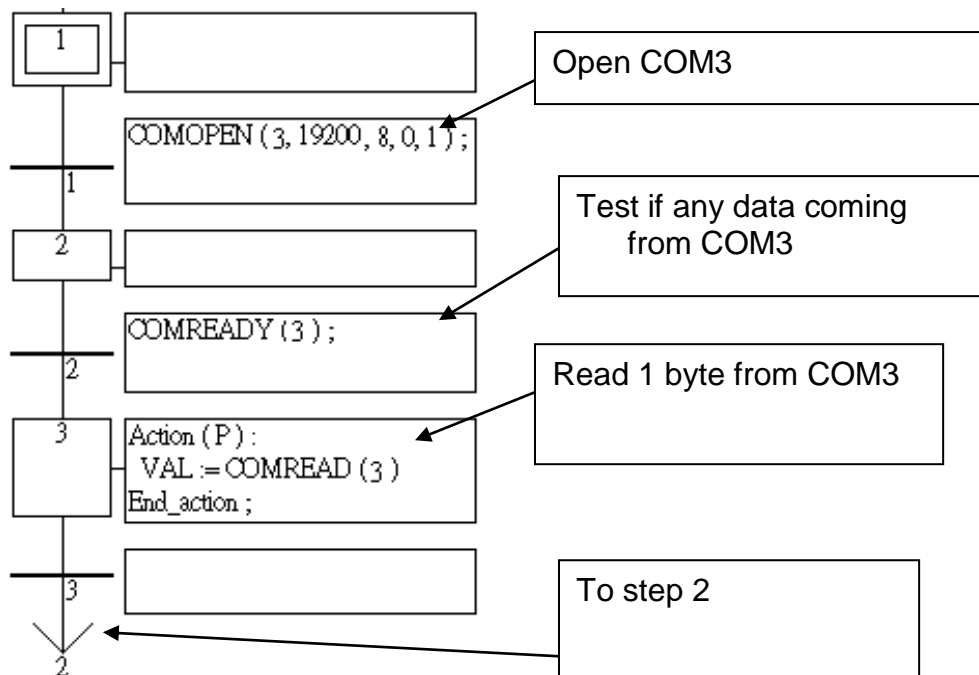
COMSTR_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comstr_w
PORT_
STR      Q
```

Description : C_Function

Write a String to COM port

Input Parameters :

PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
                              W-8xx7:2, 3, or …

STR_   Message   The String to write (Max. length: 255).

Return :

Q_   Boolean   TRUE: ok, FALSE: fail.

Note:

1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port. (Refer to section of C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports. (Refer to "The Getting started" of W-8xx7)
3. I-8xx7:
   - ComPort No. on slot 0: Com5 ~ Com8
   - ComPort No. on slot 1: Com9 ~ Com12
   - ComPort No. on slot 2: Com13 ~ Com16
   - ComPort No. on slot 3: Com17 ~ Com20
   - ComPort No. on slot 4 ~ 7 : do not support.

Example:

SW1 & TMP must be declared as internal Boolean

If SW1 is TRUE, continue.

Write 'Hello' to COM4.

If SW1 is FALSE, back to step 2.
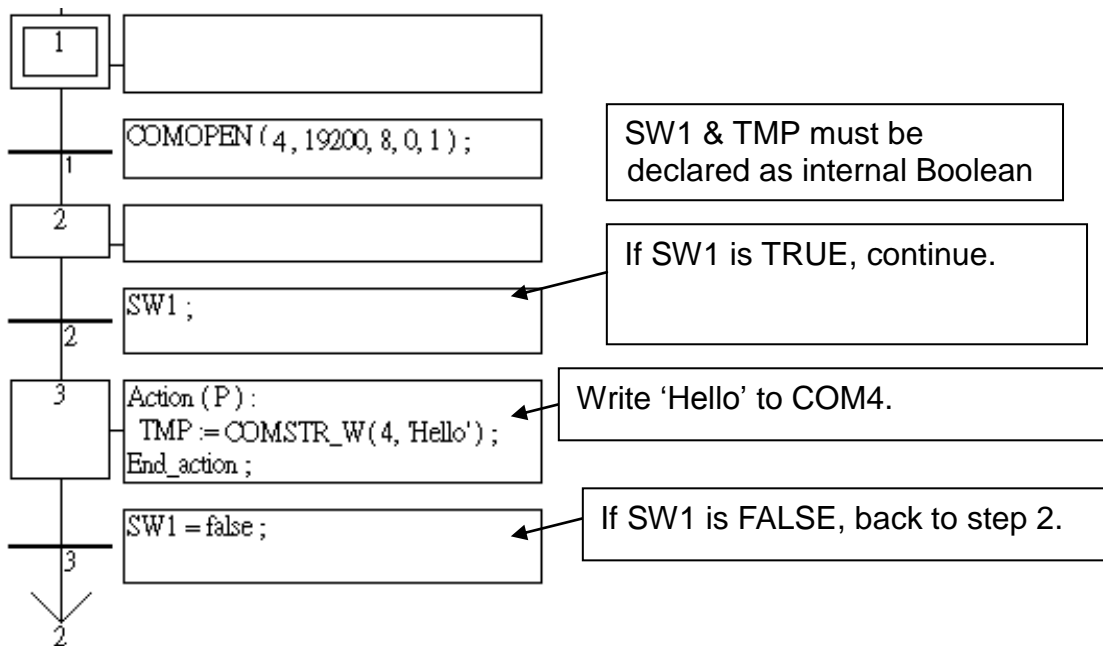
COMWRITE

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
comwrite
─PORT_
─DATA    Q─
```
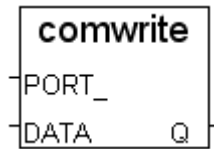
Description : C_Function
Write 1 byte (unsigned 8-bit) to COM port

Input Parameters :
PORT_   Integer   Port number, I-8xx7:1, 3 ~ 20; I-7188EG:1~8; I-7188XG:2~8;
W-8xx7:2, 3, or …

DATA_   Integer   The byte to write (0 ~ 255)

Return :
Q_   Boolean   TRUE: ok, FALSE: fail.

Note:
1. If use COM1 of I-8xx7 & I-7188EG, please preset COM1 as a non-Modbus-RTU port.
   (Refer to section of C.1)
2. If use COM2 or COM3 of W-8xx7, please preset them as non-Modbus-RTU ports. (Refer
   to "The Getting started" of W-8xx7)
3. I-8xx7:
   ComPort No. on slot 0: Com5 ~ Com8
   ComPort No. on slot 1: Com9 ~ Com12
   ComPort No. on slot 2: Com13 ~ Com16
   ComPort No. on slot 3: Com17 ~ Com20
   ComPort No. on slot 4 ~ 7 : do not support.

Example:

```
┌──────┐
│  1   │
└──────┘
   │
   ┤ COMOPEN ( 4 , 19200 , 8 , 0 , 1 ) ;
   1

┌──────┐
│  2   │
└──────┘
   │
   ┤ SW1 ;
   2

┌──────┐
│  3   │    Action ( P ) :
└──────┘      TMP := COMWRITE ( 4 , VAL ) ;
   │        End_action ;
   │        SW1 = false ;
   3
   │
   ∨
   2
```

SW1 & TMP must be internal Boolean; VAL must be internal integer

Give VAL a value (0~255), set SW1 to TRUE, it will write VAL in step 3

Write VAL to COM4

If SW1 is FALSE, back to step 2

CRC_16

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 crc_16
NUM_
ADR_    CR_H_
SIZE_   CR_L_
```
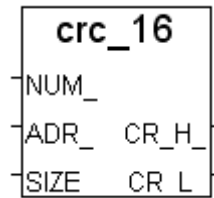
Use the algorithm of Section 5.3.

Description : C_Function Block

Calculate CRC_16 checksum value from byte array

Input Parameters :

NUM_   Integer   The byte array number to be calculated,
Range: I-8xx7 & I-7188EG/XG: 1~24; Wincon-8xx7: 1~48;
ADR_   Integer   Starting address to calculate in this array (1 ~ 256)
SIZE_   Integer   Total byte count to calculate

Return :

CR_H_   Integer   High byte of CRC-16 after calculation
CR_L_   Integer   Low byte of CRC-16 after calculation

Example:

TMP is declared to be internal Boolean. ii, CR_H_ & CR_L_ are internal integers, CRC16_1 is FB instance with type of CRC_16.

```
1   Action ( P ) :
      TMP := ARRAY_W ( 5, 3, 16#41 ) ;
      TMP := ARRAY_W ( 5, 4, 16#42 ) ;
      TMP := ARRAY_W ( 5, 5, 16#43 ) ;
      TMP := ARRAY_W ( 5, 6, 16#44 ) ;
    End_action ;
```

Store 4 Hex. Values (41, 42, 43, 44) to address 3 to 6 of array number 5.

```
      COMOPEN ( 3, 19200, 8, 0, 1 ) ;
1
```

```
2   Action ( P ) :
      FOR ii := 3 TO 6 DO
        TMP := COMWRITE( 3, ARRAY_R ( 5, ii ) ) ;
      END_FOR ;
      CRC16_1 ( 5, 3, 4 ) ;
      TMP := COMWRITE( 3, CRC16_1.CR_H_ ) ;
      TMP := COMWRITE( 3, CRC16_1.CR_L_ ) ;
    End_action ;
```

Read 4 bytes from address 3 to 6 of array number 5, and write to COM3.
Then Call CRC16_1 to calculate CRC-16 checking number (from the address 3 of array number 5, total are 4 bytes).
Then write the High & Low bytes of the result to COM3.

```
2
3
      GS3.t > T#1s ;
3

2
```

To step 2 after 1 sec.

DI_CNT

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        Di_Cnt
                    Q_
  RS1_          CN1_
  RS2_          CN2_
  RS3_          CN3_
  RS4_          CN4_
  RS5_          CN5_
  RS6_          CN6_
  RS7_          CN7_
  RS8_          CN8_
```

Description : C_Function Block

Get parallel D/I counter at slot 0 of I-8xx7 & I-7188EG/XG, or slot 1 of W-8xx7, please refer to Section 3.8

Input Parameters:

RS1_ ~ RS8_	Boolean	Reset the associated D/I counter to 0 when rising from False to True

Return:

Q_	Boolean	TRUE: work ok. FALSE: no parallel D/I modules is found at slot 0 (I-8xx7 & I-7188EG/XG) or slot 1 (W-8xx7).
CN1_ ~ CN8_	Integer	D/I Counter value of Ch.1 to Ch.8. (0 ~ 2,147,483,647), If Value is over 2,147,483,647, it restarts at 0.
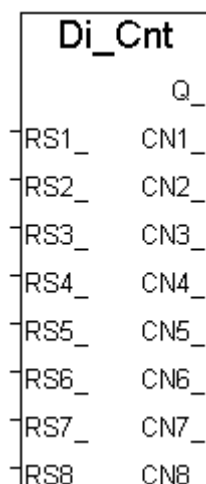
Note :

1. The related parallel D/I modules in I-8xx7 & I-7188EG/XG should be plugged at slot 0.
2. The related parallel D/I modules in W-8xx7/8xx6 should be plugged at slot 1.
3. Only Ch.1 to Ch.8 support Di_Cnt.
4. I-7188EG/XG must plug an Xxxx card at slot 0 then using "Di_Cnt" is possible.
5. The max frequency of counter input in I-8xx7 & I-7188EG/XG is up to 500 Hz with minimum pulse width of 1 ms.
6. The max frequency of counter input in W-8xx7 is up to 250 Hz with minimum pulse width of 2 ms.
7. Reset Counter channels: 8 , Reset the D/I counter when rising from False to True Counter value channels: 8 , valid value from 0 to 2,147,483,647 .

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

DT2MESAG

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        dt2mesag
  DAY_
  MONTH
  YEAR_
  HOUR_
  MINUT
  SECONMESSA
```

Description : C_Function

Convert Date & Time to Message

Input Parameters :

DAY_	Integer	Day (1~31)
MONTH_	Integer	Month (1~12)
YEAR_	Integer	Year, Ex. 01,02
HOUR_	Integer	Hour (0~23)
MINUTE_	Integer	Minute (0~59)
SECOND_	Integer	Second (0~59)

Return :

MESSAGE_:	Message	"day.month.year hours:minutes:seconds", Ex: "20.01.07 11:05:40"
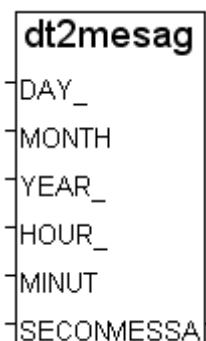
Note :

If input parameter is not correct, the return Message will be ''. (Empty Message).

EBUS_B_R

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
ebus_b_r
           B1_
           B2_
           B3_
           B4_
           B5_
           B6_
           B7_
  PACK     B8_
```

Description : C_Function Block
Read a Boolean package from the Ebus device

Input Parameters:
PACK_   Integer   Which package No. to read
                               I-8xx7, I-7188EG/XG: 1 ~ 128
                               ISaGRAF WinCE PAC: 1~256
Return :
B1_ ~ B8_   Boolean   The 8 Boolean values contained in the package
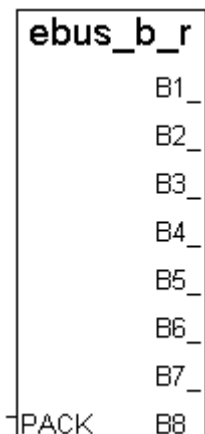
Note: Refer to Section 7.5

EBUS_B_W

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
ebus_b_w
  PACK_
  B1_
  B2_
  B3_
  B4_
  B5_
  B6_
  B7_
  B8_        Q
```

Description : C_Function Block
Write a Boolean package to the Ebus device

Input Parameters:
PACK_   Integer   Which package No. to write on
                               I-8xx7, I-7188EG/XG: 1 ~ 128
                               ISaGRAF WinCE PAC: 1~256
B1_ ~ B8_   Boolean   The 8 Boolean values contained in the package
Return :
Q   Boolean   Return TRUE always.

Note: Refer to Section 7.5

EBUS_F_R

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
ebus_f_r
        F1_
        F2_
        F3_
        F4_
        F5_
        F6_
        F7_
PACK    F8_
```
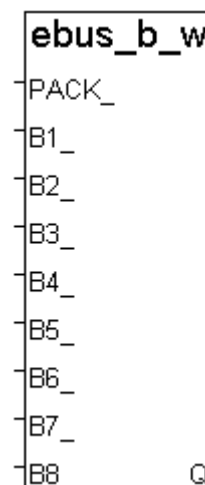
Description : C_Function Block
        Read a Real package from the Ebus device

Input Parameters:
        PACK_NO_   Integer   which package No. to read
                                                I-8xx7, I-7188EG/XG : 1 ~ 128
                                                ISaGRAF WinCE PAC : 1~256
Return :
        F1_ ~ F8_   Real   The 8 Real values contained in the package
                                                If any wrong, return 1.23E-20.
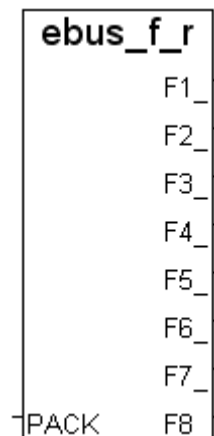
Note:
1. "EBUS_F_R" & "EBUS_N_R" use the same memory space.
2. Please **DO NOT** use INTEGER and REAL with the same package No., or it will return error.
    It may cause the error: "ERROR 115: EBUS_F_R Float error".

EBUS_F_W

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
ebus_f_w
PACK_
F1_
F2_
F3_
F4_
F5_
F6_
F7_
F8_        Q
```

Description : C_Function Block
        Write a Real package to the Ebus device

Input Parameters:
        PACK_   Integer   Which package No. to write on
                                        I-8xx7, I-7188EG/XG: 1 ~ 128
                                        ISaGRAF WinCE PAC: 1~256
        F1_ ~ F8_   Real   The 8 Real values contained in the package
Return :
        Q   Boolean   Return TRUE always.
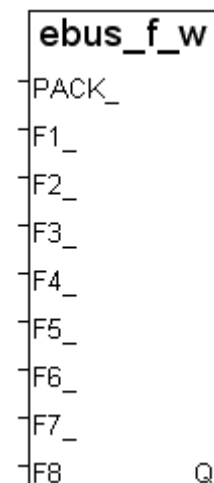
Note:
    1. "EBUS_F_W" & "EBUS_N_W" use the same memory space.
    2. Please **DO NOT** use INTEGER and REAL with the same package number, or it will
        return error. It may cause the error: " ERROR 115: EBUS_F_R float error".
    3. For using EBUS_F_R and EBUS_F_W, the driver must be the version listed below or
        higher version:
                I-7188EG : since ver. 2.17
                I-7188XG : since ver. 2.15
                I-8xx7   : since ver. 3.19
                W-8xx7   : recommend upgrade to ver. 3.36 or higher version

EBUS_N_R

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function Block
       Read an Integer package from the Ebus device
Input Parameters:
    PACK_   Integer   Which package number to read.
                      I-8xx7, I-7188EG/XG : 1 ~ 128
                      ISaGRAF WinCE PAC : 1~256

Return :
    N1_ ~ N8_   Integer   The 8 integer values contained in the package
Note: refer to Section 7.5.
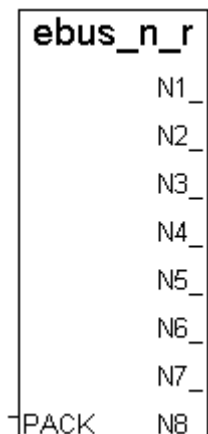
EBUS_N_W

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function Block
       Write an Integer package to the Ebus device,
Input Parameters:
    PACK_   Integer   Write which package No
                      I-8xx7, I-7188EG/XG : 1 ~ 128
                      ISaGRAF WinCE PAC : 1~256
    N1_ ~ N8_   Boolean   The 8 integer values contained in the package
Return :
    Q   Boolean   Return TRUE always.
Note: Refer to Section 7.5
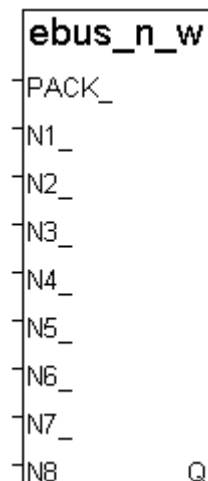
EBUS_STS

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function
       Get package status of Ebus
Input Parameters:
    ID_   Integer   Which package type? (0~1)
                      0: Boolean package, 1: Integer package
    PACK_   Integer   Which package No?
                      I-8xx7, I-7188EG/XG: 1 ~ 128
                      ISaGRAF WinCE PAC: 1~256

Return :
    Q_   Boolean   TRUE: Package is alive, FALSE: Package is dead
                        Reasons may cause the package dead: Ebus_m controller did not enable the package number, communication breakdown, the controller for sending that package is dead, Ebus_m controller is dead, and so on.
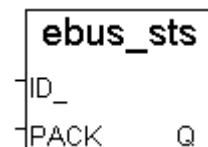
EEP_B_R

```
eep_b_r
ADR       Q
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
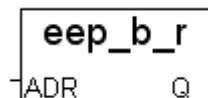
Description : C_Function
Read a Boolean from EEPROM

Input Parameters :
**ADR_**   Integer   Which address to read
                                    I-8xx7 & I-7188EG/XG: (1 ~ 256) ,
                                    ISaGRAF WinCE PAC: (1 ~ 1024)
                                    µPAC-7186EG, µPAC -5xx7, iP-8xx7, VP-2117: (1 ~ 1024)

Return :
Q_   Boolean   The Boolean returned.

EEP_B_W

```
eep_b_w
ADR_
DATA      Q
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function
Write 1 Boolean to EEPROM

Input Parameters :
ADR_   Integer   The address be write to
                                      I-8xx7 & I-7188EG/XG: (1 ~ 256) ,
                                      ISaGRAF WinCE PAC: (1 ~ 1024)
                                      µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117: (1 ~ 1024)

DATA_   Boolean   The Boolean to be write
Return :
Q_   Boolean   TRUE: ok.

Note:
1. There is no operation limitation for reading from EEPROM.
2. There is some operation limitation for writing to EEPROM. (Refer to Section 10.2)
3. Before writing to EEPROM, user must use EEP_EN to remove the EEPROM write protection.
4. To read/write EEPROM will consume lots of CPU time, and increase the PLC Scan Time a lot. Please use these functions very carefully.

Example: Refer to demo_17.

EEP_BY_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
     Read one Byte (unsigned 8-bit) from EEPROM

```
eep_by_r
ADR      Q
```

Input Parameters :
    ADR_   Integer   The address to read from
                           I-8xx7 & I-7188EG/XG: (1 ~ 1512);
                           ISaGRAF WinCE PAC: (1 ~ 6144);
                           µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 14272)
Return :
    Q_   Integer   The byte returned (0~255)

EEP_BY_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
     Write one Byte (unsigned 8-bit) to EEPROM

```
eep_by_w
ADR_
DATA     Q
```

Input Parameters :
    ADR_   Integer   The address be write to
                           I-8xx7 & I-7188EG/XG: (1 ~ 1512);
                           ISaGRAF WinCE PAC: (1 ~ 6144);
                           µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 14272)

    DATA_   Integer   The byte to be write (0 ~ 255)
Return :
    Q_   Boolean TRUE: ok.

EEP_EN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
     Remove the EEPROM write protection

```
eep_en
         Q
```
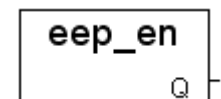
Return :
    Q_   Boolean   TRUE: ok.
Note :
  1. EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R
     & EEP_F_W functions use the same memory space, please be very careful when using
     them. The address numbers listed below use the same memory space:

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

  2. There is some operation limitation for writing to EEPROM. (Refer to Section 10.2)
  3. Before writing to EEPROM, user must use EEP_EN to remove the EEPROM write
     protection.
  4. To read/write EEPROM will consume lots of CPU time, and increase the PLC Scan Time
     a lot. Please use these functions very carefully.
Example: Please refer to Chapter 11: demo_17.

EEP_F_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  eep_f_r
ADR       Q
```

Description : C_Function

Read one Real from EEPROM

Input Parameters :

ADR_   Integer   Address to read (use the same EEPROM address as EEP_N_R)
                        Range: I-8xx7 & I-7188EG/XG: (1 ~ 378);
                               ISaGRAF WinCE PAC: (1 ~ 1536);
                        µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 3568).

Return :

Q_   Real   The Real returned. If ADR_ is not in the range, Q_ = 1.23E-20.
                        If the value stored in EEPROM is not a REAL, Q_ will be wrong,
                        sometimes got the error: "ERROR 114: EEP_F_R float error".

Important :

1. "EEP_F_R" & "EEP_N_R" use the same EEPROM address
2. Please **DO NOT** use INTEGER and REAL in the same EEPROM address, it will return error.
   It may cause the error: " ERROR 114: EEP_F_R   float error".
3. To read/write EEPROM will consume lots of CPU time, especially the WRITE operation.
   Please use these functions very carefully.

Note :

1. This function can be used even without calling the EEP_EN function.
2. EEP_BY_R, EEP_BY_W, EEP_WD_R, EEP_WD_W, EEP_N_R, EEP_N_W, EEP_F_R &
   EEP_F_W functions use the same memory space. Please be very careful when using them.
   Ex: ADR_ 2 of EEP_N_R use 4 bytes. It is the same memory space with the ADR_ 3, 4 of
   EEP_WD_R and the ADR_ 5, 6, 7, 8 of EEP_BY_R.
3. W-8xx7/8xx6 use the 16th ~ 31st segments of EEPROM to store Boolean (64 bytes per
   segment), 32nd ~ 254th segments to store Byte, Word and Long. The 0th ~ 15th segments are
   no use. The 255th segment is reserved.
4. For using EEP_F_R and EEP_F_W, the driver must be the version listed below or higher
   version:
              I-7188EG : since ver. 2.17
              I-7188XG : since ver. 2.15
              I-8xx7   : since ver. 3.19
              W-8xx7   : recommend upgrade to ver. 3.36 or higher version

EEP_F_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 eep_f_w
ADR_
DATA      Q
```
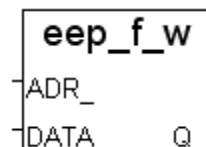
Description : C_Function
      Write one Real to EEPROM

Input Parameters :
    ADR_   Integer   Address be write to (use the same EEPROM address as EEP_N_W)
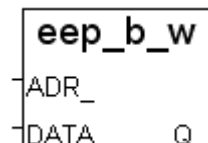   Range: I-8xx7 & I-7188EG/XG: (1 ~ 378);
                                    ISaGRAF WinCE PAC: (1 ~ 1536);
                            μPAC-7186EG, μPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 3568).
    DATA_   Real   The written Real
Return :
    Q_   Boolean   TRUE: ok, FALSE: fail.

Importtant :

1. "EEP_F_W" & "EEP_N_W" use the same EEPROM address
2. Please **DO NOT** use INTEGER and REAL in the same EEPROM address, it will return error.
   It may cause the error: " ERROR 114: EEP_F_R   float error"
3. To read/write EEPROM will consume lots of CPU time, especially the WRITE operation.
   Please use the functions very carefully.

Note :

1. User must use EEP_EN to remove the EEPROM write protection before writing to EEPROM.
2. If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_BY_R, and
   EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because
   they all occupy the same memory area. For example, ADR_2 of EEP_N_R occupies 4 bytes,
   and it uses the same memory area as ADR_3 and ADR_4 of EEP_WD_R and the same
   address of ADR_5, 6, 7, and 8 of EEP_BY_R.
3. The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to
   write to the EEPROM more than 100,000 times. Over writing will damage the EEPROM.
4. Before "write", call EEP_EN( ) to remove the EEPROM write portection.
5. After "write", you may call EEP_PR( ) to protect the EEPROM.
6. W-8xx7/8xx6 use the 16th ~ 31st segments of EEPROM to store Boolean (64 bytes per
   segment), 32nd ~ 254th segments to store Byte, Word and Long. The 0th – 15th segments are
   no use. The 255th segment is reserved.
7. For using EEP_F_R and EEP_F_W, the driver must be the version listed below or higher
   version:
             I-7188EG : since ver. 2.17
             I-7188XG : since ver. 2.15
             I-8xx7   : since ver. 3.19
             W-8xx7   : suggest upgrade to ver. 3.36 or higher version

EEP_N_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function

read a signed 32-bit integer value from the EEPROM

```
      eep_n_r
 ─┤ADR      Q├─
```
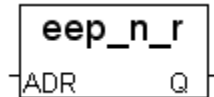
Input Parameters :

ADR_   Integer   Address in the EEPROM where the 32-bit Integer value is stored.
Range: I-8xx7 & I-7188EG/XG: (1 ~ 378);
ISaGRAF WinCE PAC: (1 ~ 1536);
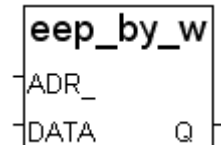µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 3568).

Return :

Q_   Integer   The signed 32-bit Integer value returned

Note:

1. If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_BY_R, and
   EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because
   they all occupy the same memory area.

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

2. There is no number of times limitation for reading from EEPROM. User can use EEP_N_R to
   read from EEPROM whatever using or not using EEP_EN to remove the protection of
   EEPROM

3. There is number of times limitation for writing to EEPROM. (Refer to Section 10.2)

Example: Please refer to Chapter 11: demo_17

EEP_N_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
eep_n_w
ADR_
DATA      Q
```
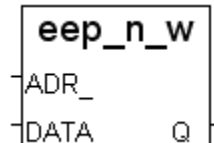
Description : C_Function
> Write a signed 32-bit Integer value to the EEPROM

Input Parameters :
    ADR_   Integer   Address in EEPROM where the 32-bit Integer value will be written to
                               Range: I-8xx7 & I-7188EG/XG: (1 ~ 378);
                                         ISaGRAF WinCE PAC: (1 ~ 1536);
                               µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 3568).
    DATA_   Integer   The 32-bit Integer value to be written to
Return :
    Q_   Boolean   TRUE: ok.

Note:
1. If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_BY_R, and EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because they all occupy the same memory area.

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

2. There is number of times limitation for writing to EEPROM. (Refer to Section 10.2)
3. Before writing, please call EEP_EN to remove the EEPROM write protection.
4. To read/write EEPROM will consume lots of CPU time, and may increase the PLC Scan time a lot. Please use the functions very carefully.

Example: Please refer to demo_17 of Chapter 11.

EEP_PR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
eep_pr
          Q
```
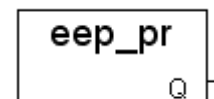
Description : C_Function
> Protect EEPROM, write operation is not allowed

Return :
    Q_   Boolean   TRUE: ok

Note :
* There is number of times limitation for writing to EEPROM. (Refer to Section 10.2)
* Before writing, call EEP_EN to remove the EEPROM write protection.

Example: Please refer to demo_17 of Chapter 11.

EEP_WD_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
eep_wd_r
ADR      Q
```
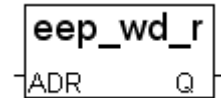
Description : C_Function
Read a Word (signed 16-bit Integer) value from the EEPROM

Input Parameters :
ADR_   Integer   Address in the EEPROM where the word value is stored
                                I-8xx7 & I-7188EG/XG: (1 ~ 756);
                                ISaGRAF WinCE PAC: (1 ~ 3072);
                                µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 7136).

Return :
Q_   Integer   The returned Word value (-32768 ~ +32767)

EEP_WD_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
eep_wd_w
ADR_
DATA     Q
```
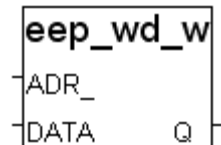
Description : C_Function
Write a Word (signed 16-bit Integer) value to the EEPROM

Input Parameters :
ADR_   Integer   Address in the EEPROM where the Word value is to be written to.
                                I-8xx7 & I-7188EG/XG: (1 ~ 756);
                                ISaGRAF WinCE PAC: (1 ~ 3072)
                                µPAC-7186EG, µPAC-5xx7, iP-8xx7, VP-2117, W-8xx7: (1 ~ 7136).

DATA_   Integer   The word value to be written to (-32768 ~ 32767)

Return :
Q_   Boolean   TRUE: ok.

Note:
1. If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_BY_R, and EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because they all occupy the same memory area.

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

2. There is number of times limitation for writing to EEPROM. (Refer to Section 10.2)
3. Before writing, please call EEP_EN to remove the EEPROM write protection.
4. To read/write EEPROM will consume lots of CPU time, and may increase the PLC Scan time a lot. Please use the functions very carefully.

Example: Please refer to demo_17 of Chapter 11.

F_APPEND

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
f_append
SRC_
DES        Q
```
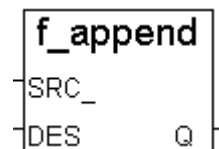
Description : C_Function
Append one file to the other file

Input Parameters :
SRC_   Message   File path of source file. Ex: '\Micro_SD\data.txt'
DES_   Message   File path of destination file. Ex: '\Micro_SD\data1.txt'

Return :
Q_   Boolean   True: Ok, False: Fail

Note:
1. If one of these two files is not found, return FALSE.
2. Source and destination file should be closed, not opened.
3. Please refer to ISaGRAF standard function F_wopen , F_ropen , F_close , F_eof ,
   Fa_read , Fa_write (F_wopen: Open existing file for Read & Write)
4. Please refer to ICP DAS functions for ISaGRAF: F_creat, F_copy, F_append, F_dir, F_end,
   F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
5. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and
   VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6)
   will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\ ,
   will be better, but it will lost the file content after turning off the controller.

F_CLOSE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
F_CLOSE
ID        ok
```
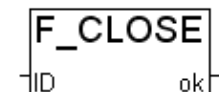
Description : Standard_Function
Close a Binary file opened with the function of F_ROPEN, F_WOPEN or F_CREAT.

Input Parameters :
ID   Integer   The file number returned by F_ROPEN, F_WOPEN or F_CREAT.

Return :
OK   Boolean   Return status
                 TRUE: file closed ok; FALSE: file closed fail.

Example :
```
(* ST Program: *)
file_id := F_ROPEN('\Micro_SD\data.bin');
ok := F_CLOSE(file_id);

(* IL Equivalence: *)
LD            '\Micro_SD\data.bin'
F_ROPEN
ST            file_id
F_CLOSE                    (* file_id is already in the current IL result *)
ST            ok
```

F_COPY

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 f_copy
SRC_
DES      Q
```
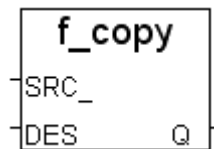
Description : C_Function
Copy one file to another one

Input Parameters :
SRC_ Message File path & name of source file. Ex: '\Micro_SD\data.txt'
DES_ Message File path & name of destination file. Ex: '\Micro_SD\data1.txt'
Return :
Q_ Boolean True: Ok, False: Fail

Note:
1. Copy the SRC_ file to the DES_ file.
2. SRC_ file and DES_ file should be closed, but not opened.
3. If there is an old DES_ file existing, it will be replaced by new contents.

F_CREAT

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 f_creat
PATH    ID
```
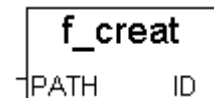
Description : C_Function
Create an empty file for reading & writing.

Input Parameters :
Path_ Message File path & name. Ex: '\Micro_SD\data.txt'
Return :
ID_ Integer File ID returned. If error happens, it returns 0.

Note:
1. If file exists already, it will be destroyed.
2. For reading an existing file, please use the ISaGRAF standard function – "F_ROPEN( )"
3. For writing an existing file, please use the ISaGRAF standard function – "F_WOPEN( )"
4. Please refer to ISaGRAF standard functions: F_wopen, F_ropen, F_close, F_eof, Fa_read,
   Fa_write
5. Please refer to ICP DAS functions for ISaGRAF: F_creat, F_copy, F_append, F_dir, F_end,
   F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w.
6. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and
   VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6)
   will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\,
   will be better, but it will lost the file after turning off the controller.

Example: Wpdmo_11 in WP-8xx7 CD-ROM:\napdos\isagraf\wp-8xx7\demo\ .

F_DELETE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
    Delete a file

```
 ┌─────────────┐
 │  f_delete   │
 │             │
─┤name      Q ├─
 └─────────────┘
```
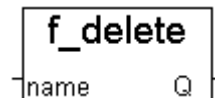
Input Parameters :
    Name_ Message   File name. Ex: '\Micro_SD\data.txt'
Return :
    Q_   Boolean   True: Ok,   False: fail

F_DIR

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
    Create a new directory

```
 ┌─────────────┐
 │    f_dir    │
 │             │
─┤DIR       Q ├─
 └─────────────┘
```
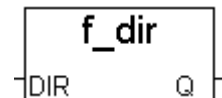
Input Parameters :
    Dir_   Message   Directory name. Ex: **'\DATA21'**
Return :
    Q_   Boolean   True: Ok,   False: fail (Ex: Directory already exists)

Example:
    (* init is declared as an internal Boolean variable with initial value = True *)
    (* tmp is declared as an internal Boolean variable *)

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

F_END

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
    Move file position to End-Of-File

```
 ┌─────────────┐
 │   f_end     │
 │             │
─┤ID        Q ├─
 └─────────────┘
```
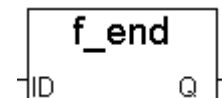
Input Parameters :
    ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
Return :
    Q_   Boolean   True: Ok,   False: Fail
Note:
1. Please use F_SEEK to move to a specified file position
2. Please refer to ISaGRAF standard functions: F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write
3. Please refer to ICP DAS functions for ISaGRAF: F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w
4. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

F_EOF

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : Standard_Function

Test if End Of File has been reached
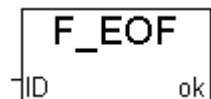
```
┌─────────────┐
│   F_EOF     │
│ID        ok │
└─────────────┘
```

Input Parameters :

ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)

Return :

OK   Boolean   True: reach End Of File, False: not yet

F_EXIST

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Test if a directory or a file exists.

```
┌─────────────────┐
│     F_EXIST     │
│en            Q_ │
│                 │
│Name             │
└─────────────────┘
```
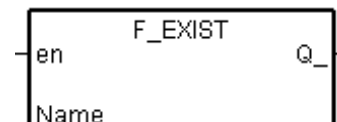
Call:

Name_   message   directory path name or file name.
For ex '\Micro_SD\Dir2' , '\Micro_SD\DTAT1.txt'

Return:

Q_   boolean   TRUE: exist ,   FASLE: not found.

Example:

```
(* init is declared as an internal boolean variable with initial value = True *)
(* tmp is declared as an internal boolean variable *)
 if  init  then
    init := False ;
    tmp := f_exist('\Micro_SD\Dir2');  (* find a directory or a file *)
    if  tmp = False  then
      (* not found then create it *)
      tmp := f_dir('\Micro_SD\Dir2') ;  (* create a directory *)
    end_if ;
 end_if;
```

Note:
1. Please refer to ISaGRAF standard function F_wopen , F_ropen , F_close , F_end , Fa_read , Fa_write.
2. Please refer to F_creat , F_copy , F_append , F_dir , F_end , F_seek , F_writ_b , F_writ_f , F_writ_s , F_writ_w.

F_READ_B

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
    Read one Byte from current position of an open file

```
 F_READ_B
─ID      Q ─
```

Input Parameters :
    ID_   Integer   File ID No., (returned by F_ROPEN( ) , F_CREAT( ) or F_WOPEN( ))

Return :
    Q_   Integer   The returned Byte (0 ~ 255)

F_READ_F

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 f_read_f
─ID      Q ─
```
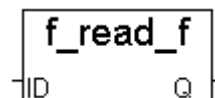
Description : C_Function
    Read one Real (32-bit Float) value from current position of an open file

Input Parameters :
    ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)

Return :
    Q_   Real   The returned Real value (32-bit Float format)
                    If the value is not 32-bit Float format, the returned value will be error.
                    It may cause: "ERROR 117: F_READ_ error".

Note:

1. Please use the ISaGRAF standard function –"FA_READ" & "FA_WRITE" to read/write the Long Integer (signed 32-Bit).

2. Please use the ISaGRAF standard function –"FM_READ", "FM_WRITE" or "F_writ_s" to read/write String.

3. Refer to the usage of other ISaGRAF standard functions – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write.

4. Refer to the usage of ICP DAS functions for ISaGRAF – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w.

5. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

Example: Refer to WP-8xx7 CD:\napdos\isagraf\wp-8xx7\demo\ "wpdmo_01" & "wpdmo_02

F_READ_W

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
f_read_w
ID        Q
```
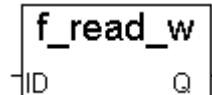
Description : C_Function
    Read one Word (signed 16-bit Integer) from current position of an open file.
Input Parameters :
    ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
Return :
    Q_   Integer   The returned word (-32768 ~ +32767)

F_ROPEN

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
F_ROPEN
Path        ID
```

Description : Standard_Function
    Open a Binary file in read mode.

Input Parameters :
    Path_   Message   File name
                        It may include the access path to the file using the \ or / symbol to
                        specify a directory. To ease application portability, / or \ is equivalent.
Return :
    ID_   Integer   File number
                        0: error occurs or file does not exist.

Example :
```
  (* ST program: *)
 file_id := F_ROPEN(' \Micro_SD\data.bin ');
 error := (file_id=0);

 (* IL Equivalence: *)
 LD        '\Micro_SD\data.bin'
 F_ROPEN
 ST        file_id
 EQ        0
 ST        error
```

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and
    VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6)
    will consume much more CPU time. To do the file operation in RAM Disk, such as in
    \Temp\, will be better, but it will lost the file after turning off the controller.

F_SEEK

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG □ I-7188XG
■ ISaGRAF WinCE PAC
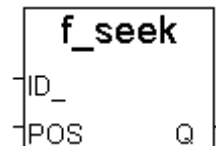
Description : C_Function
          Move file position to a new position...…
Input Parameters :
   ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
   POS_   Integer   Position, unit is Byte (1 to ...)
Return :
   Q_   Boolean   True: ok. False: fail

F_TRIG

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
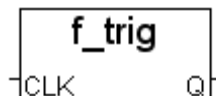
Description : Standard_Function
          Detect a falling edge of a Boolean variable
Input Parameters :
   CLK_   Boolean   Any Boolean variable
Return :
   Q_   Boolean   TRUE: if CLK change from TRUE to FALSE
                             FALSE: all other cases

Example :
   (* FBD Program *)
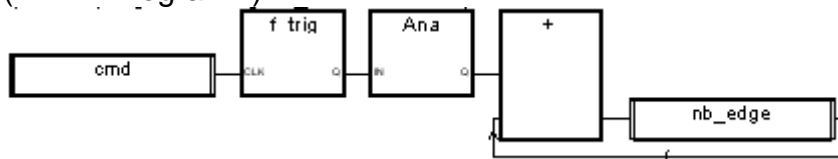
   (* ST Equivalence: We suppose F_TRIG1 is an instance F_TRIG block *)
   F_TRIG1(cmd);
   nb_edge := ANA(F_TRIG1.Q) + nb_edge;

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

F_WOPEN

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
F_WOPEN
Path        ID
```
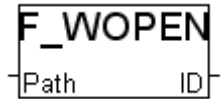
Description : Standard_Function

Open a Binary file in Write mode, then the file can be read/write.

Input Parameters :

Path   Message   File name

It may include the access path to the file using the \ or / symbol to specify a directory. To ease application portability, / or \ is equivalent.

Return :

ID   Integer   File number

0: error occurs. If file exists already, it will be overwritten.

Example :

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


(* IL Equivalence: *)
LD          '\Micro_SD\hello.dat'
F_WOPEN
ST          file_id
EQ          0
ST          error
```

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

F_WRIT_B

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
f_writ_b
─ID_
─IN        Q ─
```
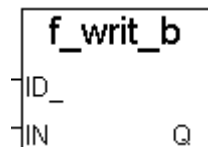
Description : C_Function
Write one Byte to current position of an open file

Input Parameters :
ID_	Integer	File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
IN_	Integer	Byte value to write (0 ~ 255), if value > 255 or <0, the lowest byte is written

Return :
Q_	Boolean	True: ok. False: fail

Note:
1. Using ISaGRAF standard function –"FA_READ" & "FA_WRITE" to read/write Long Integer (signed 32-bit)
2. Using ISaGRAF standard function –"FM_READ" & "FM_WRITE" or "F_writ_s" to read/write String.
3. Refer to the usage of ISaGRAF standard function – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write.
4. Refer to the ICP DAS functions for ISaGARF – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w.
5. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

Example: Refer to WP-8xx7 CD:\napdos\isagraf\wp-8xx7\demo\ "wpdmo_01" & "wpdmo_02"

F_WRIT_F

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
f_writ_f
─ID_
─IN        Q ─
```
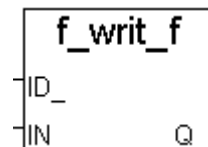
Description : C_Function
Write one Float value (32-bit format) to current position of an open file

Input Parameters :
ID_	Integer	File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
IN_	Real	Float value to write (32-bit format)

Return :
Q_	Boolean	True: ok. False: fail

F_WRIT_S

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 f_writ_s
─┤ID_
 ┤STR      Q├
```
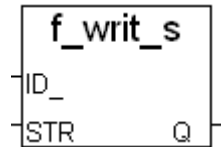
Description : C_Function

Write one String to current position of a file without <CR> <LF> at the end

Input Parameters :

ID_   Integer   File ID No. (returned by F_ROPEN , F_WOPEN or F_CREAT)
STR_   Message   Message (String) to write

Return :

Q_   Boolean   True: ok. False: fail

Note:

1. "**FM_write**" writes <CR> <LF> at the end of the message, but "**F_writ_s**" doesn't.
2. Using ISaGRAF Standard Function – "**FA_READ**" & "**FA_WRITE**" to R/W Long Integer (signed 32-bit).
3. Using ISaGRAF Standard Function – "FM_READ" or "FM_WRITE" to R/W Message (String).
4. Refer to the usage of ISaGRAF standard function – F_wopen, F_ropen, F_close, F_eof, Fa_read, Fa_write.
5. Refer to the ICP DAS functions for ISaGARF – F_creat, F_copy, F_append, F_dir, F_end, F_seek, F_writ_b, F_writ_f, F_writ_s, F_writ_w.
6. To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

Example:

Please refer to WP-8xx7 CD:\napdos\isagraf\wp-8xx7\demo\ "wpdmo_01" & "wpdmo_02"

F_WRIT_W

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 f_writ_w
─┤ID_
 ┤IN       Q├
```
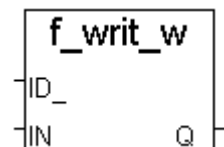
Description : C_Function

Write one Word value (signed 16-bit Integer) to current position of an open file.

Input Parameters :

ID_   Integer   File ID No., (returned by F_ROPEN , F_WOPEN or F_CREAT)
IN_   Integer   Word to write (-32768 ~ +32767)

Return :

Q_   Boolean   True: ok. False: fail

FA_READ

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
FA_READ
ID        Q
```
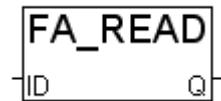
Description : Standard_Function

       Read one Integer value (32-bit signed) from a Binary file.

Input Parameters :
    ID   Integer   File ID No. (returned by F_ROPEN, F_WOPEN or F_CREAT)
Return :
    Q   Integer   Integer value read from file

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

(* IL Equivalence: *)
LD        '\Micro_SD\data.dat'
F_ROPEN
ST        file_id
LD        file_id
FA_READ               (* read vinc *)
ST        vinc
LD        file_id
FA_READ               (* read timer: delta_tim *)
TMR                   (* convert into a timer *)
ST        delta_tim
LD        file_id
F_CLOSE
ST        ok
```

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

FA_WRITE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
FA_WRITE
ID
IN        ok
```
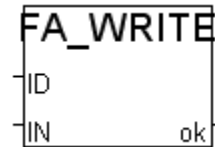
Description : Standard_Function
Write one Integer value (32-bit signed) to a Binary file.

Input Parameters :
ID	Integer	File ID No. (returned by F_ROPEN, F_WOPEN or F_CREAT).
IN	Integer	Integer value be write to file

Return :
OK	Boolean	execution status. TRUE: ok

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

(* IL Equivalence: Please refer to the menu of ISaGRAF Projects: Help/language Reference*)

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

FBUS_B_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
                                    ┌─────────────┐
                                    │  Fbus_b_r   │
                                    │          B1_│
                                    │          B2_│
                                    │          B3_│
                                    │          B4_│
                                    │          B5_│
                                    │          B6_│
                                    │          B7_│
                                    │PACK_     B8_│
                                    └─────────────┘
```

Description : C_Function Block
      Read a Boolean package from the Fbus device

Input Parameters :
    PACK_   Integer   Which package No. to read (1 ~ 128)
Return :
    B1_ ~ B8_ Boolean   The 8 Boolean values contained in the package

Example: Please refer to Chapter 7 or demo_11a & demo_11b

FBUS_B_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
                                    ┌─────────────┐
                                    │  Fbus_b_w   │
                                    │PACK_        │
                                    │B1_          │
                                    │B2_          │
                                    │B3_          │
                                    │B4_          │
                                    │B5_          │
                                    │B6_          │
                                    │B7_          │
                                    │B8_         Q│
                                    └─────────────┘
```
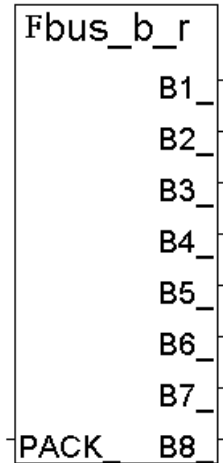
Description : C_Function Block
      Write a Boolean package to the Fbus device

Input Parameters :
    PACK_   Integer   Write which package No., 1 ~ 128
    B1_ ~ B8_ Boolean   the 8 Boolean values in the package to write
Return :
    Q_   Boolean   Return TRUE only.

Example: Please refer to Chapter 7or demo_11a & demo_11b
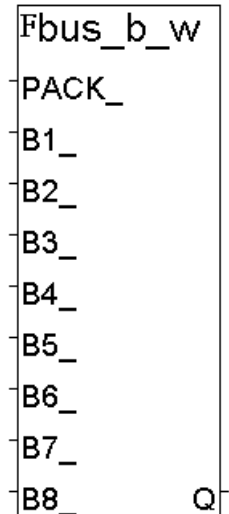
FBUS_F_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

Description : C_Function Block

      Read an Real package from the Fbus device

Input Parameters :
    PACK_NO_   Integer   Read which package No. 1 ~ 128
Return :
    F1_ ~ F8_   Real   The 8 Real values contained in the package
                                      If error, return 1.23E-20.
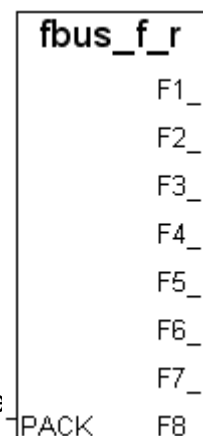
Important :
1. "FBUS_F_R" use the same memory area as "FBUS_N_R".
2. Please **DO NOT** use Integer and Real data in the same package number, the data may be error. It may cause: "ERROR 116:   FBUS_F_R   float error".
3. Please use "FBUS_N_R" & "FBUS_N_W" to transfer Integer value.

Example: Please refer to Chapter 7 or demo_11a & demo_11b

FBUS_F_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

Description : C_Function Block

      Write a Real package to the Fbus device

Input Parameters :
    PACK_NO_   Integer   Write which package No., 1 ~ 128
    F1_ ~ F8_   Real   The 8 Real values in the package to write
Return :
    Q_   Boolean Return TRUE only.
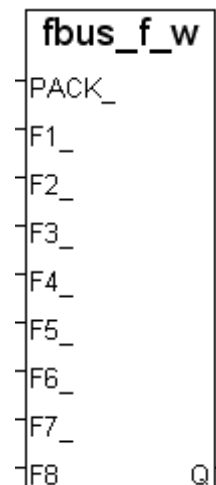
Important :
1. "FBUS_F_W" use the same memory area as "FBUS_N_W".
2. Please **DO NOT** use Integer and Real data in the same package number, the data may be error. It may cause: "ERROR 116:   FBUS_F_R   float error".
3. Please use "FBUS_N_R" & "FBUS_N_W" to transfer Integer value.
4. For using FBUS_F_R and FBUS_F_W, the driver must be the version listed below or higher version:
              I-7188EG : since ver. 2.17
              I-7188XG : since ver. 2.15
              I-8xx7   : since ver. 3.19

Example: Please refer to Chapter 7 or demo_11a & demo_11b

FBUS_N_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
┌─────────────┐
│  Fbus_n_r   │
│          N1_│
│          N2_│
│          N3_│
│          N4_│
│          N5_│
│          N6_│
│          N7_│
│PACK_     N8_│
└─────────────┘
```

Description : C_Function Block
       Read an Integer package (signed 32-bit) from the Fbus device

Input Parameters :
    PACK_   Integer   Read which package No., 1 ~ 128
Return :
    N1_ ~ N8_   Integer   The 8 Integer values contained in the package

Example: Please refer to Chapter 7 or demo_11a & demo_11b

FBUS_N_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
┌─────────────┐
│  Fbus_n_w   │
│PACK_        │
│N1_          │
│N2_          │
│N3_          │
│N4_          │
│N5_          │
│N6_          │
│N7_          │
│N8_        Q │
└─────────────┘
```
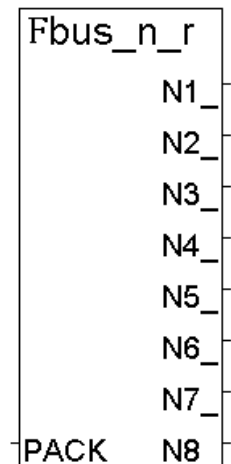
Description : C_Function Block
       Write an Integer package (signed 32-bit) to the Fbus device

Input Parameters :
    PACK_   Integer   Write which package No., 1 ~ 128
    N1_ ~ N8_   Boolean   The 8 Integer values contained in the package
Return :
    Q_   Boolean   Return TRUE always.

Example: Please refer to Chapter 7 or demo_11a & demo_11b
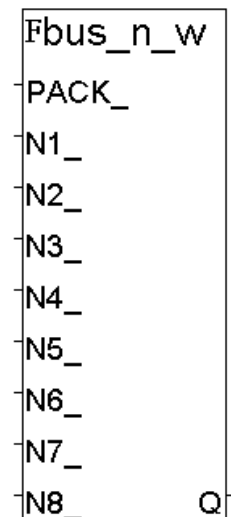
FBUS_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
┌─────────────┐
│  fbus_sts   │
│ID_          │
│PACK      Q  │
└─────────────┘
```

Description : C_Function Block
       Get package status of Fbus

Input Parameters :
    ID_   Integer   What to get?   0: Boolean package , 1: Integer package
    PACK_   Integer   Which package No. to get? (1-128)
Return :
    Q_   Boolean   TRUE: package is alive; FALSE: package is dead.
                            The reasons may cause package dead:
                            Fbus_m controller didn't enable the package number,
                            communication break, the controller for sanding that package is
                            dead, Fbus_m controller is dead, or other…
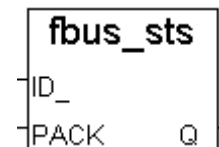
FM_READ

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
FM_READ
ID        Q
```
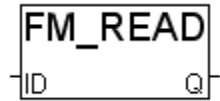
Description : Standard_Function

Read Message variables from a Binary file.

Input Parameters :

ID   Integer   File ID No. (returned by F_ROPEN, F_WOPEN or F_CREAT)

Return :

**Q**   Message   The returned message

Example :

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


(* IL Equivalence: *)
LD              '\Micro_SD\m1.txt'
F_ROPEN
ST              file_id
FM_READ                     (* read status1 *)
ST              status1
LD              file_id
F_CLOSE
ST              ok
```

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6) will consume much more CPU time. To do the file operation in RAM Disk, such as in \Temp\, will be better, but it will lost the file after turning off the controller.

FM_WRITE

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
FM_WRITE
─ID
─IN        ok ─
```
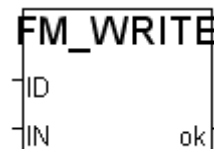
Description : Standard_Function
       Write Message into a Binary file.

Input Parameters :
    ID   Integer   File ID No. (returned by F_ROPEN, F_WOPEN or F_CREAT)
    IN   Message   The message to be write into the file
Return :
    OK   Boolean   Execution status. TRUE: ok.

Example :
(* ST Program: *)
file_id := F_WOPEN('\Micro_SD\m1.txt ');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);

(* IL Eauivalence: *)
LD 'trace.txt'
F_WOPEN
ST file_id
FM_WRITE 'First message'   (* write the first message *)
ST ok
LD file_id
FM_WRITE 'Last message'   (*write the second message *)
ST ok
LD file_id
F_CLOSE
ST ok

Note: To do the file operation in the \Micro_SD and \System_disk of WP-8xx7, WP-5xx7 and
     VP-2xW7 (or \System_disk2 and \System_disk of XP-8xx7-CE6 and XP-8xx7-Atom-CE6)
     will consume much more CPU time. To do the file operation in RAM Disk, such as in
     \Temp\, will be better, but it will lost the file after turning off the controller.

FR_B

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────────┐
│      fr_b   │
│             │
┤Slot_        │
│             │
┤Port_        │
│             │
┤Addr_        │
│             │
┤Ch_          │
│             │
┤B         Q  ├
└─────────────┘
```
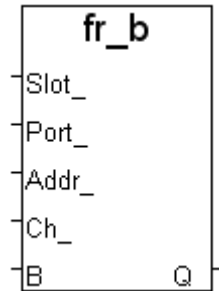
Description : C_Function

Set ISaGRAF Boolean variables as an FRNET I/O.
(RECOMMEND TO USE OTHER BETTER FUNCTION-BLOCKS –
"**FR_16di**" and "**FR_16do**" , **please refer to Chapter 23**)

Input Parameters :

Slot_	Integer	Slot No. that the related i-8172 is plugged in (1-7)
Port_	Integer	Port No. used in the i-8072 (0 or 1)
Addr_	Integer	Module address , D/O (0 - 7) , D/I (8 - 15)
Ch_	Integer	Channel number been used (1 - 16)
B_	Boolean	Boolean variable name

Return :

Q_	Boolean	True: Ok. False: parameter error.

Note :

1. Please connect i-8172 in Wincon's slot 1 to 7, and then connecting FRNET I/O modules from its Port0 or Port1.
2. Fr_B & Fr_B_A should be called in the first PLC scan. They don't work in the 2nd or other PLC scan.
3. FRNET D/O module doesn't support communication state detection. However FRNET D/I module support it.
4. Every FRNET Output module has a 'RESET' dip on its Dip switch or a special Jumper. User may set it to 'ON' position (or enable it), this will reset the output channels to OFF state when the communication is break between i-8172 and FRNET D/O module.
    For example, set 8th Dip to ON of FR-2057 means enable it.
5. You may visit below web site for more information:
    http://www.icpdas.com/faq/isagraf.htm   'FAQ048'   &
    http://www.icpdas.com/products/Remote_IO/frnet/frnet_list.htm

Example : Please refer to 'FAQ082' at http://www.icpdas.com/faq/isagraf.htm , and the Example file: Wdemo_70.

Fr_B_A

□ I-8xx7 ■ iPAC-8447/8847 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
    ┌─────────────┐
    │   fr_b_a    │
    ├─────────────┤
    ┤Slot_        │
    ┤Port_        │
    ┤Addr_        │
    ┤Ch_          │
    ┤NetW_        │
    ┤Num       Q ├
    └─────────────┘
```

Description : C_Function

Set ISaGRAF Boolean "variable array" as several FRNet I/O(s)
(Please refer to Section 2.6 for the information about "Variable array")
(RECOMMEND TO USE OTHER BETTER FUNCTION-BLOCKS –
"**FR_16di**" and "**FR_16do**" , **please refer to Chapter 23**)

Input Parameters :

Slot_	Integer	Slot No. that the related i-8172 is plugged in (1~7)
Port_	Integer	Port No. used in the i-8072 (0 or 1)
Addr_	Integer	Starting module address, D/O (0~7), D/I (8~15)
Ch_	Integer	Starting Channel number (1~16)
NetW_	Integer	Network address No. for the 1st element of the related "Variable Array": 1~8191
Num_	Integer	Amount of Boolean in the "Variable Array" to set as FRNet IOs. Valid range: 1~255. Ex: Bi[0..15] has size of 16, You may set NUM_ as 1 to 16. ABC[0..127] has size of 128, you may set NUM_ as 1 to 128.

Return :

Q_   Boolean   True: Ok.   False: parameter error.

Note :

1. Please connect i-8172 in Wincon's slot 1 to 7, and then connecting FRNET I/O modules from its Port0 or Port1.
2. Fr_B , Fr_B_A should be called in the first PLC scan. They don't work in the 2nd or other PLC scan.
3. FRNET D/O module doesn't support communication state detection. However FRNET D/I module support it.
4. Every FRNET Output module has a 'RESET' dip on its Dip switch or a special Jumper. User may set it to 'ON' position (or enable it), this will reset the output channels to OFF state when the communication is break between i-8172 and FRNET D/O module.
    For example, set 8th Dip to ON of FR-2057 means enable it.
5. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
    Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
             [DEBUG]
             arrays=1
6. You may visit below web site for more information:
    http://www.icpdas.com/faq/isagraf.htm   'FAQ082'   &
    http://www.icpdas.com/products/Remote_IO/frnet/frnet_list.htm

Example : Please refer to 'FAQ082' at http://www.icpdas.com/faq/isagraf.htm , and the Example file: Wdemo_70.

GET_INFO

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Get the controller information.

```
get_info
TYPE   DATA
```

Input Parameters :

    TYPE_ Integer   What information to get?

Return :

    DATA_ Integer   Returned information value.

Table : The following is the meaning of "TYPE_" V.S. "DATA_"

TYPE_		DATA_
1	Get the controller ID (slave No.)	1 - 255
2	reserved for the future	0
3	reserved for the future	0
4	reserved for the future	0

GET_SN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block

Get hardware unique serial No. Total is 8 Integer.

Return :

    SN_1_~SN_8_ Integer   The returned serial No.

                                    8 bytes (-128 to +127)

```
get_sn
       sn_1_
       sn_2_
       sn_3_
       sn_4_
       sn_5_
       sn_6_
       sn_7_
       sn_8_
```

GET_VER

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
get_ver
          OS_
       DRIVE
```
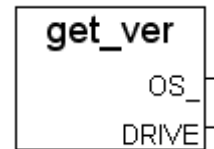
Description : C_Function Block

Get the version information of hardware driver
(Used for I-8xx7: v2.19, I-7188EG: v1.10, I-7188XG: v1.08 or higher version)

Return :

OS_	Message	The OS version should be updated (length: 48)
		Ex: "Must use 8n020704.img"
DRIVER_	message	The current Driver version (length: 48)
		Ex: "I-8xx7 : isa.exe - 2.19 , Dec.09,2002"

GETCTS

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
getcts
PORT      Q
```

Description : C_Function Block

Get CTS state of a COM port.

Input Parameters :

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

Return :

| Q_ | Boolean | TRUE: enable,   FALSE: disable. |

I_DICNT

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block

```
        i_dicnt
 ─ADR_
 ─ST_CN      Q_ ─
 ─RS1_      CN1_ ─
 ─RS2_      CN2_ ─
 ─RS3_      CN3_ ─
 ─RS4       CN4_ ─
```

Get the 4 DI Counters of extension DI modules link with COM3/4 of the I-8xx7 controller or COM2 of the I-7188XG/EG controller or COM3 of the W-8xx7 controller.

Input Parameters :

ADR_	Integer	The address of the I/O module(1~255), must be a constant value, not a variable.
ST_CN_	Integer	Starting Channel number (1~13), must be a constant value, not a variable. Ex: The I-87052 has 8 DI. If ST_CN_ set as 5, CN1_ get the Ch.5' counter value. CN2_ get Ch.6's, CN3_ get Ch.7's, CN4_ get Ch.8's value.
RS1_ ~ RS4_	Boolean	Reset the associated D/I counter when rising from False to True.

Return :

Q_	Boolean	TRUE: work ok,　FALSE: fail. "If Q_ is FALSE, it means communication is bad, the following return value has no meaning"
CN1_ ~ CN4_	Integer	The DI Counter value of channel No. starting from ST_CN_.

**The modules listed below have DI counters (Max to 100 Hz) :

"I_DICNT" support: i-87051 , 87052 , 87053 , 87054 , 87055 , 87058 , 87063
　　　　　　　　i-7041, 7044, 7050 , 7051 , 7052 , 7053 , 7055, 7058, 7060 , 7063 , 7065
"I_DICNT2" support: i-87040
Counter input channels: 4, valid value: 0 ~ 65535 (Max to 100 Hz)
Reset Counter channels: 4, Reset the D/I counter to 0 when rising from False to True.

Note :

1. I-DiCnt function block valid for version: I-8xx7:2.18 , I-7188EG:1.10 , I-7188XG:1.08 , W-8x37:3.20C or higher version.
2. For remote I-87040 module (32 D/I) please use "i_DiCnt2" function block.

VERY IMPORTANT:

If the I-87018/87019 is used as a RS-485 Remote I/O, please operate the following steps before connecting them.

A. At I/O module side, please use "DCON Utility" to setup:
　1. Set "address" to a unique No. (1~255)
　2. Set proper "checksum disabled" or "checksum enabled" and set Analog Input module to be "2's complement".
　3. Set communication parameter to "baud rate" and "8, N, 1"
　4. Set "Range Type" if using Analog board
B. At I-8xx7/I-7188EG/I-7188XG side:
　1. Connect I/O complex equipment "bus7000B" and set its "baud rate" and "Checksum" equal to the "baud rate" & "Checksum" setting of the I/O modules.
　2. I-8xx7 & 7188EG/XG can connect up to 64 remote I/O modules, W-8xx7 can up to 255.
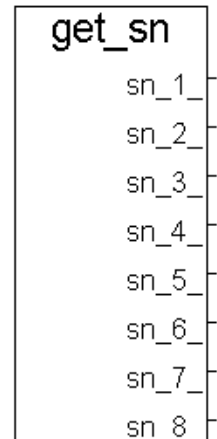　3. Please use i_7xxx function block or i_87xxx Function Block when programming RS-485 Remote I/O module.

I_DICNT2

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block

Get the 4 DI Counters of extension DI modules link with COM3/4 of the I-8xx7 controller or COM2 of the I-7188XG/EG controller or COM3 of the W-8xx7 controller.

```
        i_dicnt2
 ─ ADR_
 ─ ST_CN        Q_ ─
 ─ RS1_        CN1_ ─
 ─ RS2_        CN2_ ─
 ─ RS3_        CN3_ ─
 ─ RS4_        CN4 ─
```

Input Parameters:

ADR_ Integer The address of the I-8740 module(1~255), must be a constant value, not a variable.

ST_CN_ Integer Starting Channel number (1~29), must be a constant value, not a variable.
Ex: The I-87040 has 32 DI.
If ST_CN_ set as 15, CN1_ get the Ch.15' counter value.
CN2_ get Ch.16's, CN3_ get Ch.17's, CN4_ get Ch.18's value.

RS1_~RS4_ Boolean Reset the associated D/I counter when rising from False to True.

Return :

Q_ Boolean TRUE: work ok, FALSE: fail.
"If Q_ is FALSE, it means communication is bad, the following return value has no meaning"

CN1_~CN4_ Integer The DI Counter value of channel No. starting from ST_CN_.

**The modules listed below have DI counters (Max to 100 Hz) :

"I_DICNT" support: i-87051 , 87052 , 87053 , 87054 , 87055 , 87058 , 87063
 i-7041, 7044, 7050 , 7051 , 7052 , 7053 , 7055, 7058, 7060 , 7063 , 7065
"I_DICNT2" support: i-87040
Counter input channels: 4, valid value: 0 ~ 65535 (Max to 100 Hz)
Reset Counter channels: 4, Reset the D/I counter to 0 when rising from False to True.

Note :

1. I-DiCnt function block valid for version: I-8xx7:2.18 , I-7188EG:1.10 , I-7188XG:1.08 , W-8x37:3.20C or higher version.
2. For remote I-87040 module (32 D/I) please use "i_DiCnt2" function block.

VERY IMPORTANT:

If the I-87018/87019 is used as a RS-485 Remote I/O, please operate the following steps before connecting them.

A. At I/O module side, please use "DCON Utility" to setup:
1. Set "address" to a unique No. (1~255)
2. Set proper "checksum disabled" or "checksum enabled" and set Analog Input module to be "2's complement".
3. Set communication parameter to "baud rate" and "8, N, 1"
4. Set "Range Type" if using Analog board
B. At I-8xx7/I-7188EG/I-7188XG side:
1. Connect I/O complex equipment "bus7000B" and set its "baud rate" and "Checksum" equal to the "baud rate" & "Checksum" setting of the I/O modules.
2. I-8xx7 & 7188EG/XG can connect up to 64 remote I/O modules, W-8xx7 can up to 255.
3. Please use i_7xxx function block or i_87xxx Function Block when programming RS-485 Remote I/O module.

I_RESET

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
i_reset
           Q
```
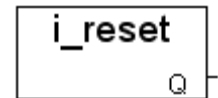
Description : C_Function
　　　Reset the controller

Return :
　Q_　　　Boolean　　　The return value has no meaning since the controller will reset

Note : Please use this function very careful.
1. If the controller is always reset, please refer to section 1.3.7 to delete the project inside the
　　I-8xx7 controller.
2. For I-8417/8817/8437/8837, I-7188EG, I-7188XG:
　　If the controller is reset always, please refer to Section 1.3.7 of "User's Manual Of The
　　I-8417/8817/8437/8837" for deleting the project inside the controller.
3. For W-8xx7/8xx6 (Wincon ISaGRAF version):
　　If the controller is reset always, unplug the CF card and restart WinCon power, plug on
　　CF card again, then delete the file - "\Micro_SD\ISaGRAF\ISA11". After that, restart
　　WinCon controller power again.

Example :
　(* OK1 is declared as Boolean input, TMP as Boolean internal *)
　if OK1=TRUE then
　　TMP := i_reset();
　end_if;

I7000_EN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
i7000_en
 EN 70    Q
```
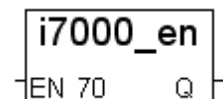
Description : C_Function
　　　Enable / Disable "Bus7000"

Input Parameters :
　EN_7000_　　　Boolean　　TRUE: enable,   FALSE: disable

Return :
　Q_　　　　　Boolean　　always return TRUE.

Note :
1. Default status is "Enable".
2. Only work when I/O complex equipment - "Bus7000" or "Bus7000B" is connected.

I8KE_B

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────────┐
│   i8ke_b    │
┤IP_          │
┤ADR_         │
┤IO_          │
┤B          Q ├
└─────────────┘
```

Description : C_Function
   Set ISaGRAF Boolean variable as an I-8KE4/8-MTCP Ethernet I/O
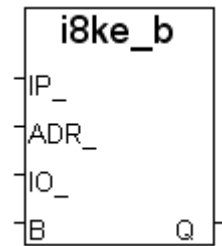
Input Parameters :
   IP_   Message   IP address of the related I-8KE4/8-MTCP, Ex: '192.168.100.123'
   ADR_   Integer   Modbus address of i-8KE4/8-MTCP DI/DO, 0~267.
   IO_   Boolean   True: input, False: output
   B_   Boolean   The Boolean variable name

Return :
   Q_   **:** Boolean   True: Ok.   False: parameter error.

Note :
1. For detail information, please refer to Chapter 22 or
      http://www.icpdas.com/faq/isagraf.htm   'FAQ042'   and
      http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i-8KE_B , i8KE_N , i8KE_F , i-8KE_B_A , i8KE_N_A , i-8KE_F_A should be called in the first
   PLC scan. They don't work in the 2nd & other PLC scan.

Demo Program: Wdemo_30 & Wdemo_31 at   http://www.icpdas.com/faq/isagraf.htm
   'FAQ042'

I8KE_B_A

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
i8ke_b_a
─IP_
─ADR_
─IO_
─NETW_
─NUM        Q─
```

Description : C_Function

Set ISaGRAF Boolean 'Variable Array' as several i-8KE4/8-MTCP
Ethernet I/O(s)
(Please refer to Section 2.6: Variable Array)

Input Parameters :

IP_	Message	IP address of the related i-8KE4/8-MTCP, Ex: '192.168.100.123'
ADR_	Integer	Modbus address of i-8KE4/8-MTCP DI/DO, 0~267
IO_	Boolean	True: input, False: output
NetW_	Integer	Network address No. for the 1st element of "Variable Array". 1~8191
Num_	Integer	Amount of Boolean in the "Variable Array" to set as Ethernet IO

Valid range: 1~255. (ADR_ + Num_) cannot be larger than 264.
Ex: Bi[0..15] has size of 16, NUM_ could be (1 ~ 16).
ABC[0..7] has size of 8, NUM_ could be (1 ~ 8).

Return :

Q_	Boolean	True: Ok. False: parameter error.

Note :

1. For detail information, please refer to Chapter 22 or
http://www.icpdas.com/faq/isagraf.htm   'FAQ042'   &
http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A should be called in the first
PLC scan. They don't work in the 2nd & other PLC scan.
3. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in
the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF
workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to
assign the size of the "variable array".
Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
        [DEBUG]
        arrays=1

Demo Program: Wdemo_30 & Wdemo_31

Please refer to   http://www.icpdas.com/faq/isagraf.htm   'FAQ042'

I8KE_F

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────────┐
│   i8ke_f    │
├─────────────┤
┤IP_          │
┤ADR_         │
┤IO_          │
┤F_           │
┤X0_          │
┤Y0_          │
┤XF1_         │
┤YF1_       Q ├
└─────────────┘
```

Description : C_Function

Set ISaGRAF REAL variable as an I-8KE4/8-MTCP Ethernet I/O,
and scaling to Real type

Input Parameters :

IP_   Message   IP address of the related I-8KE4/8-MTCP,
                     Ex: '192.168.100.123'
ADR_   Integer   Modbus address of I-8KE4/8-MTCP AI/AO: 0~127
IO_   Boolean   True: input , False: output
F_   REAL   REAL variable name

----- For scaling, if no scaling, please set below parameters as (0 , 0 , 0.0 , 0.0) -----

X0_   Integer   original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                     Valid range: -32768 <= X0_ <= +32767
Y0_   Integer   Original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                     Valid range: -32768 <= Y0_ <= +32767
XF1_   REAL   Engineering value after scaling, XF1_ **DO NOT equal to** YF1_ .
YF1_   REAL   Engineering value after scaling, XF1_ **DO NOT equal to** YF1_ .

Return :

Q_   Boolean True: Ok.   False: parameter error.

Example :

Ex 1: I-8017h set range_type as '+/- 10 V' (i-8017h's input value is -32768 to +32767). User
   may want to scale (0 , 10 V) to become engineering value of (0 , 1000 Psi). Please set
   (X0_ , Y0_) = (0 , +32767) , (XF1_ , YF1_) = (0.0 , 1000.0)

Ex 2: I-8024 set range_type as '0 to 20 mA' (i-8024's output value is 0 to +32767). User may
   want to scale (4 , 20 mA) to become engineering value of (0 , 3000 rpm). Please set
   (X0_ , Y0_) = (6553 , +32767) , (XF1_ , YF1_) = (0.0 , 3000.0)

Note :

1. For detail information, please refer to Chapter 22 or
     http://www.icpdas.com/faq/isagraf.htm   'FAQ042'   &
     http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A should be called in the first
   PLC scan. They don't work in the 2nd & other PLC scan.

Demo Program: Wdemo_30 & Wdemo_31

Please refer to http://www.icpdas.com/faq/isagraf.htm   'FAQ042'

I8KE_F_A

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
i8ke_f_a
IP_
ADR_
IO_
NETW_
NUM_
X0_
Y0_
XF1_
YF1_      Q
```

Description : C_Function

Set ISaGRAF REAL 'Variable Array' as several i-8KE4/8-MTCP Ethernet I/O(s), and scaling to Real type (Please refer to Section: 2.6: 'Variable Array')

Input Parameters :

IP_ Message IP address of the related I-8KE4/8-MTCP,
Ex: '192.168.100.123'

ADR_ Integer Modbus address of I-8KE4/8-MTCP AI/AO: 0~127

IO_ Boolean True: input , False: output

NetW_ Integer Network address No. for the 1st element of "Variable Array".
1~8191

Num_ Integer Amount of Real in the "Variable Array" to set as Ethernet IO
Valid range: 1~255. (ADR_ + Num_) cannot be larger than 128.
Ex: R1[0..31] has size of 32, NUM_ could be (1 ~ 32).
R3[0..7] has size of 8, NUM_ could be (1 ~ 8).
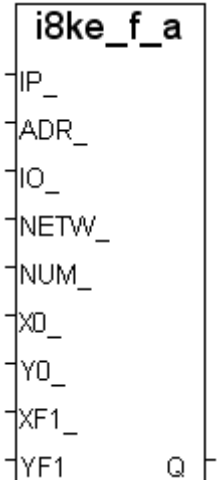
----- **For scaling, if no scaling, please set below parameters as (0 , 0 , 0.0 , 0.0)** -----

X0_ Integer original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
Valid range: -32768 <= X0_ <= +32767

Y0_ Integer original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
Valid range: -32768 <= Y0_ <= +32767

XF1_ REAL Engineering value after scaling, XF1_ **DO NOT equal to** YF1_ .

YF1_ REAL Engineering value after scaling, XF1_ **DO NOT equal to** YF1_ .

Return :

Q_ Boolean True: Ok. False: parameter error.

Example :

Ex : I-87024 set range_type as '4 to 20 mA' (i-87024's output value is 0 to +32767). User may want to scale (4 , 20 mA) to become engineering value of (0 , 5000 rpm). Please set (X0_ , Y0_) = (0 , +32767) , (XF1_ , YF1_) = (0.0 , 5000.0)

Note :

1. For detail information, please refer to Chapter 22 or
http://www.icpdas.com/faq/isagraf.htm 'FAQ042' &
http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm

2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A should be called in the first PLC scan. They don't work in the 2nd & other PLC scan.
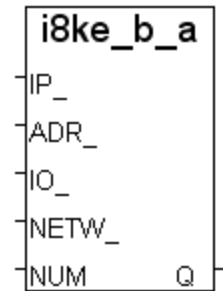
3. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
[DEBUG]
arrays=1
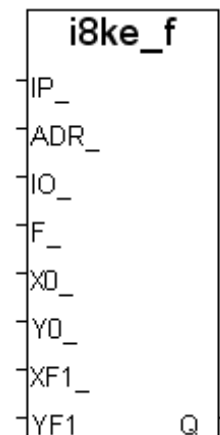
Demo Program: Wdemo_30 & Wdemo_31.
Please refer to http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_N

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
i8ke_n
IP_
ADR_
IO_
N_
X0_
Y0_
X1_
Y1_        Q
```

Description : C_Function

Set ISaGRAF Integer 'Variable Array' as an I-8KE4/8-MTCP Ethernet I/O(s), and scaling to Integer type.

Input Parameters :

IP_   Message IP address of the related I-8KE4/8-MTCP,
                     Ex: '192.168.100.123'
ADR_   Integer   Modbus address of I-8KE4/8-MTCP AI/AO: 0~127
IO_   Boolean   True: input , False: output
F_   REAL   REAL variable name
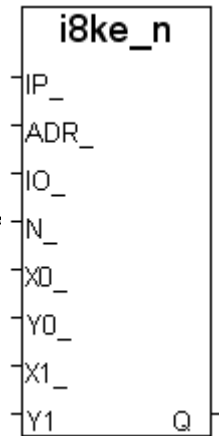
----- For scaling, if no scaling, please set below parameters as (0 , 0 , 0.0 , 0.0) -----

X0_   Integer   original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                     Valid range: -32768 <= X0_ <= +32767
Y0_   Integer   original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                     Valid range: -32768 <= Y0_ <= +32767
X1_   Integer   Engineering value after scaling. X1_ **DO NOT equal to** Y1_ .
                     Valid range: -30000 <= X1_ <= +30000
Y1_   Integer   Engineering value after scaling. X1_ **DO NOT equal to** Y1_ .
                     Valid range: -30000 <= Y1_ <= +30000

Return :

Q_   Boolean True: Ok.   False: parameter error.

Example :

Ex 1: i-8017h set range_type as '+/- 10 V' (i-8017h's input value is -32768 to +32767). User may want to scale (0 , 10 V) to become engineering value of (0 , 1000 Psi). Please set (X0_ , Y0_) = (0 , +32767) , (X1_ , Y1_) = (0 , 1000)

Ex 2: i-8024 set range_type as '0 to 20 mA' (i-8024's output value is 0 to +32767). User may want to scale (4 , 20 mA) to become engineering value of (0 , 3000 rpm). Please set (X0_ , Y0_) = (6553 , +32767) , (X1_ , Y1_) = (0 , 3000)

Note :

1. For detail information, please refer to Chapter 22 or:
   http://www.icpdas.com/faq/isagraf.htm   'FAQ042'   &
   http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A should be called in the first PLC scan. They don't work in the 2nd & other PLC scan.

Demo Program: Wdemo_30 & Wdemo_31.

Please refer to http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

I8KE_N_A

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
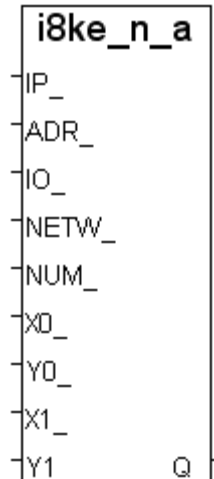
```
i8ke_n_a
─┤IP_
─┤ADR_
─┤IO_
─┤NETW_
─┤NUM_
─┤X0_
─┤Y0_
─┤X1_
─┤Y1_      Q├─
```

Description : C_Function

   Set ISaGRAF Integer 'Variable Array' as several i-8KE4/8-MTCP
   Ethernet I/O(s), and scaling to Integer type.
   (Please refer to Section 2.6: 'Variable Array')

Input Parameters :

   IP_   Message   IP address of the related I-8KE4/8-MTCP,
                        Ex: '192.168.100.123'
   ADR_   Integer   Modbus address of I-8KE4/8-MTCP AI/AO: 0~127
   IO_   Boolean   True: input , False: output
   NetW_ Integer   Network address No. for the 1st element of "Variable Array". 1~8191
   Num_   Integer   Amount of Integer in the "Variable Array" to set as Ethernet IO
                        Valid range: 1~255. (ADR_ + Num_) cannot be larger than 128.
                        Ex: ENG1[0..63] has size of 64, NUM_ could be (1 ~ 64).
                            Ai[0..7] has size of 8, NUM_ could be (1 ~ 8).

   ----- **For scaling, if no scaling, please set below parameters as (0 , 0 , 0.0 , 0.0)** -----

   X0_   Integer   original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                        Valid range: -32768 <= X0_ <= +32767
   Y0_   Integer   original value of the analog Input/Output board. X0_ **DO NOT equal to** Y0_.
                        Valid range: -32768 <= Y0_ <= +32767
   X1_   REAL   Engineering value after scaling. X1_ **DO NOT equal to** Y1_ .
                        Valid range: -30000 <= X1_ <= +30000
   Y1_   REAL   Engineering value after scaling. X1_ **DO NOT equal to** Y1_ .
                        Valid range: -30000 <= Y1_ <= +30000

Return :

   Q_   Boolean True: Ok. False: parameter error.

Example :

   I-87018R set range_type as 'Thermo-Couple K-type: -270 to +1372 degree celsius'
   (i-87018R's input value is -6448 to +32767). User may want to scale (-270 , +1372 degree)
   to become engineering value of (-2700 , +13720). Please set (X0_ , Y0_) = (-6448 ,
   +32767) , (X1_ , Y1_) = (-2700 , +13720)

Note :

1. For detail information, please refer to Chapter 22   or
      http://www.icpdas.com/faq/isagraf.htm   'FAQ042'   &
      http://www.icpdas.com/products/PAC/i-8000/i-8KE4_8KE8_MTCP.htm
2. i8KE_B , i8KE_N , i8KE_F , i8KE_B_A , i8KE_N_A , i8KE_F_A should be called in the first
   PLC scan. They don't work in the 2nd & other PLC scan.
3. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in
   the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF
   workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to
   assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
          [DEBUG]
          arrays=1

Demo Program: Wdemo_30 & Wdemo_31.

   Please refer to http://www.icpdas.com/faq/isagraf.htm 'FAQ042'

INP10LED

inp10led

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
□ ISaGRAF WinCE PAC

RUN_
VA_I_
NUM_
U1_
D1_
L1_
R1_   VA_O

Description : C_Function
Input a decimal Integer from the S_MMI

Input Parameters :
RUN_	Boolean	When "TRUE", process & display value to SMMI
VAL_I_	Integer	Initial value displayed on S-MMI (0 ~ 99999)
NUM_	Integer	Number of digits to display (1 ~ 5)
U1_	Boolean	When it rising from "false" to "true", add 1 to current displayed digit.
D1_	Boolean	When it rising from "false" to "true", subtract 1 from current displayed digit.
L1_	Boolean	When it rising from "false" to "true", shift left 1 position from current displayed digit
R1_	Boolean	When it rising from "false" to "true", shift right 1 position from current displayed digit

Return :
VAL_O_ Integer   The displayed Integer value after operation

Example: Please refer to demo_08 & demo_11a.

inp10led

TRUE	RUN_
100	VAL_I_
4	NUM_
UU	U1_
DD	D1_
LL	L1_
FALSE	R1_ VAL_O_ A
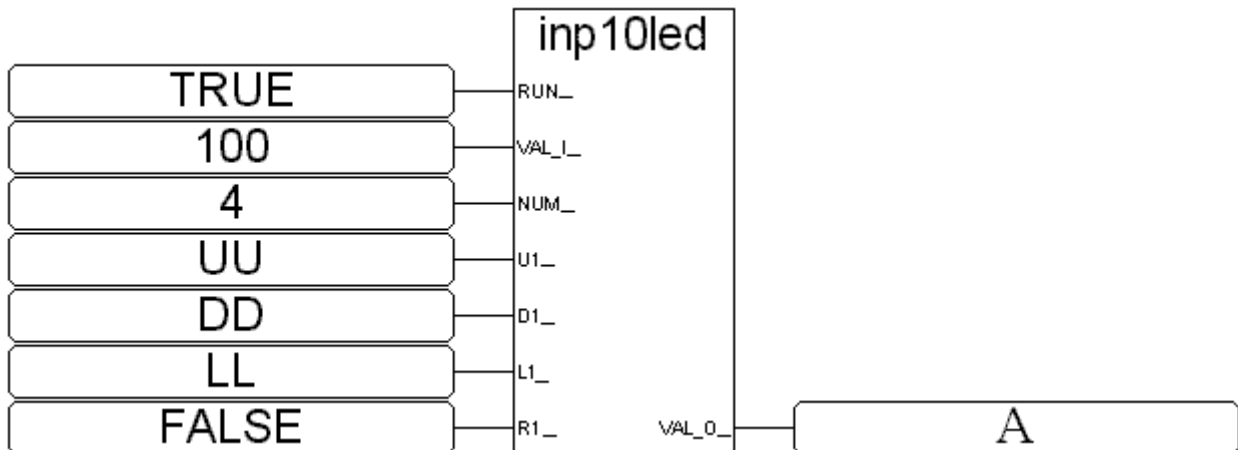
ST Equivalence:

A := INP10LED(TRUE,100,4,UU,DD,LL,FALSE);
(* A is declared as an integer variable *)
(* UU,DD,LL are declared as input boolean variables, can be linked to "push4key"
  board *)

INP16LED

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
□ ISaGRAF WinCE PAC

```
 inp16led
RUN_
VA_I_
NUM_
U1_
D1_
L1_
R1      VA_O
```

Description : C_Function
Input a hexadecimal Integer from the S_MMI

Input Parameters :
RUN_	Boolean	When "TRUE", process & display value to SMMI
VAL_I_	Integer	Initial value displayed on S-MMI (0 ~ 16#FFFFF)
NUM_	Integer	Number of digits to display (1 ~ 5)
U1_	Boolean	When it rising from "false" to "true", add 1 to current displayed digit.
D1_	Boolean	When it rising from "false" to "true", subtract 1 from current displayed digit.
L1_	Boolean	When it rising from "false" to "true", shift left 1 position from current displayed digit
R1_	Boolean	When it rising from "false" to "true", shift right 1 position from current displayed digit

Return :
VAL_O_	Integer	The displayed Integer value after operation

Example:

```
                        inp16led
  TRUE          RUN
  16#2F04       VAL_I
  5             NUM
  UU            U1
  FALSE         D1
  LL            L1
  FALSE         R1      VAL_O        A
```

ST equivalence:
    A := INP16LED(TRUE,16#2F04,4,UU,FALSE,LL,FALSE);
    (* A is declared as an integer variable *)
    (* UU,LL are declared as boolean variables,can be linked to "push4key" board *)

INT_REAL

Int_Real
Long   Real

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
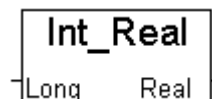
Description : C_Function
      Map a Long Integer to a Real value.
      The algorithm in C language is   Real_ = *((float *)&Long_);

Input Parameters :
    Long_   Integer   The 32-bit Integer to map

Return :
    Real_   Real   The Real value after mapping

Note:
1. "Real_Int" can be used to map a Real value to a Long Integer.
2. If you just want to convert one Integer value to Real value, Ex: 32 → 32.0 , please use
    "REAL( )" function.

VERY IMPORTANT:

The "**int_real**(L1)" may cause controller un-stable if the parameter "L1" is not generated by a
    previous "L1 := **real_int**(R1)" command (Refer to **section 10.6**).

INT_STR3

int_str3
INT_
WIDTH   STR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
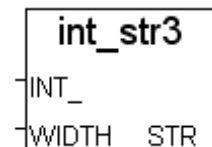
Description : C_Function
      Convert an Integer value to a String with fixed digit number.

Input Parameters :
    INT_   Integer   The Integer to be converted.
    WIDTH_   Integer   Max number of digit to show, 1 ~ 13

Return :
    STR_   Message   The returned String (max. length: 13).
                       Ex. if WIDTH_ = 4
                      12 ---> ' 12'
                      123456 ---> '****'

LONG_WD

long_wd
     Lo_
Long   Hi

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC
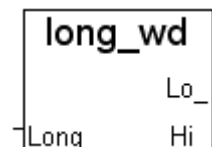
Description : C_Function Block
      Convert one Long Integer (signed 32-bit) to two Words (signed 16-bit)

Input Parameters :
    LONG_   Integer   The 32-bit Integer to be converted.

Return :
    LO_   Integer   The low Word value after the conversion, (-32768 ~ +32767)
    HI_   Integer   The high Word value after the conversion, (-32768 ~ +32767)

MAIL_SET

□ I-8xx7 ■ iPAC-8447/8847 □ I-7188EG/XG ■ uPAC-7186EG ■ ISaGRAF WinCE PAC

Description : C_Function
Set email receiver , mail server and sender.
Please refer to Chapter 12 and
www.icpdas.com – FAQ – Software – ISaGRAF – 067 , 071 , 072 , 076 ~ 080

MAIL_SND

□ I-8xx7 ■ iPAC-8447/8847 □ I-7188EG/XG ■ uPAC-7186EG ■ ISaGRAF WinCE PAC

Description : C_Function
Send an Email via ethernet port.
Please refer to Chapter 12 and
www.icpdas.com – FAQ – Software – ISaGRAF – 067 , 071 , 072 , 076 ~ 080

MBUS_B_R
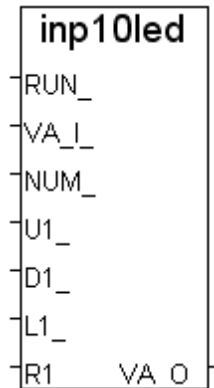
■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
                        mbus_b_r
                               Q_
                              B1_
                              B2_
                              B3_
                              B4_
                              B5_
                              B6_
                    SLAVE_    B7_
                    ADDR_     B8_
```

Description : C_Function Block
Read 8 bits (Booleans) from the Modbus device

Use the Modbus function code ---- 1

Input Parameters :

SLAVE_	Integer	Slave No (Net-ID) of the Modbus device.
		It must be a constant, not a variable
		Modbus Device ID = remainder of SLAVE_ / 1000
		COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
		I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
		I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
		I-7188XG : (2:COM2, 3:COM3)
		W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
		Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.
ADDR_	Integer	The starting Modbus address to read (0~65535)
		It must be a constant, not a variable

Return :

Q_	Boolean	Ok. return TRUE, else return FALSE
B1_ ~ B8_	Boolean	The 8 Boolean values that have been read

Note: The total number of (MBUS_B_R + MBUS_BR1) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller.
For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_B_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
mbus_b_w
SLAVE
ADDR_
NUM_W
ACTIO
B1_
B2_
B3_
B4_      Q
```

Description : C_Function Block

Write 1 to 4 bits (Booleans) to the Modbus device

Use Modbus function code 5 when NUM_W = 1
Use Modbus function code 15 when NUM_W = 2 to 4

Input Parameters :

SLAVE_ Integer Slave No (Net-ID) of the Modbus device.
It must be a constant, not a variable
Modbus Device ID = remainder of SLAVE_ / 1000
COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
  I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
  I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
  I-7188XG : (2:COM2, 3:COM3)
  W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.

ADDR_ Integer The starting Modbus address to write (0~65535).
It must be a constant, not a variable

NUM_W_ Integer The number of Booleans to write (1 to 4).
It must be a constant, not a variable

ACTION_ Boolean Set true to write, set FALSE to do nothing

B1_ ~ B4_ Boolean Boolean to write

Return :

Q_ Boolean Ok, return TRUE, else return FALSE

Note: The total number of (MBUS_B_W) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller.
For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_BR1

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
        mbus_br1
                  Q
                  B1_
                  B2_
                  B3_
                  B4_
                  B5_
        SLAVE     B6_
        ADDR_     B7_
        PERIO     B8_
```

Description : C_Function Block

Read 8 bits (Booleans) from the Modbus device with a period time

Using Modbus function code 1

Input Parameters :

SLAVE_	Integer	Slave No (Net-ID) of the Modbus device.
		It must be a constant, not a variable
		Modbus Device ID = remainder of SLAVE_ / 1000
		COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
		I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
		I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
		I-7188XG : (2:COM2, 3:COM3)
		W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
		Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.
ADDR_	Integer	The starting Modbus address to read (0~65535).
		It must be a constant, not a variable
PERIOD_	Integer	The period to read (1~600), unit: second

Return :

Q_	Boolean	Ok, return TRUE, else return FALSE
B1_~B8_	Boolean	The 8 Boolean values that have been read

Note: The total number of (MBUS_B_R + MBUS_BR1) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller.
For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

MBUS_N_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
mbus_n_r
              Q
         N1_
         N2_
         N3_
         N4_
         N5_
         N6_
SLAVE    N7_
ADDR_    N8_
```

Description : C_Function Block
        Read 8 Words (signed 16-bit) from the Modbus device

Using Modbus function code 3

Input Parameters :
SLAVE_ Integer   Slave No (Net-ID) of the Modbus device.
                    It must be a constant, not a variable
                            Modbus Device ID = remainder of SLAVE_ /
        1000
                            COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
                              I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
                              I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
                              I-7188XG : (2:COM2, 3:COM3)
                              W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
                            Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.
ADDR_   Integer   The starting Modbus address to read (0~65535).
                        It must be a constant, not a variable

Return :
Q_   Boolean   Ok, return TRUE, else return FALSE
N1_ ~ N8_ Integer   The 8 words values that have been read (-32768 ~ 32767)

Note: The total number of (MBUS_N_R) that can be used in one ISaGRAF project is up to
            64 for I-8xx7or I-7188EG/XG controller.
            For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_N_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block
      Write 1 to 4 Word (signed 16-bit) to the Modbus device

Use Modbus function code 6 when NUM_W = 1
Use Modbus function code 16 when NUM_W = 2 to 4

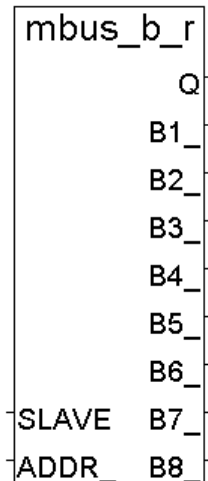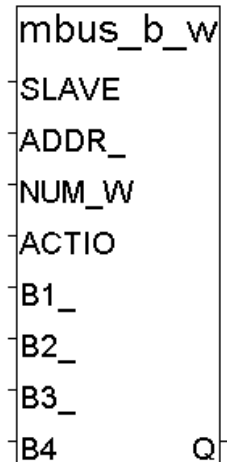**Input Parameters :**

SLAVE_ Integer		Slave No (Net-ID) of the Modbus device.
		It must be a constant, not a variable
		Modbus Device ID = remainder of SLAVE_ / 1000
		COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
		I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
		I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
		I-7188XG : (2:COM2, 3:COM3)
		W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
		Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.
ADDR_	Integer	The starting Modbus address to write (0~65535).
		It must be a constant, not a variable
NUM_W_	Integer	The number of Words to write (1 to 4).
		It must be a constant, not a variable
ACTION_	Boolean	Set true to write, set FALSE to do nothing
N1_ ~ N4_	Integer	words to write(-32768 ~ 32767)

Return :

Q_	Boolean	Ok, return TRUE, else return FALS

Note: The total number of (MBUS_N_W) that can be used in one ISaGRAF project is up to
       64 for I-8xx7or I-7188EG/XG controller.
       For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

Note: There is one new "Mbus12w" function block for writing 1 to 12 words. The driver
       should be the following to support it.

I-7188EG:	(V3.15 or later)
I-7188XG:	(V3.14 or later)
iP-8xx7:	(V1.10 or later)
VP-2117:	(V1.02 or later)
WP-8xx7/8xx6:	(V1.37 or later)
VP-2xW7/2 xW6:	(V1.29 or later)
XP-8xx7-CE6/ XP-8xx6-CE6:	(V1.17 or later)
XP-8xx7-Atom-CE6/ XP-8xx6-Atom -CE6:	(V1.01 or later)
WP-5147/5147-OD:	(V1.01 or later)

MBUS_NR1

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
mbus_nr1
            Q_
            N1_
            N2_
            N3_
            N4_
            N5_
  SLAVE     N6_
  ADDR_     N7_
  PERIO     N8_
```
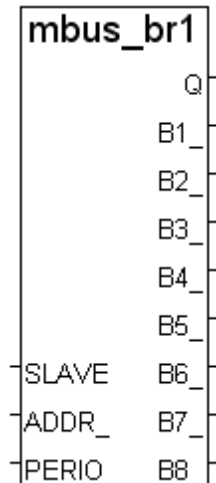
Description : C_Function Block
Read 8 Words (signed 16-bit) from the Modbus device with a period time

Using Modbus function code 3

Input Parameters :

SLAVE_ Integer Slave No (Net-ID) of the Modbus device.
It must be a constant, not a variable
Modbus Device ID = remainder of SLAVE_ / 1000
COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
  I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
  I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
  I-7188XG : (2:COM2, 3:COM3)
  W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.

ADDR_ Integer The starting Modbus address to read (0~65535).
It must be a constant, not a variable

PERIOD_ Integer The period to read (1~600), unit: second

Return :

Q_ Boolean Ok, return TRUE, else return FALSE
B1_ ~ B8_ Boolean The 8 words values that have been read (-32768 ~ 32767)

Note: The total number of (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller. For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_R

```
                                        ┌─────────┐
                                        │ mbus_r  │
                                        ┤         ├ Q
                                        │         ├ N1_
                                        │         ├ N2_
                                        │         ├ N3_
                                        │         ├ N4_
                                        │         ├ N5_
                                        │         ├ N6_
                                        │         ├ N7_
                                        │         ├ N8_
                                     SLAVE       ├ N9_
                                     ADDR_       ├ N10_
                                     CODE_       ├ N11_
                                     NUM         ├ N12_
                                        └─────────┘
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function Block
Read data from the Modbus device

* Using Modbus function code 1 or 2 or 3 or 4
* Please refer to FAQ-096 and FAQ-101 to use Mbus24R, Mbus24R1,
   Mbus_XR, Mbus_XR1 in the ISaGRAF WinCE PAC.

Input Parameters :

SLAVE_	Integer	Slave No (Net-ID) of the Modbus device.
		It must be a constant, not a variable
		Modbus Device ID = remainder of SLAVE_ / 1000
		COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
		I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
		I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
		I-7188XG : (2:COM2, 3:COM3)
		W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
		Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.
ADDR_	Integer	The starting Modbus address to read (0~65535).
		It must be a constant, not a variable
CODE_	Integer	The Modbus function code to use, 1 – 4
		It must be a constant, not a variable
NUM_	Integer	If CODE_=1 or 2, it's to request how many bits? 1~192
		If CODE_=3 or 4, it's to request how many words? 1~12
		It must be a constant, not a variable

Return :

Q_	Boolean	Ok, return TRUE, else return FALSE
N1_~N12_	Integer	The returned Bit or Word
		If CODE_=1 or 2, N1_ return Bit 1 to 16, N2_ return Bit 17 to 32, ... N12_ return Bit 177 to 192.
		If **CODE_=**3 or 4, N1_ to N12_ return Word 1 to 12 (value between -32768 to 32767).

Note: The total number of (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller. For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_R1

```
                                              ┌─────────┐
                                              │ mbus_r1 │
                                              │        Q├
                                              │       N1_├
■ I-8x17/iP-8x17  ■ I-8x37/iP-8x47  ■ I-7188EG/7186EG  ■ I-7188XG
                                              │       N2_├
■ ISaGRAF WinCE PAC                           │       N3_├
                                              │       N4_├
**Description : C_Function Block**            │       N5_├
      Read data from the Modbus device with a period time
                                              │       N6_├
                                              │       N7_├
*  Using Modbus function code 1 or 2 or 3 or 4
*  Please refer to FAQ-096 and FAQ-101 to use Mbus24R, Mbus24R1,
```
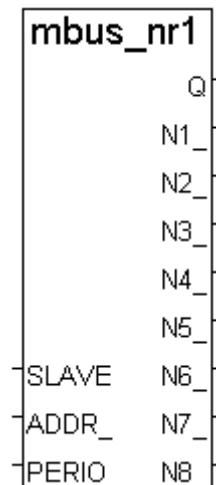
```
                                        ┤SLAVE   N8_├
                                        ┤ADDR_   N9_├
                                        ┤CODE_  N10_├
                                        ┤NUM_   N11_├
                                        ┤PERIO  N12_├
                                        └─────────┘
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

Description : C_Function Block
      Read data from the Modbus device with a period time

* Using Modbus function code 1 or 2 or 3 or 4
* Please refer to FAQ-096 and FAQ-101 to use Mbus24R, Mbus24R1,
   Mbus_XR, Mbus_XR1 in the ISaGRAF WinCE PAC.

Input Parameters :

SLAVE_   Integer   Slave No (Net-ID) of the Modbus device.
It must be a constant, not a variable
Modbus Device ID = remainder of SLAVE_ / 1000
COM port used by ISaGRAF Controller= quotient of SLAVE_ / 1000
  I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
  I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
  I-7188XG : (2:COM2, 3:COM3)
  W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 … 14:COM14)
Ex: SLAVE_ = 3001, then use COM3, and the Device ID is 1.

ADDR_   Integer   The starting Modbus address to read (0~65535).
It must be a constant, not a variable

CODE_   Integer   The Modbus function code to use, 1~4
It must be a constant, not a variable

NUM_   Integer   If CODE_=1 or 2, it's to request how many bits. 1~192
If CODE_=3 or 4, it's to request how many words. 1~12
It must be a constant, not a variable

PERIOD_   Integer   The period to read (1~600), unit: second.

Return :

Q_   Boolean   Ok, return TRUE, else return FALSE

N1_ ~ N12_   Integer   the returned bit or word
If CODE_= 1 or 2, N1_ return Bit 1 to 16, N2_ return Bit 17 to 32, ... N12_ return Bit 177 to 192.
If **CODE_=** 3 or 4, N1_ to N12_ return Word 1 to 12
(value between -32768 to 32767).

Note: The total number of (MBUS_N_R + MBUS_NR1 + MBUS_R + MBUS_R1) that can be used in one ISaGRAF project is up to 64 for I-8xx7or I-7188EG/XG controller. For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

Example: Please refer to Chapter 8.

MBUS_WB

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
mbus_wb
SLAVE
ADDR_
NUM_
ACTIO
B1_
B2_
B3_
B4_
B5_
B6_
B7_
B8_
B9_
B10_
B11_
B12_
B13_
B14_
B15_
B16_        Q
```

Description : C_Function Block

Write 1~16 Boolean values to Modbus device

Using Modbus function code 15
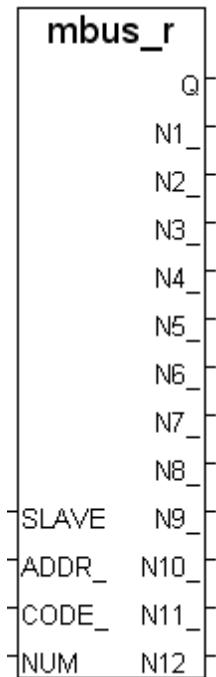
Input Parameters :

SLAVE_ Integer   Slave No (Net-ID) of the Modbus device.
It must be a constant, not a variable
Modbus Device ID = remainder of SLAVE_ / 1000
COM port used by ISaGRAF Controller= quotient of
SLAVE_ / 1000
I-8xx7 : (1:COM1, 3:COM3, 4:COM4, 5:COM5)
I-7188EG/7186EG : (1:COM1, 2:COM2, 3:COM3)
I-7188XG : (2:COM2, 3:COM3)
W-8xx7/8xx6 : (2:COM2, 3:COM3, 4:COM5 …
14:COM14)
Ex: SLAVE_ = 3001, then use COM3, and the
Device ID is 1.

ADDR_   Integer   The starting Modbus address to write (0~65535).
It must be a constant, not a variable

NUM_W_   Integer   The number of Booleans to write (1 ~ 16).
It must be a constant, not a variable

ACTION_   Boolean   Set true to write, set FALSE to do nothing

B1_~B16_   Boolean   Booleans to write (-32768 ~ 32767)
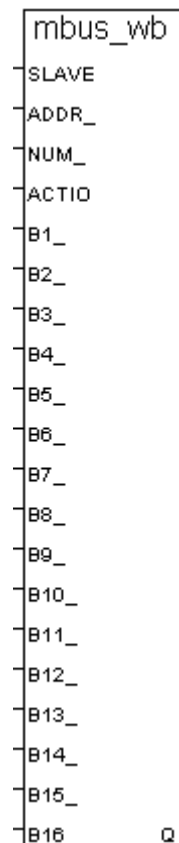
Return :

Q_   Boolean   Ok, return TRUE, else return FALSE

Note: The total number of (MBUS_B_W + MBUS_WB) that can be used in one ISaGRAF
project is up to 64 for I-8xx7or I-7188EG/XG controller.
For ISaGRAF WinCE PAC, the total number is up to 256 for each COM Port.

MI_BOO

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
Display a Boolean value on MMICON

```
   ┌─────────────┐
   │   mi_boo    │
  ─┤X_           │
  ─┤Y_           │
  ─┤BOO        Q ├─
   └─────────────┘
```

Input Parameters :
X_	Integer	X position, 1-30
Y_	Integer	Y position, 1-8
BOO_	Boolean	Boolean value to display.
		TRUE display "ON", FALSE display "OFF"

Return :
Q_	Boolean	Ok, return TRUE, else return FALSE

Example: Please refer to Chapter 16 & demo_38, demo_39

MI_INP_N

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
Input an Integer value from MMICON

```
   ┌─────────────┐
   │  mi_inp_n   │
  ─┤EN_          │
  ─┤INIT     INT ├─
   └─────────────┘
```
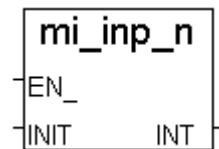
Input Parameters :
EN_	Boolean	TRUE: enable
INIT_	Integer	Initial value to input

Return :
INT_	Integer	The Integer value been inputted. If EN_ is FALSE, it returns 0.

Note :
MI_INP_N & MI_INP_S can be used only at one place in the project. Called at 2 or more places
will work failed.

Example: Please refer to Chapter 16 & demo_38, demo_39

MI_INP_S

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
   mi_inp_s
─EN_
─INIT    STR ─
```
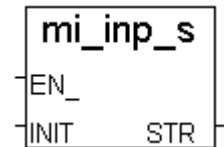
Description : C_Function
Input a String from MMICON

Input Parameters :
EN_   Boolean   TRUE: enable
INIT_   Message   Initial string value to input

Return :
STR_   Message   The String been inputted. If EN_ is FALSE, it returns '' (empty String)

Note:
MI_INP_N & MI_INP_S can be used only at one place in the project. Called at 2 or more places will work failed.

Example: Please refer to Chapter 16 & demo_38, demo_39

MI_INT

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
    mi_int
─X_
─Y_
─LEN_
─INT      Q ─
```
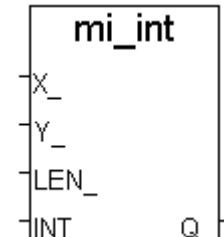
Description : C_Function
Display an Integer value on MMICON

Input Parameters :
X_   Integer   X position, 1-30
Y_   Integer   Y position, 1-8
LEN_   Integer   Max number of digits to display (1~11).
INT_   Integer   Integer value to display. If the length is over LEN_, it display '******'.

Return :
Q_   Boolean   Ok, return TRUE, else return FALSE

Example: Please refer to Chapter 16 & demo_38, demo_39

MI_REAL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
    mi_real
―X_
―Y_
―LEN_
―LEN1_
―REAL    Q―
```

Description : C_Function
        Display a Real value on MMICON
Input Parameters :
   X_   Integer   X position, 1~30
   Y_   Integer   Y position, 1~8
   LEN_   Integer   Max number of digits to display, 1~13
   LEN1_   Integer   Number of digit after '.' (0~4) and less than LEN_.
                              Ex. if LEN_=7, LEN1_=2, "123.4567" will be displayed as " 123.45"
   REAL_   Real   Real value to display, if the length is over LEN_, it display '******'.
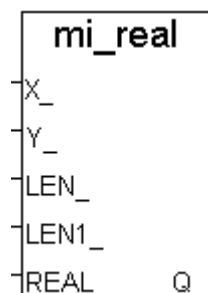
Return :
   Q_   Boolean   Ok, return TRUE, else return FALSE

Note:
If abs. of the real value >= 1,000,000 or (> 0 & < 0.0001) , please give LEN_ as 13 to display.
Ex. -123,456,789, please set LEN_ to 13 and it is displayed as -1.23457e+008.
      And 0.0000123456, please set LEN_ to 13 and it is displayed as 1.23456e-005

Example: Please refer to Chapter 16 & demo_38, demo_39

MI_STR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
    mi_str
―X_
―Y_
―LEN_
―STR_
―PASSW    Q―
```

Description : C_Function
        Display a String on MMICON

Input Parameters :
   X_   Integer   X position, 1~30
   Y_   Integer   Y position, 1~8
   LEN_   Integer   Max number of characters to display, 1~240
   STR_   Message   The String to display. If the number of characters exceeds LEN_,
                              only the first LEN_ of char. will be displayed
   PASSWD_   Boolean   TRUE: display as password, all char. are replaced as '*'.
                              FALSE: displayed as String.
Return :
   Q_   Boolean   Ok, return TRUE, else return FALSE
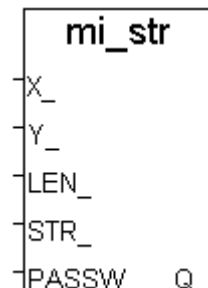
Example: Please refer to Chapter 16 & demo_38, demo_39

MSG_F

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
      msg_f
MSG_
ARY_N  NUM
```
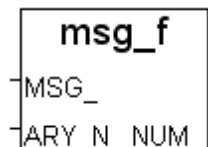
Description : C_Function

Extract some Real values (floating point value) from a String (Message).
These extracted Real values are stored in the Float array.

Input Parameters :

MSG_	Message	The Message (String) to be extracted. (The length of a String is 1 to 255 bytes (characters))
ARY_NO_	Integer	The Float array No. to store these extracted Real values. I-8xx7, I-718xEG/XG: 1 ~ 6 (The Float array & Integer array use the same memory. Please use them carefully.) W-8xx7/8xx6: 1 ~ 18

Return :

NUM_	Integer	The total number of Real values been extracted. (0 ~ 128) If returns -1, it means format error.

Note :

1. The Real value been extracted can be Read / Write by using "ARY_F_R" and "ARY_F_W" functions
2. The delimiter between Real values in the String should be "SPACE" character or "Comma" or "TAB" or "ENTER" or "NEW LINE" characters.
3. If the related Value in the String is not correct Real format, for example - ' 1.23 , 2.45A , 3.0 , 2+3 ' , the 2nd & 4th value are not correct Real format, this function returns -1.
4. For using MSG_F and MSG_N, the driver must be the version listed below or higher version:

> I-7188EG : since ver. 2.17
> I-7188XG : since ver. 2.15
> I-8xx7   : since ver. 3.19
> W-8xx7   : recommend upgrade to ver. 3.36 or higher version

Example:

(1.) If given MSG_ = '2.3 , -567.002   +0.0025 1   , 1.4E-5' and ARY_NO_ = 1 , it will return NUM_ as 5. The five REAL values are stored in Float array No.1 's address 1 to 5 respectively. addr.1 = 2.3 , addr.2 = -567.002 , addr.3 = 0.0025 , addr.4 = 1.0 , addr.5 = 1.4E-5

(2.) If given MSG_ = '4.01 , -8.09 , +-3.45' and ARY_NO_ = 2 , it will return NUM_ as -1. Because the 3rd value +-3.45 is not a correct REAL format

Demo program:

1. wpdmo_52.pia at WP-8xx7 CD-ROM:\napdos\isagraf\ wp-8xx7\demo\
2. wpdmo_52.pia at ftp://ftp.icpdas.com/pub/cd/winpac-8xx7/napdos/isagraf/ wp-8xx7/demo/

MSG_N

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
   msg_n
─MSG_
─ARY_N  NUM─
```
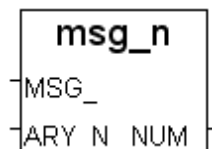
Description : C_Function

Extract some Integer values from a String (Message).
These extracted Integer values are stored in the Integer array.

Input Parameters :

MSG_	Message	The Message (String) to be extracted. (The length of a String is 1 to 255 bytes (characters).)
ARY_NO_	Integer	The Integer array No. to store these extracted Integer values I-8xx7, I-718xEG/XG: 1 ~ 6 (the Float array & Integer array use the same memory. Please use them carefully.) W-8xx7/8xx6: 1 ~ 18

Return :

NUM_	Integer	The total number of Integer values been extracted. (0 ~ 128) If returns -1, it means format error.

Note :

1. The Integer value been extracted can be Read / Write by using "ARY_N_R" and "ARY_N_W" functions.
2. The delimiter between Integer values in the String should be "SPACE" character or "Comma" or "TAB" or "ENTER" or "NEW LINE" characters.
3. If the related value in the String is not correct Integer format, for example - ' 123 , -8G , 3', the 2nd value is not correct Integer format, this function will return -1
4. For using MSG_F and MSG_N, the driver must be the version listed below or higher version:
   I-7188EG : since ver. 2.17
   I-7188XG : since ver. 2.15
   I-8xx7   : since ver. 3.19
   W-8xx7   : recommend upgrade to ver. 3.36 or higher version

Example:

(1.) If given MSG_ = '3 , -567   +02' and ARY_NO_ = 1 , it will return NUM_ as 3. The three Integer values are stored in Integer array No.1 's address 1 to 3 respectively. addr.1 = 3 , addr.2 = -567 , addr.3 = 2

(2.) If given MSG_ = '401 , 3A , +-345' and ARY_NO_ = 2 , it will return NUM_ as -1. Because the 2nd value 3A and the 3rd value +-345 are not correct Integer format

Demo Program :

1. wpdmo_53.pia at WP-8xx7 CD-ROM:\napdos\isagraf\wp-8xx7\demo\
2. wpdmo_53.pia at ftp://ftp.icpdas.com/pub/cd/ winpac-8xx7/napdos/isagraf/wp-8xx7/demo/

MSGARY_R

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 msgary_r
ADDR_  Msg_
```

Description : C_Function
    Read a String from the Message array

Input Parameters :
    ADDR_   Integer   Address to read from, 1 ~ 1024
Return :
    MSG_   Message   The String returned (The length is between 0 to 255)

Example: Please refer to Chapter 11 & the wdemo_06 of Wincon-8xx7.

MSGARY_W

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
 msgary_w
ADDR_
Msg_     Q_
```
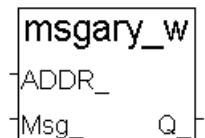
Description : C_Function
    Write a String to the Message array

Input Parameters :
    ADDR_   Integer   Address to write to, 1 ~ 1024
    Msg_   Message   The String to write (The length is between 0 to 255)
Return :
    Q_   Boolean   True: ok, False: fail.

Example: Please refer to Chapter 11 & the wpdmo_06 of Wincon-8xx7.

PID_AL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block

Example:
Please refer to Chapter 11 - Demo_18, and ICP DAS CD-ROM:
\Napdos\ISaGRAF\8000\English_Manu\PID_AL.Complex PID algorithm implementation.htm
or wpdmo_08 in WP-8xx7 CD-ROM:\napdos\isagraf\wincon\demo\ or
ftp://ftp.icpdas.com/pub/cd/winpac-8xx7/napdos/isagraf/wp-8xx7/demo/

PID_AL is a function provided by CJ International (copyright). CJ's description is listed below:

```
Author  : EDS
Product : ISaGRAF V3
Date    : 26 aug 96
File    : PID_AL.Complex PID algorithm implementation.htm
Subject : Complex PID algorithm implementation
Keywords: PID - PID_AL
```
_____
```
A pid is a process regulator. Using the Feed Back concept, an output
is regulated according to the difference between what it is really
and what it should be.
```
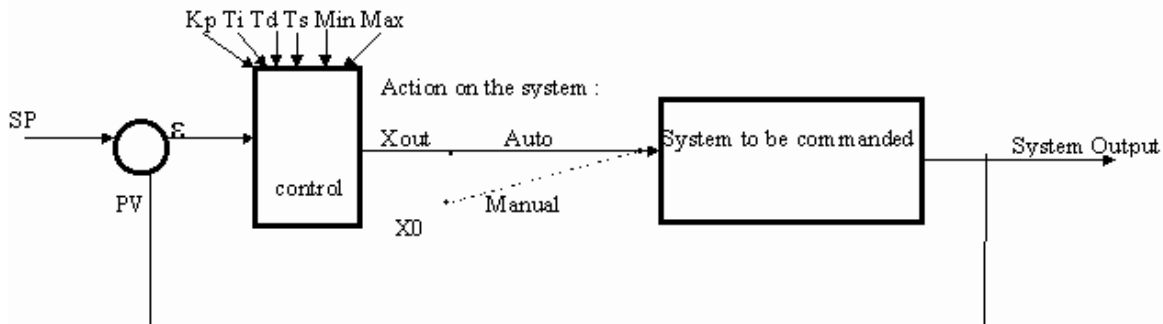


```
PV is the output. At the entry the error e is calculated as the
difference SP-PV. The control calculates the action to do on the system
to keep the regulation.
System to command : for the simulation, it is a second order system that
has been simulated.
On the contrary, on a non regulated system we have :
```
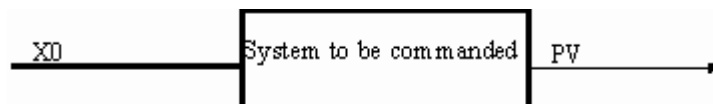


```
Major points of the PID described here :
SP is the set point : value wanted at the output.
X0 (open loop case, in manual mode) is the non regulated value entering
   the system.
```

Xout (close loop case with regulation) is the regulated value entering
    the system.

CJ PID-Al description
call: - Auto:automatic or manual mode
Pv :Process output value
Sp :Set point value
X0 :Adjustment value: In manual mode, output pid controller equal to X0
Kp :Proportionality constant
Ti :Integral time constant
Td :derivative time constant
Ts :Sampling period
Min, Max : range of accepted values of Xout
return: - Xout : Command
prototype: - PID (Auto,Pv,Sp,X0,Kp,Ti,Td,Ts, Min, Max);
command := PID.Xout;
notes: - automatic mode must be set to false at init
- The implemented model is :

$$u(t) = Kp(\varepsilon(t) + \frac{1}{Ti}\int_0^t \varepsilon(t)d(t) + Td\frac{d\,\varepsilon(t)}{dt})$$

Theoretical Continue formula

$$u(k) = Kp(\varepsilon(k) + \frac{Ts}{Ti}I(k) + \frac{Td}{Ts}[\,\varepsilon(k) - \varepsilon(k-1)])$$

Implemented Discret formula

$$I(k) = I(k-1) + \varepsilon(k)Ts$$

_____

Copyright CJ International 1996.

PLC_mode

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
plc_mode
MODE      Q
```
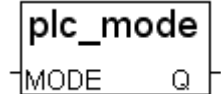
Description : C_Function
Change Wincon PLC speed

Input Parameters :
MODE_   Integer   Can be 0 , 1, 2, or 3
0: Fast Mode, default setting, the minimum PLC scan time is 4 ms
1: Slow Mode, the minimum PLC scan time is about 6 to 7 ms
2: Slower Mode, the minimum PLC scan time is about 9 to 11 ms
3 or other value: Slowest Mode, the minimum PLC scan time is
about 19 to 21 ms

Return :
Q_   Boolean   Return TRUE always

Note :
1. The "PLC_mode" is supported since driver of version 3.24B
2. The system's default setting is "Fast Mode"
3. User may call "PLC_mode( )" in the first PLC scan to change the PLC speed.
4. The reason to slow down the PLC speed is to improve the speed performance of other HMI
program running with ISaGRAF driver at the same time. Ex: running Indusoft with ISaGRAF
in the same Wincon.

Example :
```
(* TMP is declared as Boolean internal variable *)
(* INIT is declared as Boolean internal variable and init at TRUE *)
if  INIT  then
  INIT  :=  False ;   (*Only do it once in the 1st PLC scan*)
  TMP  :=  PLC_mode(2) ;   (*Set PLC speed to 2:slower mode*)
end_if ;
```

The PWM functions listed below are C_Function, please refer to User manual Section 3.7 for detail information and examples.

PWM_DIS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Disable PWM output**

PWM_EN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Enable PWM input.**

PWM_EN2

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Enable PWM output to output some pulse**

PWM_OFF

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Set parallel D/O to False immediately**

PWM_ON

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Set parallel D/O to TRUE immediately**

PWM_SET

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Dynamical change the ON_, OFF_ & NUM_ setting**

PWM_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Get PWM status**

PWM_STS2

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC
Function   **Get the pulse number which has been output by "pwm_en2" or "pwm_en"**

R_MB_ADR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
Read Boolean or Integer variable by using Modbus address
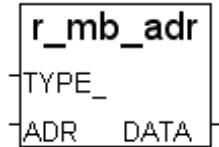
```
 r_mb_adr
TYPE_
ADR    DATA
```

Input Parameters :
TYPE_	Integer	0: Boolean variable; 1: Integer variable
ADR_	Integer	The Modbus address to read,
		Valid range: 1~4095 for I-8xx7 & I-7188xG; 1~8191 for Wincon.

Return :
DATA_	Integer	The read Integer value (if TYPE_ is Boolean, 0: False, 1: True)

Note :
1. Please use R_MB_REL function to read "REAL" variable.
2. If no variable defined with the given Modbus address, this function returns 0.
3. If TYPE_ is given as Integer however the related variable is "Boolean" typed, the returned value is (0: False, 1: True)
4. If TYPE_ is given as Integer however the related variable is "Real" typed, the related 32-bit is copied to DATA_. User can use "int_real" function to map this 32-bit Integer to a Real value. (It is better to use "R_MB_REL" to read "Real" variable)
5. If TYPE_ is given as Boolean however the related variable is not "Boolean" typed, returns 0.
6. If Long Integer value (32-bit Integer) is to be delivered to HMI via Modbus protocol, it should occupy 2 Modbus address Number. Please refer to Section 4.2 of "User's Manual Of The ISaGRAF PAC".
7. "**R_MB_ADR(1,9994)" and "R_MB_ADR(1,9995)**" are special usage for sending email with attached file via Ethernet communication. Please refer to Chapter 12.
8. "**R_MB_ADR(1,9998)" and "R_MB_ADR(1,9999)**" are special usage for detecting controller fault. Please refer to section 10.6

R_MB_REL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
Read Real variable by using Modbus address
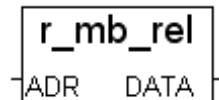
```
 r_mb_rel
ADR    DATA
```

Input Parameters :
ADR_	Integer	The Modbus address to read,
		Valid range: 1~4095 for I-8xx7 & I-7188xG, 1~8191 for Wincon

Return :
DATA_	Real	The read Real value

Note :
1. Please make sure the related variable is typed of "Real". If it is "Integer", please use "R_MB_ADR" function.
2. If the related variable type is not "Analog" (Real or Integer), this function returns 1.23E-20
3. If no variable defined with the given Modbus address, this function returns 1.23E-20
4. If Real value is to be delivered to HMI via Modbus protocol, it should occupy 2 Modbus address No. Please refer to Section 4.2 of "User's Manual Of The ISaGRAF PAC".
5. If the related variable type is not "Real", it may cause Local Fault of No.103. Please refer to Section 10.6.

R_TRIG

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : Standard_Function
Detect a rising edge of a Boolean variable.
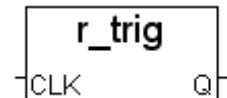
Input Parameters :
    CLK   Boolean   Any Boolean variable

Return :
    Q   Boolean   TRUE: CLK rises from FALSE to TRUE
                                          FALSE: all other cases

Example :
(* FBD Program *)
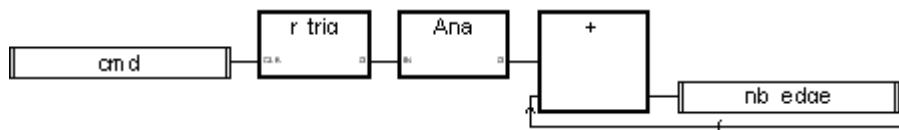
(* ST Equivalence: We suppose R_TRIG1 is an instance of R_TRIG block. *)

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

(* IL Equivalence: *)
LD   cmd
ST   R_TRIG1.clk
CAL   R_TRIG1
LD   R_TRIG1.Q
ANA
ADD   nb_edge
ST   nb_edge

<u>The following RDN functions are Redundant related functions, please refer to The ISaGRAF
User Manual Chapter 20 for detail information and examples.</u>

RDN_A

```
┌──────────┐
│  rdn_a   │
┤NetW_     │
┤AMOUN    Q├
└──────────┘
```

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
C_Function   Set a "Variable Array" as Redundant synchronous data.

RDN_B

```
┌──────────┐
│  rdn_b   │
┤B        Q├
└──────────┘
```
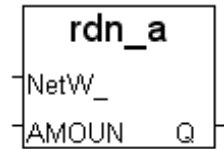
□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
C_Function   Set a Boolean variable as Redundant synchronous data.

RDN_F

```
┌──────────┐
│  rdn_f   │
┤F        Q├
└──────────┘
```
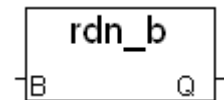
□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
C_Function   Set a Real variable as Redundant synchronous data.

RDN_N

```
┌──────────┐
│  rdn_n   │
┤N        Q├
└──────────┘
```
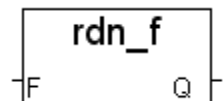
□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
C_Function   Set an Integer variable as Redundant synchronous data.

RDN_T

```
┌──────────┐
│  rdn_t   │
┤T        Q├
└──────────┘
```
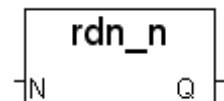
□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC
C_Function   Set a Timer variable as Redundant synchronous data.
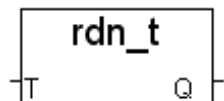
Note:
1. The ISaGRAF program for the 2 redundant Wincon controllers is the same project.
2. All I-7K & I-87K remote IO function blocks should be located on the top of the ISaGRAF
   project.
3. Before using RDN_functions, please make sure the "rdn" of "IO complex equipment" is well
   connected in the ISaGRAF IO connection window.
4. RDN_B, RDN_F, RDN_N, RDN_T, RDN_A must be called only in the 1st PLC scan.
5. Max number of synchronous bytes is 6000 bytes. Boolean: 1 byte; Integer & Real & Timer is
   4 bytes.
6. Please refer to Chapter 20 for more information.

REAL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : Standard_Function

Convert any variable to a REAL one.

```
        Real
 ─IN         Q─
```
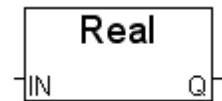
Input Parameters :

IN	Any	Any non-Real analog value (no message)

Return :

Q	Real	If IN is FALSE, return 0.0; if IN is TRUE, return 1.0.
		If IN is a Timer, it return the number of millisecond.
		If IN is an analog Integer, it return the equivalent number.

Example :

(* FBD Program *)
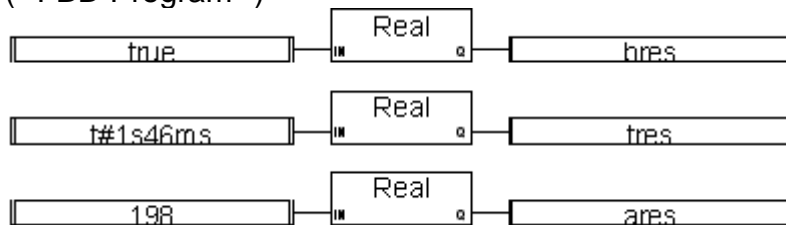
(* ST Equivalence: *)
bres := REAL (true);   (* bres is 1.0 *)
tres := REAL (t#1s46ms);   (* tres is 1046.0 *)
ares := REAL (198);   (* ares is 198.0 *)

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

REA_STR2

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
      Convert a Real value to a String with fixed digit number

```
rea_str2
REAL_
DEC    STR
```
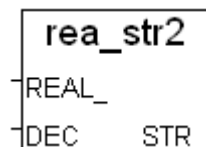
Input Parameters:
    REAL_   Real   The Real value to convert
    DEC_   Integer   The digit number after the dot "." symbol (0 ~ 5)

Return:
    STR_   Message   The String returned (Max. length is 13).
                              Ex: DEC_ = 2
                              1.234   ---> '1.23'
                              123456.0   ---> '123456.00'
                              0.00001234 ---> '0.00'

Note: Converting a String to a Real please use "STR_REAL".
Example: Please refer to Chapter 16 and demo_38, demo_39

REAL_INT

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
      Map a Real value to a Long Integer.

```
Real_Int
Real    Long
```
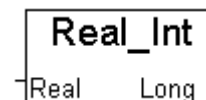
Input Parameters:
    Real_   Real   The Real value to map
Return:
    Long_   Integer   The 32-bit Integer after mapping
Note:
1. Mapping an Integer to a Real please use "Int_Real".
2. Converting a Real to an Integer please use ANA( ), Ex: -76.345 → -76

REAL_STR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function
      Convert a Real value to a String.

```
real_str
REAL    STR
```
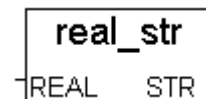
Input Parameters :
    REAL_   Real   The Real value to convert
Return :
    STR_   Message   The returned String (Max. length is 13),.
                          1.234   ---> '1.234'
                          123456789.0 ---> '1.23457E+008'
                          0.00001234   ---> '1.234E-005'

RETAIN_A
■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  retain_a
-X_
-NetW_
-AMOUN
-ADR      Q-
```
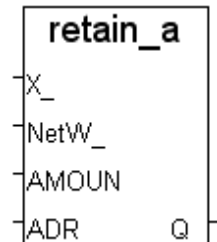
Description : C_Function
Set a "Variable array" to be as "retained variable" by using its 1st element's Network address No.

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
      iP-8xx7 or µPAC-7186EG + X607 (or X608)
Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
      WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters :

X_	Message	'B' or 'b': Boolean, 'N' or 'n': Integer, 'F' or 'f': Real, 'T' or 't': Timer
NetW_	Integer	Network address No. for the 1st element of the "Variable Array". Target 1/2/3 : 1~4095 , Target 4: 1~8191
Amount_	Integer	the size of the "Variable Array", Valid range: 1~255, Ex: CNT[0..15] has size of 16, ABC[0..7] has size of 8.
ADR_	Integer	The retained address inside the retained memory for this variable. Target 1/2/3: Boolean & Timer: 1~256; Integer & Real: 1~1024. Target 4: Boolean (1 ~ 2048), Timer (1 ~ 1024), Integer and Real (1 ~ 4096)

Return :

Q_	Boolean	True: ok; False: error (ex: cannot find retained variable)

Note :
1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions, please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF IO connection window is well connected.
    a. I-7188EG/XG: "X607_608"
    b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
    Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
        [DEBUG]
        arrays=1
3. Before using "Retain_A", please assign a network address number to "network address" column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex: Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

RETAIN_B

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
retain_b
 B_
 ADR      Q
```
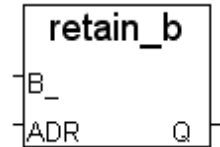
Description : C_Function

Set a Boolean variable to be as "Retained Variable"

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
  Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
        iP-8xx7 or µPAC-7186EG + X607 (or X608)
  Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
  Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
  Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
        WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters:

B_	Boolean	B_ should be a Boolean variable name, not a constant value.
ADR_	Integer	The retained address inside the retained memory for this Boolean variable. Target 1 & 2 & 3: 1 ~ 256; Target 4: 1 ~ 2048

Return:

Q_	Boolean	True: ok; False: error (ex: cannot find retained variable)

Note:

1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions, please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF IO connection window is well connected.
    a. I-7188EG/XG: "X607_608"
    b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
        [DEBUG]
        arrays=1
3. Before using "Retain_A", please assign a network address number to "network address" column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex: Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW 7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

RETAIN_F

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 ┌─────────────┐
 │  retain_f   │
 ┤F_           │
 ┤ADR        Q ├
 └─────────────┘
```
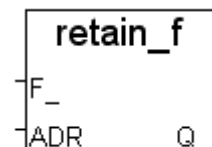
Description : C_Function
        Set a Real variable to be as "Retained Variable"

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
   Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
              iP-8xx7 or μPAC-7186EG + X607 (or X608)
   Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
   Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
   Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
              WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters :
   F_   Real   F_ should be a Real variable name, not a constant value.
   ADR_   Integer   The retained address inside the retained memory for this Real
                           variable.
                           Target 1 & 2 & 3: 1~1024; Target 4: 1 ~ 4096

Return :
   **Q_**   Boolean   True: ok;   False: error (ex: cannot find retained variable)

Note :
1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions,
   please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF
   IO connection window is well connected.
      a. I-7188EG/XG: "X607_608"
      b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in
   the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF
   workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to
   assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
           [DEBUG]
           arrays=1
3. Before using "Retain_A", please assign a network address number to "network address"
   column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex:
   Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7:
   S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW 7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have
   already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

RETAIN_N

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────────────┐
│    retain_n     │
│                 │
│─N_              │
│                 │
│─ADR         Q─  │
└─────────────────┘
```
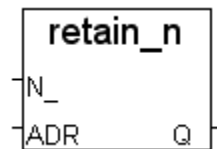
Description : C_Function

Set an Integer variable to be as "Retained Variable"

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
  Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
          iP-8xx7 or µPAC-7186EG + X607 (or X608)
  Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
  Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
  Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
          WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters :

N_   Integer   N_ should be an Integer variable name, not a constant value.
ADR_   Integer   The retained address inside the retained memory for this
                         Integer variable.
                         Target 1 & 2 & 3: 1~1024; Target 4: 1 ~ 4096

Return :

**Q_**   Boolean   True: ok;   False: error (ex: cannot find retained variable)

Note :

1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions,
   please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF
   IO connection window is well connected.
      a. I-7188EG/XG: "X607_608"
      b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in
   the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF
   workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to
   assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
           [DEBUG]
           arrays=1
3. Before using "Retain_A", please assign a network address number to "network address"
   column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex:
   Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7:
    S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW 7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have
    already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

RETAIN_T

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
┌─────────────┐
│  retain_t   │
│             │
┤T_           │
│             │
┤ADR       Q ├
└─────────────┘
```
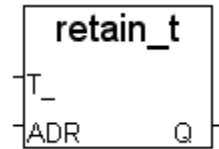
Description : C_Function

Set a Timer variable to be as "Retained Variable"

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
    Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
            iP-8xx7 or μPAC-7186EG + X607 (or X608)
    Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
    Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
    Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
            WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters :

T_	Timer	T_ should be a Timer variable name, not a constant value.
ADR_	Integer	The retained address inside the retained memory for this Timer variable.
		Target 1 & 2 & 3: 1 ~ 256; Target 4: 1 ~ 1024

Return :

|		
Q_	Boolean	True: ok; False: error (ex: cannot find retained variable)

Note :

1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions, please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF IO connection window is well connected.
    a. I-7188EG/XG: "X607_608"
    b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
    Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
            [DEBUG]
            arrays=1
3. Before using "Retain_A", please assign a network address number to "network address" column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex: Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7: S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW 7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

RETAIN_X

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
              retain_x
           ─X_
           ─NetW_
           ─ADR        Q─
```
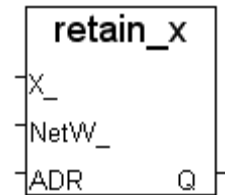
Description : C_Function

Set a variable to be as "Retained Variable" by using its Network
address number.

* Please refer to Chapter 10 for detail information.
* ISaGRAF controllers support "Retained Variable" function since the following driver version:
   Target 1: I-8417/8817/8437/8837 + S-256 or S-512 (Retained variable), since driver Ver 3.13
         iP-8xx7 or µPAC-7186EG + X607 (or X608)
   Target 2: I-7188EG + X607 or X608 (Retained variable), since driver Ver 2.11
   Target 3: I-7188XG + X607 or X608 (Retained variable), since driver Ver 2.09
   Target 4: W-8xx7/8xx6 + S-256 or S-512 (Retained variable), since driver Ver 3.26
         WP-8xx7, VP-2Xw7, XP-8xx7-CE6, XP-8xx7-Atom-CE6

Input Parameters :

|		
X_	Message	'B' or 'b': Boolean, 'N' or 'n' : Integer, 'F' or 'f' : Real, 'T' or 't' : Timer
NetW_	Integer	By using its Network address No. Target 1/2/3 : 1 ~ 4095; Target 4: 1 ~ 8191
ADR_	Integer	The retained address inside the retained memory for this variable. Target 1/2/3 : Boolean and Timer (1 ~ 256), Integer and Real (1 ~ 1024); Target 4: Boolean (1 ~ 2048), Timer (1 ~ 1024) , Integer and Real (1 ~ 4096)

Return :

Q_	Boolean	True: ok; False: error (ex: cannot find retained variable)

Note :

1. Before using Retain_X, Retain_A, Retain_N, Retain_B, Retain_F & Retain_T functions,
   please make sure that the device (listed below) of "IO complex equipment" in the ISaGRAF
   IO connection window is well connected.
      a. I-7188EG/XG: "X607_608"
      b. I-8417/8817/8437/8837 & W-8037/8337/8737: "S256_512"
2. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in
   the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF
   workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to
   assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the beginning of the file content:
        [DEBUG]
        arrays=1
3. Before using "Retain_A", please assign a network address number to "network address"
   column in the ISaGRAF dictionary window. Please **DO NOT** check the "Retain" box. Ex:
   Assign "Network Address"= 1A (Hex., Decimal 26) to Integer "variable array": CNT[0..7].
4. Please make sure the retain memory (7188EG/XG: X607/X608, I-8xx7 & W-8xx7:
   S-256/S-512) is well plugged in the controller.
5. The WP-8xx7, VP-2xW 7, XP-8xx7-CE6, XP-8xx7-Atom-CE6, iP-8xx7 and VP-2117 have
   already built-in a battery backup memory for retain variables.
6. If the PAC is WP-5xx7, please refer to FAQ-155.

S_B_R

```
S_B_R
ADR    BOO
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

Description: C_Function

Read one Boolean from the volatile SRAM

Input Parameters:

ADR_   Integer   The address to read, one Boolean occupy 1 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

Return:

BOO_   Boolean   The Boolean value been read, 0=FALSE,   not 0 = TRUE

S_B_W

```
S_B_W
ADR_
NUM_
B1_
B2_
B3_
B4       Q
```

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC
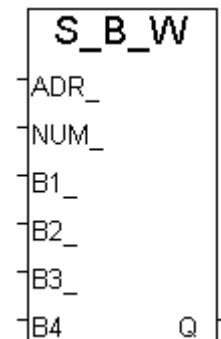
Description : C_Function

Write up to 4 Booleans to the volatile SRAM

Input Parameters :

ADR_   Integer   The starting address to write, 1 Boolean occupy 1 byte.
S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

NUM_   Integer   How many Boolean to write, 0 ~ 4
B1_~B4_   Boolean   The Boolean value to write

Return :

Q_   Boolean   Ok: TRUE, fail: FALSE
                               The Boolean value will be stored is FALSE=0, TRUE=1

Example : demo_40, demo_41, demo_42, demo_44
Please refer to Section 10.3.

S_BY_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
┌─────────┐
│  S_BY_R │
├─────────┤
┤ADR    N├
└─────────┘
```
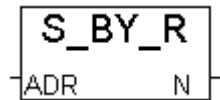
Description : C_Function
Read one Byte from the volatile SRAM

Input Parameters :
ADR_   Integer   The address to read, one Byte occupy 1 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

Return :
N_   Integer   The Byte value been read, 0 ~ 255

```
┌─────────┐
│  S_BY_W │
├─────────┤
┤ADR_     │
┤NUM_     │
┤N1_      │
┤N2_      │
┤N3_      │
┤N4     Q├
└─────────┘
```

S_BY_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC
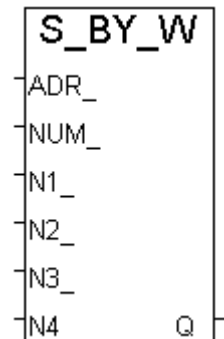
Description : C_Function
Write up to 4 Bytes to the volatile SRAM

Input Parameters :
ADR_   Integer   The starting address to write, 1 Byte occupy 1 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

NUM_   Integer   How many Byte to write, 0 ~ 4
N1_~N4_   Boolean   The Byte value to write, 0~255

Return :
Q_   Boolean   Ok: TRUE, fail: FALSE

Example : demo_40, demo_41, demo_42, demo_44
Please refer to Section 10.3.

S_DL_DIS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
s_dl_dis

              Q
```
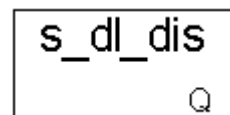
Description : C_Function
Disable the download permission, so that PC can not download data to the SRAM
Return :
    Q_   Boolean   Ok: TRUE, fail: FALSE

S_DL_EN

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
s_dl_en

              Q
```
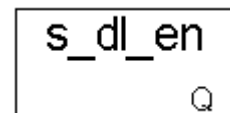
Description : C_Function
Enable the download permission for PC to download data to the volatile SRAM
Return :
    Q_   Boolean   Ok: TRUE, fail: FALSE
Note:
The default setting is "Disable". S_DL_EN should be called, then PC can download data to the volatile SRAM.

S_DL_RST

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function
Reset the Download Status to "-1:No action" for the volatile SRAM.

```
s_dl_rst

              Q
```

Return :
    Q_   Boolean   Ok: TRUE, fail: FALSE

S_DL_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description: C_Function
Get PC's download status for the volatile SRAM

```
s_dl_sts

              STS
```

Return:
    STS_   Integer   -1: no action
                             1: PC is downloading data to the SRAM now
                             2: download accomplished
                             3: PC request to download, but download never been accomplished (e.g. the communication problem).
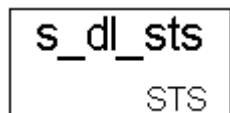
Note: S_DL_RST can be called to reset the status to -1 (reset to "No action")

Example: demo_40, demo_41, demo_42, demo_44
        Please refer to Section 10.3.

S_FL_AVL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
   ┌─────────────┐
   │   s_fl_avl  │
  ─┤ID_          │
  ─┤HEAD_        │
  ─┤TAIL_     Q ├─
   └─────────────┘
```
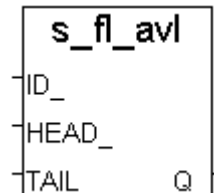
Description : C_Function
Set one file's current available byte No. for the volatile SRAM

Input Parameters :

ID_   Integer   File identifier No. (1 ~ 8)
HEAD_   Integer   The current available starting byte No.
TAIL_   Integer   The current available ending byte No.

(HEAD_, TAIL_) must resides inside the area of the associate file (Please refer to "S_FL_INI"), or Q_ will return FALSE

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

Ex:

A file with ID_ = 1 resides at byte number of 1 ~ 20000, it can store up to 20000 bytes.

1. If setting one of HEAD_ or TAIL_ to -1, it means no data of the file is available. When you load this file from PC, its size is 0 byte.

2. If setting HEAD_=1, TAIL_=1000, the current available data of the file will be at 1 ~ 1000 inside the volatile SRAM. It means when you load this file from PC, its size is 1000 bytes.

3. If setting HEAD_=10001, TAIL_=5000, the current available data of the file will be at 10001 ~ 20000 and then continued with 1 ~ 5000 inside the volatile SRAM. It means when you load this file from PC, its size is 15000 bytes.

4. If setting HEAD_=1000, TAIL_=1000, it means no data of the file is available. When you load this file from PC, its size is 0 byte.

Return :

Q_   Boolean   TRUE: ok, FALSE: fail.

Note: S_FL_INI should be called once before S_FL_AVL is called

Example : demo_40, demo_41, demo_42, demo_44, demo_71 (Chapter 11.3.7)
Please refer to Chapter 10.3.

S_FL_INI

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
 s_fl_ini
ID_
NAME_
BEGIN
END        Q
```

Description : C_Function
Initiate one file's name & location for the volatile SRAM
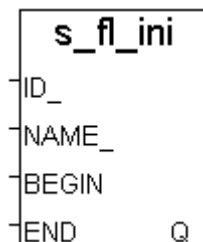
Input Parameters :
ID_	Integer	File identifier No. (1 ~ 8)
NAME_	Message	File name; Max. 8 char.(for name) + Max. 3 char.(for extension).

Valid char. are A ~ Z, a ~ z, _ , 0 ~ 9, and the 1st should be A ~ Z or a ~ z. Ex: "data1.txt", "A1234567.bin".

BEGIN_	Integer	The begin byte No. of the file. BEGIN_ must less than END_.
END_	Integer	The end byte No. of the file. BEGIN_ must less than END_.

S256: 1 ~ 249,856
S512: 1 ~ 512,000
X607: 1 ~ 118,784
X608: 1 ~ 512,000
Ex, BEGIN_=101, END_=5000 :
the file resides at 101 ~ 5000 inside the SRAM.

Return:
Q_	Boolean	TRUE: ok, FALSE: fail

S_FL_RST

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
 s_fl_rst
ID         Q
```

Description : C_Function
Reset the status of file to "Not been load by PC yet" for the volatile SRAM.
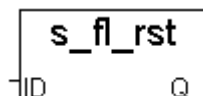
Input Parameters:
ID_	Integer	File identifier No. (1 ~ 8)

Return:
Q_	Boolean	TRUE: ok, FALSE: fail

Note:
1. S_FL_INI should be called first.
2. S_FL_RST can be called to reset the status to -1 (reset to "Not been load by PC yet").

Example : demo_40, demo_41, demo_42, demo_44, demo_71 (Chapter 11.3.7)
Please refer to Chapter 10.3.

S_FL_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
s_fl_sts
ID      END
```
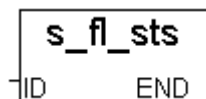
Description : C_Function
Get file's status and the end byte No. that has been load by PC for the volatile SRAM.

Input Parameters:
ID_   Integer   File identifier No. (1 ~ 8)

Return:
END_   Integer   The end byte No. that has been load by PC
                           Not been load: -1
                           S256:   1 ~ 249,856
                           S512:   1 ~ 512,000
                           X607:   1 ~ 118,784
                           X608:   1 ~ 512,000

For ex.,

A file with ID_ = 1 is located at byte No.: 1 ~ 20000, it can store up to 20000 bytes. And its current available data is setting at 1001 ~ 10000 inside the volatile SRAM.

1. If returned END_ is -1, it means PC hasn't loaded the data yet.

2. If returned END_ is 10000 (Normally the value is equal to the current available ending byte No.), it means PC has loaded the data from 1001 ~ 10000.

3. If returned END_ is 8000, it means PC has loaded data from 1001 ~ 8000.

Note:
1. S_FL_INI should be called first.
2. S_FL_RST can be called to reset the status to -1 (reset to "PC hasn't load it yet")

Example : demo_40, demo_41, demo_42, demo_44, demo_71 (Chapter 11.3.7)
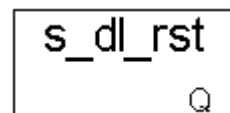   Please refer to Section 10.3.

S_M_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function

    Read one String from the volatile backup SRAM

Input Parameters:

ADR_	Integer	The address to read from
		S256: 1 ~ 249,856 (1 ~ 16#3D000)
		S512: 1 ~ 512,000 (1 ~ 16#7D000)
		X607: 1 ~ 118,784 (1 ~ 16#1D000)
		X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_	Integer	Max length of the String to read, 0 ~ 255

Return:

    STR_   Message   The String been read

Ex: If the data inside the SRAM are 16#31, 16#32, 16#33, 16#34, 16#35, **0**, 16#37, 16#38, ...

        LEN_=0   ---->   STR_= '' (Null string)
        LEN_=3   ---->   STR_= '123'
        LEN_=5   ---->   STR_= '12345'
        LEN_=6   ---->   STR_= '12345'
        LEN_=7   ---->   STR_= '12345'
        LEN_=100 ---->   STR_= '12345'
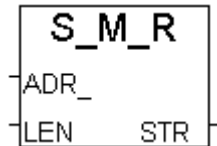
S_M_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function

    Write one String to the volatile backup SRAM

Input Parameters:

ADR_	Integer	The address to write from.
		S256: 1 ~ 249,856 (1 ~ 16#3D000)
		S512: 1 ~ 512,000 (1 ~ 16#7D000)
		X607: 1 ~ 118,784 (1 ~ 16#1D000)
		X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_	Integer	The length of the String to write, 0 ~ 255
STR_	Message	The string to write.

    Example:

   LEN_=0 , STR='12345' ----> No data can be write into
   LEN_=1 , STR='12345' ----> 16#31 (write 1 byte)
   LEN_=3 , STR='12345' ----> 16#31, 16#32, 16#33 (write 3 bytes)
   LEN_=7 , STR='12345' ----> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0 (write 7 bytes)
   LEN_=100 , STR='12345' --> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0, 0, … (write 100
                                 bytes)

**Return :**

    Q_   Boolean   Ok: TRUE, Fail: FALSE

Example : demo_40, demo_41, demo_42, demo_44, demo_71 (Chapter 11.3.7)
        Please refer to Section 10.3.

S_MB_ADR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
 s_mb_adr
ADDR_
AMOUN
INTER      Q
```

Description : C_Function

Assign Modbus network address to "Variable Array" (refer to Sec.2.6.1)

Target:

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

Input Parameters:

ADDR_	Integer	The beginning Modbus Network Address No. which is assigned to this "Variable Array" in the dictionary windows. Target 1: 1~4095; Target 2: 1~8191.
AMOUNT_	Integer	The size of the "Variable Array". Valid range: 1~255. Ex: CNT[0..15] has size of 16, ABC[0..7] has size of 8.
INTERVAL_	Integer	0 : sequence Modbus address (e.g. 10, 11, 12, 13 ...) 1 : sequence odd/even Modbus address (ex. 4, 6, 8 ... or 1, 3, 5, 7 …)

Return:

Q_	Boolean	True: Ok; False: ADDR_ or AMOUNT_ or INTERVAL_ is error.

Note:

1. This function should be called in the **1st** PLC scan cycle. It doesn't work in other PLC scan cycle.

2. The beginning Modbus Network Address of the "Variable Array" should be assigned in the ISaGRAF Dictionary windows. For example, assign Network address No. = 21 to "Variable Array" of L[0..9]

3. To declare a "Variable Array" for ISaGRAF version 3.4 (or 3.5), please edit the "isa.ini" file in the ISaGRAF sub-directory "C:\ISAWIN\EXE\". And then when you open the ISaGRAF workbench, there will be a "DIM" area in the Dictionary Declaration Windows for you to assign the size of the "variable array".
   Edit c:\isawin\exe\isa.ini file, add these 2 lines at the very beginning of the file content:
       [DEBUG]
       arrays=1

4. To view or control the value of "Variable array" when debug for the controller program, please use "Spy list" and insert the array variable name. For example, insert L[0], L[1]. Please refer to Chapter 9.12 of the "User's Manual of ISaGRAF PAC" for "Spy list"

5. Please declare and use "Variable Array" carefully. Don't declare the size too large, just declare the size you need, or it will consume lots of memory. And DON'T use the index that bigger than you declare. For example, if you declare **CNT[0..9]** , DO NOT use as below, or the program will have error at run time.
       For index := **0 to 10** do
        A := A + CNT[index] ;
       End_For ;
          (* the   **CNT[10]** **doesn't** exist in this example *)

6. Please assign the Modbus Network address carefully, do not conflict with other variables.

7. For more information about 'Redundancy' & 'Variable Array' of controller, please refer to the manual or website listed below.
   a. Wincon-8xx7 CD-ROM: \napdos\isagraf\wincon\english_manu\
   b. ftp://ftp.icpdas.com/pub/cd/wincon_isagraf/napdos/isagraf/wincon/english_manu/

S_MV

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
  ┌─────────────┐
  │    S_MV     │
 ─┤ADR1_        │
 ─┤NUM_         │
 ─┤ADR2_     Q_ ├─
  └─────────────┘
```

Description : C_Function

Copy some Bytes inside the volatile SRAM .
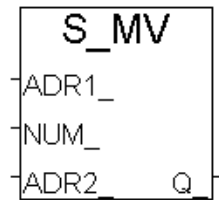
Input Parameters :

ADR1_	Integer	The starting position of the destination
		S256: 1 - 249856 (1 - 16#3D000)
		S512: 1 - 512000 (1 - 16#7D000)
		X607: 1 - 118784 (1 - 16#1D000)
		X608: 1 - 512000 (1 - 16#7D000)
NUM_	Integer	How many Bytes to copy, 0~512,000
ADR2_	Integer	The starting position to copy from

Return :

Q_	Boolean	Ok: TRUE; fail: FALSE.

S_N_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
┌─────────┐
│  S_N_R  │
├─────────┤
─┤ADR    N├─
└─────────┘
```

Description : C_Function

      Read one Integer from the volatile SRAM

Input Parameters :

   ADR_   Integer   The starting position to read from, one Integer occupy 4 bytes.
                                          S256: 1 ~ 249,856 (1 ~ 16#3D000)
                                          S512: 1 ~ 512,000 (1 ~ 16#7D000)
                                          X607: 1 ~ 118,784 (1 ~ 16#1D000)
                                          X608: 1 ~ 512,000 (1 ~ 16#7D000)
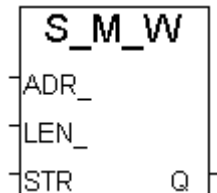
Return :

   N_   Integer   The Integer been read

Note : The integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte].
           Ex: An integer of 16#01020304 will be saved in the SRAM as [04] [03] [02] [01].

S_N_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
┌─────────┐
│  S_N_W  │
├─────────┤
─┤ADR_     │
─┤NUM_     │
─┤N1_      │
─┤N2_      │
─┤N3_      │
─┤N4     Q├─
└─────────┘
```

Description : C_Function

      Write one Integer to the volatile SRAM

Input Parameters :

   ADR_   Integer   The starting position to write, one Integer occupy 4 bytes.
                                          S256: 1 ~ 249,856 (1 ~ 16#3D000)
                                          S512: 1 ~ 512,000 (1 ~ 16#7D000)
                                          X607: 1 ~ 118,784 (1 ~ 16#1D000)
                                          X608: 1 ~ 512,000 (1 ~ 16#7D000)
   NUM_   Integer   How many Integers to write, 0 ~ 4
   N1_~N4_ Integer   The Integer value (32-bit, signed) to write
   Note : The Integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte].
           Ex: An Integer of 16#01020304 will be saved in the SRAM as [04] [03] [02] [01].

Return :

   Q_   Boolean   Ok: TRUE; fail: FALSE.

S_R_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
┌─────────┐
│  S_R_R  │
├─────────┤
┤ADR    R├
└─────────┘
```

Description : C_Function

      Read one Real value from the volatile SRAM
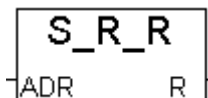
Input Parameters :

    ADR_   Integer   The address to read, one Real value occupy 4 bytes.

                                    S256: 1 ~ 249,856 (1 ~ 16#3D000)

                                    S512: 1 ~ 512,000 (1 ~ 16#7D000)

                                    X607: 1 ~ 118,784 (1 ~ 16#1D000)

                                    X608: 1 ~ 512,000 (1 ~ 16#7D000)

Return :

    R_   Real   the Real value been read (32-bit float)

Note :

1. The Real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte].
   Ex: Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F
2. If the data stored in the volatile SRAM is not Real type but use the S_R_R to read, it may cause Local Fault No.102. (please refer to Section 10.6)

```
┌─────────┐
│  S_R_W  │
├─────────┤
┤ADR_     │
┤NUM_     │
┤R1_      │
┤R2_      │
┤R3_      │
┤R4     Q├
└─────────┘
```

S_R_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

Description : C_Function

      Write one Real value to the volatile SRAM
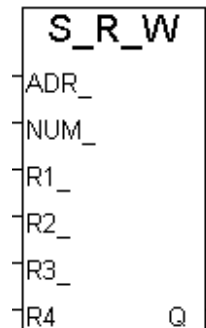
Input Parameters :

    ADR_   Integer   The starting position to write, one Real value occupy 4 bytes.

                                      S256: 1 ~ 249,856 (1 ~ 16#3D000)

                                      S512: 1 ~ 512,000 (1 ~ 16#7D000)

                                      X607: 1 ~ 118,784 (1 ~ 16#1D000)

                                      X608: 1 ~ 512,000 (1 ~ 16#7D000)

    NUM_   Integer   How many Real to write, 0 ~ 4

    R1_~R4_   Real   The Real value to write

    Note : The Real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte].
               Ex: Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F

Return :

    Q_   Boolean   Ok: TRUE, fail: FALSE

S_WD_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

```
S_WD_R
ADR      N
```

Description : C_Function

Read one Word value from the volatile SRAM

Input Parameters :

ADR_   Integer   The address to read, one Word value occupy 2 bytes.
                          S256: 1 ~ 249,856 (1 ~ 16#3D000)
                          S512: 1 ~ 512,000 (1 ~ 16#7D000)
                          X607: 1 ~ 118,784 (1 ~ 16#1D000)
                          X608: 1 ~ 512,000 (1 ~ 16#7D000)

Return :

N_   Integer   The Word value been read, -32768 ~ +32767

Note: The Word value written in the SRAM is [Low byte] [High byte].
          Ex: Word Value of 16#0102 is consists of [02] [01]

S_WD_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
S_WD_W
ADR_
NUM_
N1_
N2_
N3_
N4      Q
```

Description : C_Function

Write one Word value to the volatile SRAM

Input Parameters :

ADR_   Integer   The starting position to write, one Word value occupy 2 bytes.
                          S256: 1 ~ 249,856 (1 ~ 16#3D000)
                          S512: 1 ~ 512,000 (1 ~ 16#7D000)
                          X607: 1 ~ 118,784 (1 ~ 16#1D000)
                          X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_   Integer   How many Words to wrote, 0 ~ 4
N1_~N4_   Integer   The Word value to write, -32768 ~ 32767
Note: The Word value written in the SRAM is [Low byte] [High byte].
          Ex: Word Value of 16#0102 is consists of 2 bytes --> [02] [01]

Return :
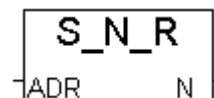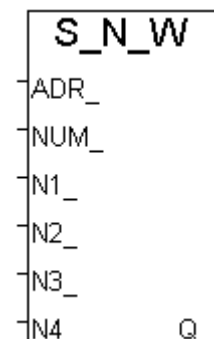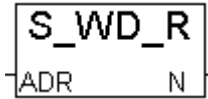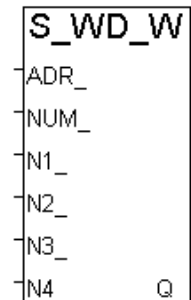
Q_   Boolean   Ok: TRUE, fail: FALSE

SET_LED

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function
    Display Message to the S-MMI controller

```
         set_led
       ┌──────────┐
   ────┤RUN_      │
   ────┤FSH_      │
   ────┤CLK_      │
   ────┤LED1_     │
   ────┤LED2_     │
   ────┤LED3_     │
   ────┤LED4_     │
   ────┤LED5_     │
   ────┤LED6_     │
   ────┤LED7    Q ├──
       └──────────┘
```

Input Parameters :

RUN_	Boolean	TRUE: display; FALSE: not display.
FLASH_	Integer	Set 1 for related position to flash.
		Ex: Set to 11 (0000011), the position 6 & 7 will blink.
		Set to 100001 (0100001), the position 2 & 7 will blink.
CLK_	Timer	The blinking period in ms, if < 50ms will act like 50ms
LED1_	Integer	The Message to display in position 1
LED2_	Integer	The Message to display in position 2
LED3_	Integer	The Message to display in position 3
LED4_	Integer	The Message to display in position 4
LED5_	Integer	The Message to display in position 5
LED6_	Boolean	To display position 6 if TRUE
LED7_	Boolean	To display position 7 if TRUE

Return :

Q_	Boolean	Always true, no meaning.

*   Please refer to appendix A.3 for the symbol table.

Example

```
                           set_led
                         ┌──────────┐
   ┌──────────────┐      │          │
   │    TRUE      ├──────┤RUN       │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │   1000110    ├──────┤FLASH     │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │   t#500ms    ├──────┤CLK       │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │      1       ├──────┤LED1      │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │      2       ├──────┤LED2      │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │      3       ├──────┤LED3      │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │      4       ├──────┤LED4      │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │      5       ├──────┤LED5      │
   └──────────────┘      │          │
   ┌──────────────┐      │          │
   │    TRUE      ├──────┤LED6      │
   └──────────────┘      │          │   ┌──────────────┐
   ┌──────────────┐      │          │   │              │
   │    TRUE      ├──────┤LED7    Q ├───┤    OUT1      │
   └──────────────┘      └──────────┘   └──────────────┘
```
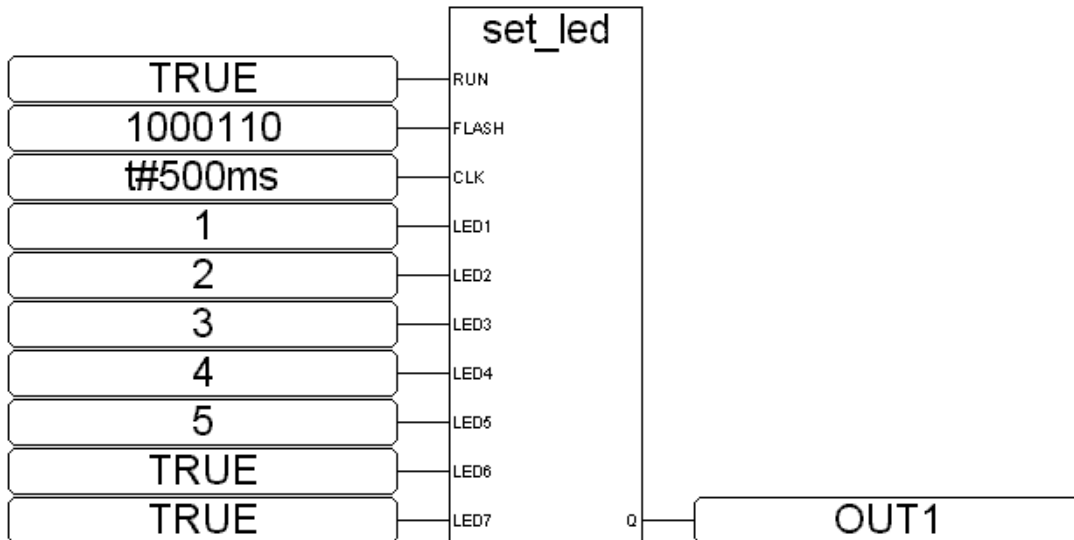
ST Equivalence:
    TMP := SET_LED(TRUE,1000110,t#500ms,1,2,3,4,5,TRUE,TRUE);
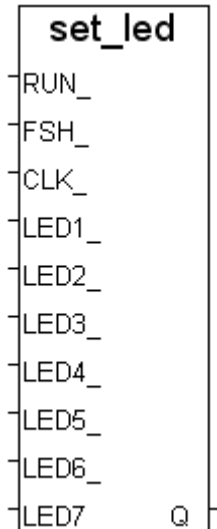   (* TMP should be declared as Boolean variable *)

SETRTS

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function

    Set the RTS of COM PORT, available for COM number: 3 ~ 5

```
      setrts
 ─│PORT_      │
 ─│ACTIV     Q│─
```
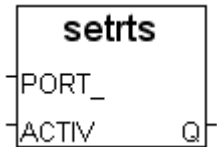
Input Parameters:

    PORT_   Integer   The COM to set RTS. 3:COM3, 4:COM4, 5:COM5
    ACTIVE_   Boolean   TRUE: set RTS active; FALSE: set RTS inactive

Return :

    Q_   Boolean   TRUE: OK, FALSE: fail

SMS_GET

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  SMS_get
 ─│Ref_    N│─
```
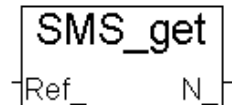
Description : C_Function

    Get Message date and time from controller's date & time (Refer to Chapter 17)

Input Parameters :

    REF_   Integer   What to get? 1 ~ 7
                          1: year,   (N_ = 2000 ~ 2099)
                          2: month,   (N_ = 1 ~ 12)
                          3: day,   (N_ = 1 ~ 31)
                          4: week day, (N_ = 1 ~ 7, 7 means Sunday.)
                          5: time,   (N_ = 0 ~ 23)
                          6: minute,   (N_ = 0 ~ 59)
                          7: second,   (N_ = 0 ~ 59)
                          Others: return N_= -1 (means error)

Return :

    N_   Integer   Return associated data with Ref_.
                          If return -1, it may be "No message" or Ref_ out of range of 1 ~ 7.

SMS_GETS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  SMS_gets
 ─│Ref_   M_│─
```
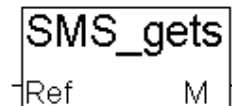
Description : C_Function

    Get Message data and other information (Refer to Chapter 17)

Input Parameters :

    REF_   Integer   What to get?   1 ~ 3
                          1: message data
                          2: get phone No. of sender
                          3: get date & time in string format
                          Others: return M_= 'error'

Return :
   M_   Message   Return associated data with Ref_. If return 'error', it may be "No
                           message" or Ref_ out of range of 1 ~ 3.
Note:
   1. SMS_gets & SMS_get can be called to get Message and the related information.
   2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No
      message". So if the other information are need, please call SMS_get(1~7) & SMS_gets(2)
      & SMS_gets(3) before calling SMS_gets(1).

Example: demo_43, demo_43a

SMS_SEND

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 SMS_send
┤No_
┤M_      Q ┝
```
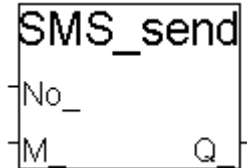
Description : C_Function
   Trigger the controller to send a new Message (Refer to Chapter 17)

Input Parameters :
   No_   Message   To which phone No. Ex. '+886920119135', max length is 31 digits
   M_   Message   The Message content to send

Return :
   Q_   Boolean   True: ok; False: wrong phone number or fail to send.

SMS_STS

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 SMS_sts
      N ┝
```
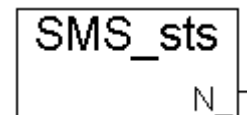
Description : C_Function
   Get Message Sending status. (Refer to Chapter 17)

Return :
   N_   Integer   The sending status code
                         0 : Waiting for a new sending request
                         1 : Busy. (One message is processing now)
                         21: The message is sent successfully

                         -1 : SMS system is not available (Check GSM Modem & SIM card)
                         -2 : Timeout, no response. (It May be no such a phone No.)
Note:
1. Please call SMS_sts to get the "Message Sending status" before calling SMS_send.
   SMS_send only works when status is not "1:busy".

2. A successfully SMS_send request will reset the "Message sending status" to "1:busy", and
   after that, by the time, it will set to the associate status. For ex. "21:successfully sent".
Example: demo_43, demo_43a

SMS_TEST

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
SMS_test

          Q_
```
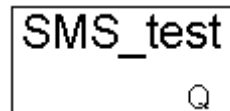
Description : C_Function
Test if message coming or not (Please refer to Chapter 17)

Return :
Q_   Boolean   TRUE: A message is coming,   FALSE: no message

Note:

1. SMS_gets & SMS_get can be called to get message

2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No message". So if the message's related information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example : demo_43 , demo_43a

STR_REAL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
str_real
STR    REAL
```

Description : C_Function
Convert a string to a Real value

Input Parameters :
STR_   Message   The String to convert, Ex: '-0.2345' , ' +2.13E10' , ' 15.2345E-2'
Return :
REAL_   Real   The Real value after conversion
If it returns "1.23E-20", it means the setting for STR_ is wrong.
Ex: If STR_= ' 123.AB' or '23-45.17' or '1.2.345',
REAL_ will return 1.23E-20.
Note:
"REAL_STR" & "REA_STR2" could be called to convert Real value to String.

Example: Please refer to Chapter 16 & demo_38, demo_39

SYSDAT_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
sysdat_r
       YY_
       MM_
       DD_
       WW_
```

Description : C_Function Block
     Read system year, month, day and date
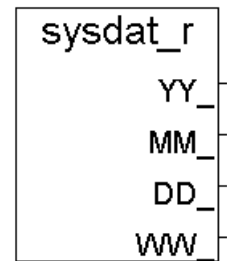
Return :
YY_	Integer	Year, e.g. 2002, 2003, 2010
MM_	Integer	Month, 1 ~ 12
DD_	Integer	Date in month, 1 ~ 31
WW_	Integer	Day of week, 1 ~ 7, 7 indicate Sunday, etc.

Example: Please refer to demo_03.
    Y1, M1, D1 & W1 are declared as Integer variables.

```
              SYSDAT_R
          en        eno                    < >

                   YY_ ─Y1

                   MM_ ─M1

                   DD_ ─D1

                   WW_ ─W1
```
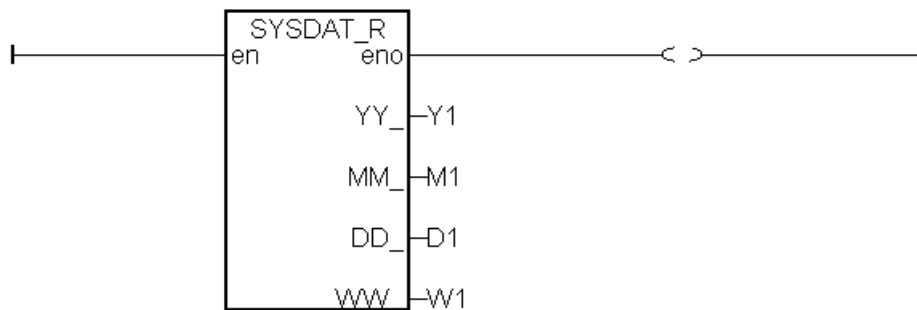
ST Equivalence:
  DAT_R1( );   (* call DAT_R1 *)
  Y1 := DAT_R1.YY_ ;   (* get year *)
  M1 := DAT_R1.MM_ ;   (* get month *)
  D1 := DAT_R1.DD_ ;   (* get date *)
  W1 := DAT_R1.WW_ ;   (* get day of week *)
  (*DAT_R1 is declared as FB instance of SYSDAT_R *)

SYSDAT_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  sysdat_w
┤IN_
┤YY_
┤MM_
┤DD_    Q_├
```

Description : C_Function Block

Set new system year, month and day
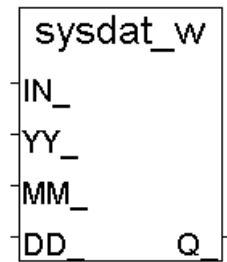
Input Parameters:

IN_	Boolean	Set new system date when rising from FALSE to TRUE
YY_	Integer	Year, e.g. 2002, 2003, 2010
MM_	Integer	Month, 1 ~ 12
DD_	Integer	Date, 1 ~ 31

Return :

Q_	Boolean	Ok returns TRUE

Example: Please refer to demo_03.

SW1 is declared as Boolean variable. Y1, M1 & D1 are declared as Integer variable.

```
     SW1              SYSDAT_W
├─────┤ ├──────────┤IN_      Q_├──────────< >──────────┤
                  Y1─┤YY_
                  M1─┤MM_
                  D1─┤DD_
```
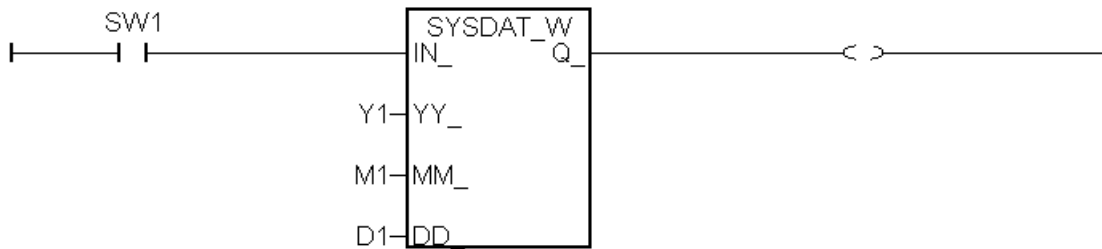
ST Equivalence:
DAT_W1(SW1, Y1, M1, D1);   (* call DAT_W1 *)
OUT1 := DAT_W1.Q_ ;   (* get return value Q_ *)
(* DAT_W1 is declared as FB instance of SYSDAT_W*)
(* OUT1 is declared as Boolean variable*)

SYSTIM_R

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
systim_r
         HH_
         MM_
         SS_
```

Description : C_Function Block
Read system hour, minute and second.

Return :

HH_	Integer	Hour, 0 ~ 23
MM_	Integer	Minute, 0 ~ 59
SS_	Integer	Second, 0 ~ 59

Example: Please refer to demo_03 & demo_15b.

H1, M1 & S1 are declared as Integer variables.

```
        SYSTIM_R
    en        eno                    < >
         HH_ H1
         MM_ M1
         SS_ S1
```
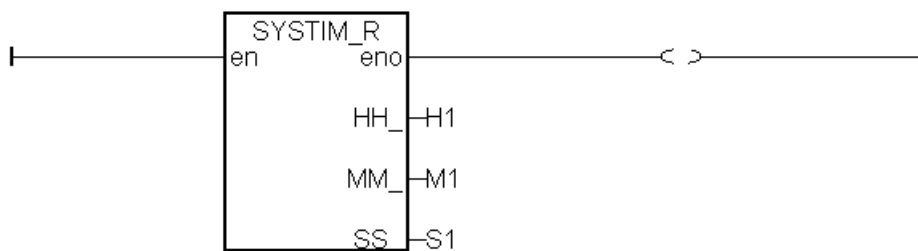
ST Equivalence:
```
(* TIM_R1 is declared as FB instance SYSTIM_R*)
TIM_R1( );                    (* call TIM_R1 *)
H1 := TIM_R1.HH_ ;            (* get hour *)
M1 := TIM_R1.MM_ ;            (* get minute *)
S1 := TIM_R1.SS_ ;           (* get second *)
```

SYSTIM_W

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
  ┌──────────────┐
  │   systim_w   │
 ─┤IN_           │
 ─┤HH_           │
 ─┤MM_           │
 ─┤SS_         Q ├─
  └──────────────┘
```

Description : C_Function Block
Set new system hour, minute and second.
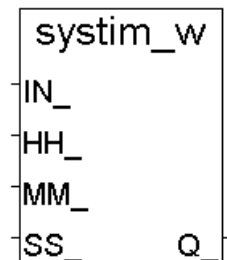
Input Parameters :
IN_	Boolean	Set new system time when rising from FALSE to TRUE
HH_	Integer	Hour, 0 ~ 23
MM_	Integer	Minute, 0 ~ 59
SS_	Integer	Second, 0 ~ 59

Return :
Q_	Boolean	If OK returns TRUE

Example: Please refer to demo_03.

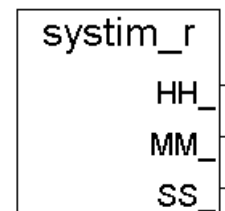SW1 is declared as Boolean variable. H1, M1 & S1 are declared as Integer variable.

```
    SW1              ┌──────────────┐
 ───┤ ├──────────────┤   SYSTIM_W   │
                     │IN_        Q_ ├──────────────< >───────────
                     │              │
                 H1──┤HH_           │
                     │              │
                 M1──┤MM_           │
                     │              │
                 S1──┤SS_           │
                     └──────────────┘
```
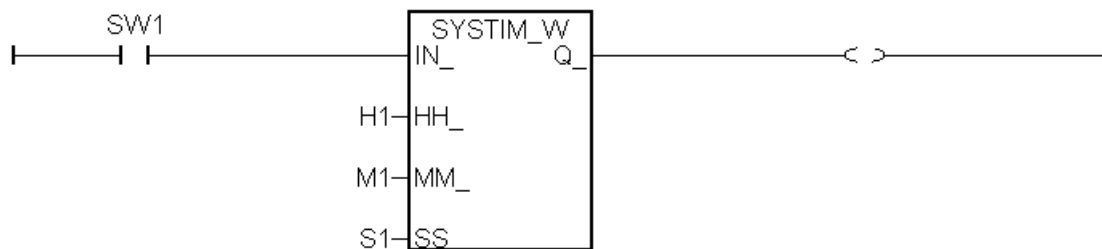
ST Equivalence:
```
TIM_W1( Sw1,2000,7,5);          (* call TIM_W1 *)
OUT1 := TIM_W1.Q_ ;             (* get the returned value Q_ *)
(* TIM_W1 is declared as FB instance of SYSTIM_W*)
(* OUT1 is declared as Boolean variable*)
```

TCP_RECV
□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG/7186EG □ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

tcp_recv
ID   Msg

 TCP Client receive message from remote PC or device's TCP/IP server (via Ethernet) (Please refer to Section 19.3)

Input Parameters :

| **ID_** | Integer | The related "Tcp_Clie" connecting number, 1 ~ 4. The related "ip address" and "port No." is defined in IO connection: "Tcp_clie". |

Return :

| **MSG_** | Message | The received Message. If Msg_ = '' (empty Message), it means no Message coming. |

Note :

1. Please connect "tcp_clie" (TCP_Client) in the IO connection windows before using "tcp_recv" & "tcp_send"
2. The receiving buffer size is 4096 bytes. If the receiving buffer is full, the oldest message will be dropped to release space for receiving new coming data.

Example : Wpdmo_33 & Wpdmo_34 (WP-8xx7 CD_ROM:\napdos\isagraf\wp-8xx7\demo\)

TCP_SEND

tcp_send
ID_
Msg   Q

□ I-8x17/iP-8x17 □ I-8x37/iP-8x47 □ I-7188EG □ I-7188XG
■ ISaGRAF WinCE PAC (ver 3.30)
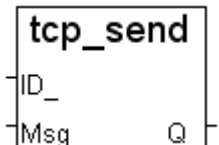
Description : C_Function

 TCP Client send Message to remote PCs or device's TCP/IP server (via Ethernet) (Please refer to Section 19.3)

Input Parameters :

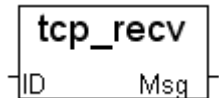| **ID_** | Integer | The related "Tcp_Clie" connecting number, 1 ~ 4. The related "ip address" and "port No." is defined in IO connection: "Tcp_clie". |
| **Msg_** | Message | The message to send |

Return :

| **Q_** | Boolean | True: send OK; False: parameter error (Ex, set ID_ to 8) or the setting of the related "Tcp_Clie" connection in IO connection window is error. |

Note :

1. Please connect "tcp_clie" (TCP_Client) in the IO connection windows before using "tcp_recv" & "tcp_send"
2. The sending buffer for Wincon is 4096 bytes. That means max. 4096 bytes in one PLC scan can be sent to remote IP. If sending buffer is full, the oldest message will be dropped to release space for new "tcp_send( )" request.
3. The controller driver will send only one message out each PLC scan when there is message in the sending buffer. For example, if there are 100 messages in the sending buffer, the controller will send over these 100 messages in 100 PLC scan cycle. However if set "Send_Time_Gap" of "Tcp_clie" to a larger value, for example 100 (ms), each message will be sent out one by one every 100 ms later.

Example: Wpdmo_33 & Wpdmo_34 (WP-8xx7 CD_ROM:\napdos\isagraf\wp-8xx7\demo\)

TIME_STR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
┌─────────────┐
│  TIME_STR   │
┤YEAR_        │
┤MON_         │
┤MDAT_        │
┤WDAT_        │
┤HOUR_        │
┤MIN_         │
┤SEC_      M_ ├
└─────────────┘
```

Description : C_Function

Convert Date & Time to String format

Input Parameters :

YEAR_	Integer	Year, 2000 ~
MON_	Integer	Month, 1 ~ 12
MDAY_	Integer	Date, 1 ~ 31
WDAY_	Integer	Day of week, 1 ~ 7 (Monday ~ Sunday)
HOUR_	Integer	Hour, 0 ~ 23
MIN_	Integer	Minute, 0 ~ 59
SEC_	Integer	Second, 0 ~ 59
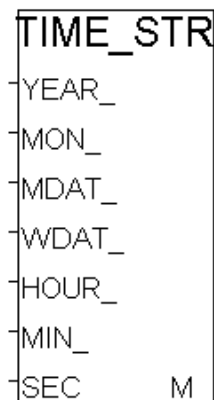
Return :

M_   Message Length is 24 Characters. Ex: 'Feb/18/2003,13:25:45,Tue'

Note:

1. Please use sysdat_r & systim_r to get date & time of the controller.
2. If giving wrong input parameters, it will returned M_ = '' (empty String). Ex: MON_=14.

TMR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : Standard Function

Convert any Analog variable to a Timer one

```
┌──────────┐
│   Tmr    │
┤IN      Q ├
└──────────┘
```
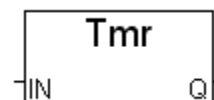
Input Parameters :

IN_   Int-Real   Any non-timer value

Only the Integer part of IN if it is a Real. Unit: millisecond.

Return :

Q_   Timer   The Timer value of IN

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

```
┌──────────┐   ┌──────┐   ┌──────────┐
│   1256   ├───┤IN Tmr Q├──┤   ares   │
└──────────┘   └──────┘   └──────────┘

┌──────────┐   ┌──────┐   ┌──────────┐
│  1256.3  ├───┤IN Tmr Q├──┤   rres   │
└──────────┘   └──────┘   └──────────┘
```
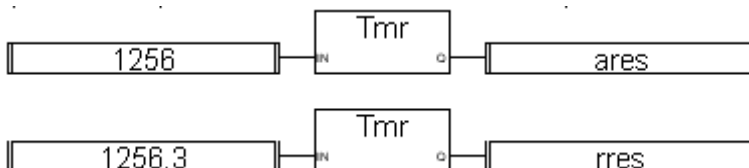
(* ST equivalence: *)
      ares := TMR (1256);   (* ares := t#1s256ms *)
      res := TMR (1256.3);   (*rres := t#1s256ms *)

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

TO_A4_20

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
 to_a4_20
─EngVa
─Range
─HiVal
─LoVal      Q ─
```

Description : C_Function

Convert User's Engineering Value ("Real" format) to Variable's AO value (Analog output signal is 4 to 20mA, Integer format)
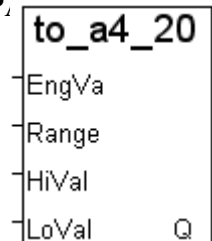
Input Parameters :

EngVal_	Real	The Engineering value to be converted.
Range_	Integer	Range setting of the Analog output board or module.

       16#0 :   0 ~ 20 mA
       16#1 :   4 ~ 20 mA
       16#30 :   0 ~ 20 mA
       16#31 :   4 ~ 20 mA

HiVal_	Real	User's related High Eng. value when analog output signal is 20 mA
LoVal_	Real	User's related Low Eng. value when analog output signal is 4 mA

EX: Convert 0 - 100 psi to become I-8024's AO value, please set HiVal_ = 100.0 , LoVal_ = 0.0 & Range_=16#30 (depends on the range setting of the related IO board).

Return :

Q_	Integer	The AO value after conversion (value is usually in 0 to +32767, depends on which Range setting of the IO board). If given incorrect Range_ or (HiVal_ = LoVal_), returns -1.

Example :

1. Convert (0 - 100) psi to become (6554 - 32767) (if the range setting of the analog output board is 16#30 : 0 - 20mA)
2. Convert (0 - 100) psi to become (0 - 32767) (if the range setting of the analog output board is 16#31 : 4 - 20mA)
3. Scale (0 to 3000 rpm) to I-8024's current output with range setting of 30: (0 to 20 mA). 0 rpm should output as 4 mA , 3000 rpm outputs as 20 mA.

```
              TO_A4_20
          ─┤en         eno├──────────────< >──────────
Rpm_val──┤EngVal_     Q_├─I_8024_2_01
  16#30──┤Range_
 3000.0──┤HiVal_
    0.0──┤LoVal_
```

I-8024_2_01 is declared as integer output, Rpm_val is declared as a Real internal.

Note: 1. The related range setting of Analog output board or module should be set as mA range.
     Ex: (-20 , +20mA) , (0 , 20mA) , (4 , 20mA) … mA
    2. When using A4_20_to, To_A4_20, To_V0_10, V0_10_to functions, user needs to upgrade the driver to the version listed below or latter version: i-7188EG: v2.16, i-7188XG: v2.14, i-8xx7:v3.18, so that the program can run well. (The older driver may cause the program cease after running a period of time.).

TO_V0_10

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
         to_v0_10
   ─┤EngVa
   ─┤Range
   ─┤HiVal
   ─┤LoVal      Q├─
```

Description : C_Function
> Convert User's Engineering Value ("Real" format) to Variable's AO value (Analog output signal is 0 to 10V, Integer format)
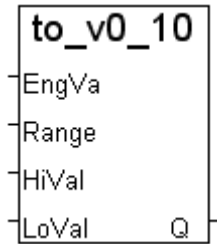
Input Parameters :
EngVal_	Real	The Engineering value to be converted.
Range_	Integer	Range setting of the Analog output board or module.

                    16#2 :   0 ~ 10 V
                    16#32 :   0 ~ 10 V
                    16#33 : -10 ~ 10 V
                    16#34 :   0 ~ 5 V
                    16#35 :   -5 ~ +5 V

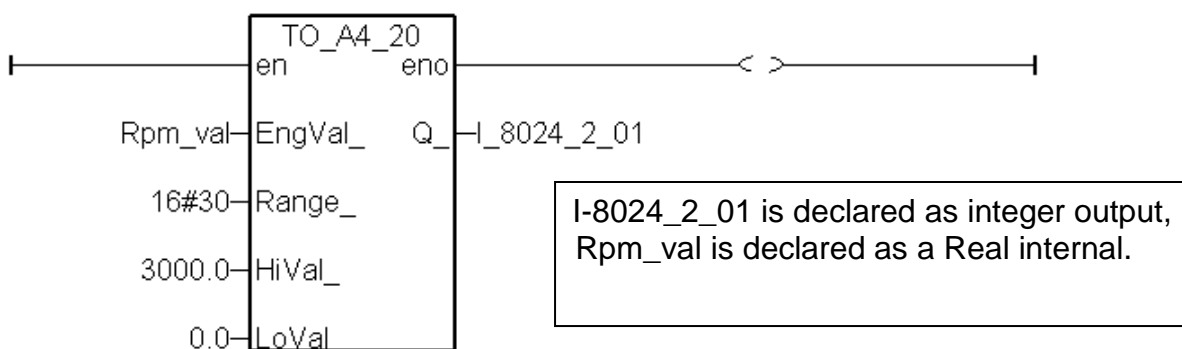|		
HiVal_	Real	User's related High Eng. value when Analog output signal is 10 V
LoVal_	Real	User's related Low Eng. value when Analog output signal is 0 V

> Ex: Convert 0 - 100 psi to become I-8024's AO value, please set HiVal_ = 100.0 , LoVal_ = 0.0 and Range_=16#33 (depends on the range setting of the related IO board).

Return :
Q_	Integer	The AO value after conversion

> Value is usually in (0 ~ +32767), depends on which Range setting of the IO board.
> If given incorrect Range_ or (HiVal_ = LoVal_) , returns -1.

Example :
1. Convert (0 - 100) psi to become (0 - 32767) (if the range setting of the analog output board is 16#32 : 0 ~ 10V)
2. Convert (0 - 100) psi to become (0 - 32767) (if the range setting of the analog output board is 16#33 : -10 ~ +10V)

Note: The related range setting of Analog output board or module should be set as Voltage range. for ex, (0 , 10 V) , (-10 , 10 V)

Note: 1. The related range setting of Analog output board or module should be set as Voltage range. E: (0 , 10 V) , (-10 , 10 V) etc.
      2. When using A4_20_to, To_A4_20, To_V0_10, V0_10_to functions, user needs to upgrade the driver to the version listed below or latter version: i-7188EG: v2.16, i-7188XG: v2.14, i-8xx7:v3.18, so that the program can run well. (The older driver may cause the program cease after running a period of time.).

TOF

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
        TOF
 ─┤IN      Q├─
 ─┤PT     ET├─
```

Description : Standard Function

OFF-Delay control
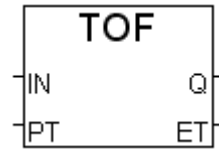
Input Parameters :

IN   Boolean   To control the input of OFF-delay.
If it's on the Rising edge, output Q to ON (set to TRUE).
If it's on the Falling edge, start the ET OFF-delay timer.

PT   Timer   Setting of the delay time,
When ET Timer count to the setting time of PT, Q will be OFF (set to False).

Return :

Q   Boolean   Output

ET   Timer   The elapsed time since TOF was active.

Timing Diagram: (OFF-delay)

Count to PT, then set Q to FALSE.

If IN is falling, begin timer ticking.
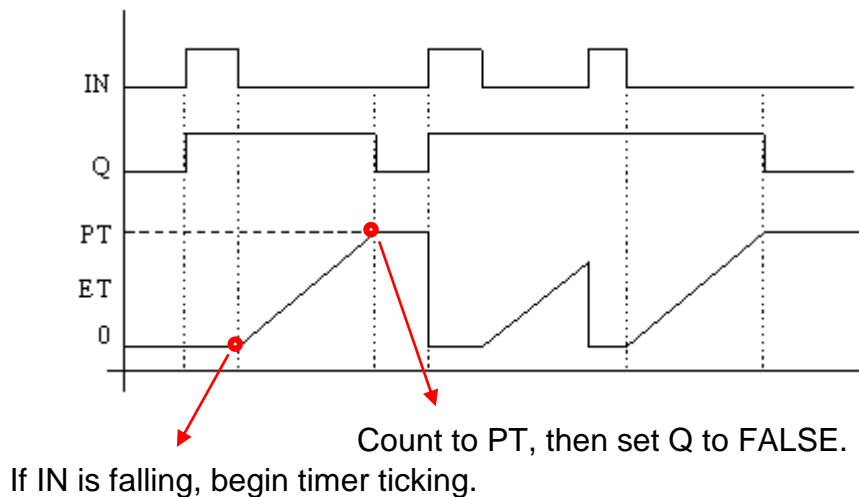
TON

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : Standard Function

ON-Delay control

```
      TON
┤IN        Q├
┤PT        ET├
```
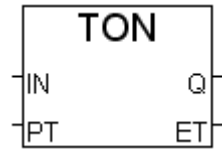
Input Parameters :

IN	Boolean	To control the input of ON-delay.
		If it's on the Rising edge, start the ET ON-delay timer.
		If it's on the Falling edge, stop timer and set Q to FALSE.
PT	Timer	Set the delay time

Return :

Q	Boolean	Output
ET	Timer	The elapsed time since TON was active

Timing Diagram: (ON-delay)
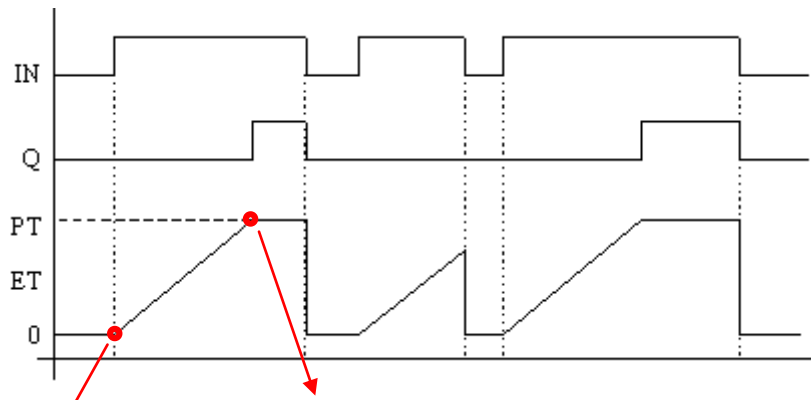
Count to PT, then set Q to TRUE.

If IN is Rising, begin timer ticking.

TP

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

```
      TP
─┤IN        Q├─
─┤PT       ET├─
```

Description : Standard Function

Set output to ON for an interval time (Pulse timer).

Input Parameters :

IN	Boolean	To control the input of TP
		If IN is rising, output Q to ON and start ET timer.
		If IN is falling, stop the timer and reset the timer.
		Note: The rising / falling of IN is work only when Q is OFF,
		and the change of IN is not effect when the timer is active.
PT	Timer	Set the time interval (How long that Q will ON).

Return :

Q	Boolean	Output
ET	Timer	The elapsed time since TP was active

Timing Diagram: (Pulse Timer)

The change of IN is not effect when the timer is active.
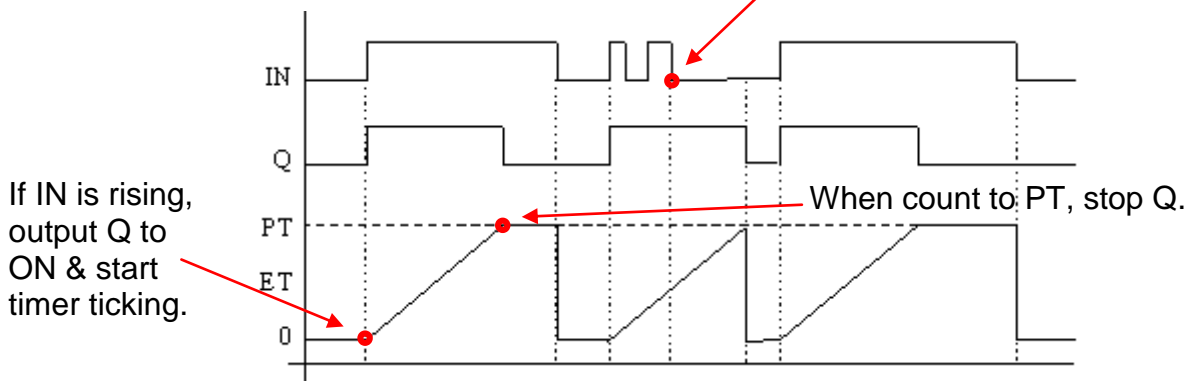
If IN is rising, output Q to ON & start timer ticking.

When count to PT, stop Q.

TWIN_LED

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
□ ISaGRAF WinCE PAC

```
   twin_led
─┤RUN_
─┤V1_
─┤V2_
─┤VAL_
─┤CLK_      Q├─
```

Description : C_Function

Show 2 screen values to the S-MMI.
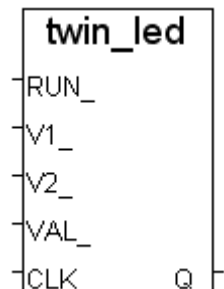
Input Parameters :

RUN_	Boolean	To show only in RUN_ is TRUE.
V1_	Integer	The value displayed on the left 2 digits of the 1st screen, 0 ~ 99.
V2_	Integer	The value displayed on the right 2 digits of the 1st screen, 0 ~ 99
VAL_	Integer	The value displayed on the 2nd screen, -99999 ~ 99999
CLK_	Timer	The blinking period of these 2 screens

Return :

Q_	Boolean	Always TRUE

Example: Please refer to demo_10.

UDP_RECV

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
udp_recv

            Msg
```
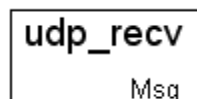
Description : C_Function

Receive Message from remote UDP/IP connection (via Ethernet) (Refer to Section 19.2)

Return :

Msg_   Message   The received Message. If Msg_ = '' (Empty message), it indicates there is no Message coming

Note :

1. The receiving buffer size of UDP for W-8xx7 is 8192 bytes including the 1 byte ending symbol of each String. The receiving buffer size for I-7188EG & I-8x37 is 2048 bytes.
2. If there coming too many Strings at the same time and the receiving buffer cannot handle in time, the new coming String will replace the earliest coming String.

Example :

Wpdmo19 & Wpdmo19a (WP-8xx7 CD_ROM:\napdos\isagraf\wp-8xx7\demo\)
www.icpdas.com > FAQ > Software > ISaGRAF > 065

UDP_SEND

□ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG □ I-7188XG
■ ISaGRAF WinCE PAC

```
udp_send

 ID_
 Msg         Q
```
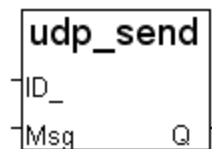
Description : C_Function

Send Message to remote UDP/IP connection (via Ethernet) (Refer to Section 19.2)

Input Parameters :

ID_   Integer   The related connection to send, 1~4. The related "ip address" and "port No." is defined in "udp_ip" of I/O connection window.
Msg_   Message   The Message to send.

Return :

Q_   Boolean   True: OK;
False: the sending buffer is full or parameter error(Ex: ID_= 8).

Note :

1. The sending buffer size of UDP for W-8xx7 is 2048 bytes including the 1 byte ending symbol. It means W-8xx7 can send out max to 2048 bytes in each ISaGRAF PLC Scan.
2. Do not send a lot of messages frequently because of the "Send_Time_Gap" parameter in the I/O connection "udp_ip". When there are too many messages, the sending buffer will full, and the udp_send( ) will return FALSE that means the buffer is full and cannot put in more Message . The controller driver sends only one message out each PLC scan relating to each UDP connection.

Example :

Wpdmo19 & Wpdmo19a (WP-8xx7 CD_ROM:\napdos\isagraf\wp-8xx7\demo\)
www.icpdas.com > FAQ > Software > ISaGRAF > 065

V_BCD

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Convert value to BCD value.

```
 v_bcd
IN        Q
```

Input Parameters :

IN_   Integer   Decimal value to be converted, valid range: 0 ~ 99999999

Return :

Q_   Integer   BCD value   ex: 12345 → 16#12345
                                            16   → 22 (16#16)

V0_10_TO

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG
■ ISaGRAF WinCE PAC

```
 v0_10_to
Analn
Range
HiVal
LoVal     Q
```
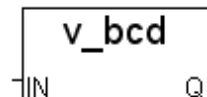
Description : C_Function

Convert Analog Input from 0-10 V to User's Engineering Value ("Real" format), Ex: Convert the Analog input of I-8017H to 0-100 psi or 0-3000 rpm.
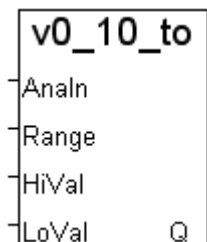
Input Parameters :

Analn_   Integer   The Integer variable related to the Analog input board or module. Valid range: -32768 ~ +32767, depends on the range setting of the IO module.

Range_   Integer   The range setting of Analog Input IO card or module.
16#0 :   -15mV ~ +15 mV
16#1 :   -50mV ~ +50 mV
16#2 :   -100mV ~ +100 mV
16#3 :   -500mV ~ +500 mV
16#4 :   -1 ~ +1 V
16#5 :   -2.5 ~ +2.5 V
16#7 :   -1.25 ~ +1.25 V
16#8 :   -10 ~ +10 V
16#9 :   -5 ~ +5 V
16#A :   -1 ~ +1 V
16#B :   -500mV ~ +500 mV
16#C :   -150mV ~ +150 mV

HiVal_   Real   User's related High Eng. value when Analog input signal is 10 V
LoVal_   Real   User's related Low Eng. value when Analog input signal is 0 V

Ex: Convert I-8017H's Input signal from 0 - 10 V to 0 - 100 psi, please set HiVal_ = 100.0 , LoVal_ = 0.0 & Range_ = 16#5 or 16#7 or 16#8 or 16#9 (depends on the range setting of the related IO board).

Return :

Q_   Real   The User Engineering value after conversion.
If given incorrect Range_ , returns 1.23E-20.

Note: 1. The related range setting of Analog Input card or module must be Voltage.
Ex: (-10 , +10V) , (-5 , +5V) , (-1 , 1 V), etc Voltage range.
2. When using A4_20_to, To_A4_20, To_V0_10, V0_10_to functions, user needs to upgrade the driver to the version listed below or latter version: i-7188EG: v2.16, i-7188XG: v2.14, i-8xx7:v3.18, so that the program can run well. (The older driver may cause the program cease after running a period of time.).

VAL_HEX
■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Convert an Integer to a fixed-length hex-message.

```
 val_hex
┤VAL_
┤DIGIT    HEX├
```

Input Parameters :

VAL_	Integer	The Integer to be converted
DIGIT_	Integer	Number of digits for HEX_ message. Valid value: 1 ~ 8. Given others will do nothing and force HEX_ to '' (Empty Message).

Return :

HEX_	Message	The hex-message after conversion.

Example:

val_hex(100,3)	--->	'064'
val_hex(192,4)	--->	'00C0'
val_hex(4589,2)	--->	'ED' ('11ED', DIGIT_ is 2, force '11' trucated)
val_hex(4589,9)	--->	' ' (DIGIT_ > 8, output ' ')
val_hex(-2,8)	--->	'FFFFFFFE'

VAL10LED

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function
      Display a decimal Integer on S-MMI

```
      val10led
    ┤RUN_
    ┤FSH_
    ┤CLK_
    ┤VA_I      Q├
```

Input Parameters :

RUN_	Boolean	If TRUE, display it
FLASH_	Boolean	If TRUE, blink it
CLK_	Timer	The blinking period
VAL_I_	Integer	The Integer to display, -9999 ~ +99999
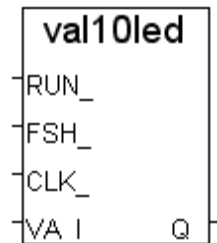
Return :

Q_	Boolean	Return TRUE always

Example: Please refer to demo_07 & demo_11b.

```
                              val10led
   ┌──────────────┐        ┌──────────────┐
   │    TRUE       │────────┤RUN           │
   ├──────────────┤        │              │
   │    TRUE       │────────┤FLASH         │       ┌──────────────┐
   ├──────────────┤        │              │     ┌──┤    OUT1       │
   │   t#500ms     │────────┤CLK           │     │  └──────────────┘
   ├──────────────┤        │          Q├──┘
   │    9875       │────────┤VAL_I         │
   └──────────────┘        └──────────────┘
```
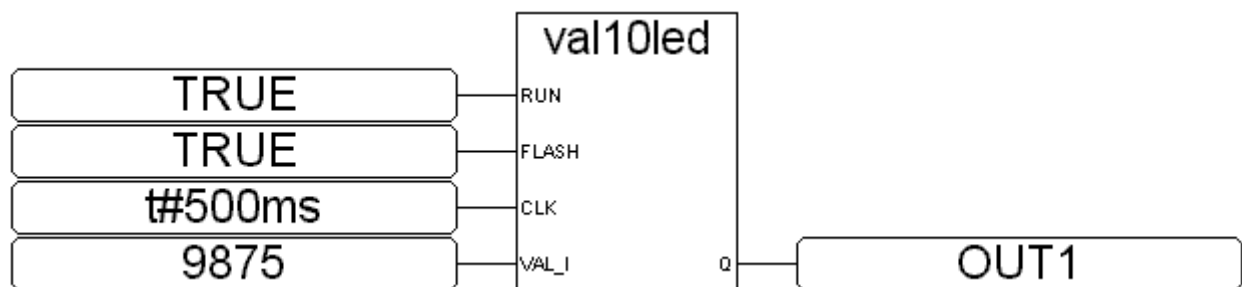
ST Equivalence:
    TMP := VAL10LED(TRUE,TRUE,t#500ms,9875);
    (* TMP is declared as Boolean variable *)

VAL16LED

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG □ ISaGRAF WinCE PAC

Description : C_Function

Display a hexadecimal Integer on S-MMI

Input Parameters :

RUN_	Boolean	If TRUE, display it
FLASH_	Boolean	If TRUE, blink it
CLK_	Timer	The blinking period
VAL_I_	Integer	The value to display, 16#0 ~ 16#FFFFF
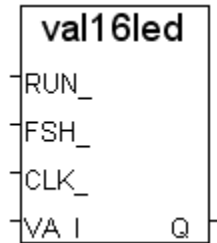
Return :

Q_	Boolean	Return TRUE always

Example:
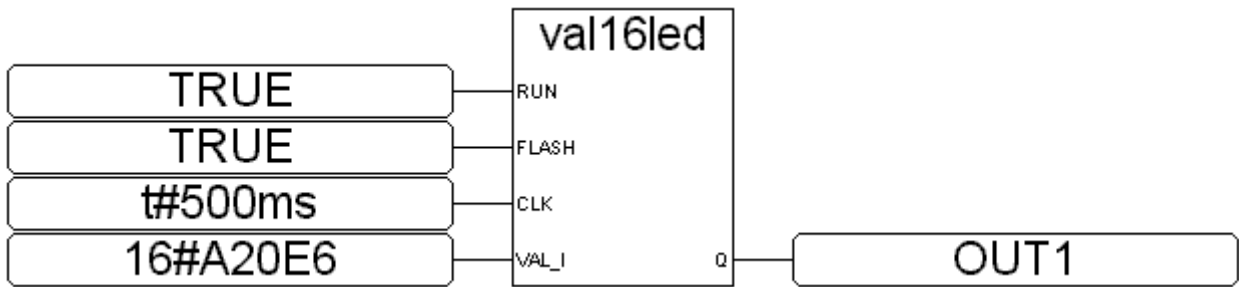
ST Equivalence:

    TMP := VAL10LED(TRUE,FALSE,t#500ms,16#A20E6);
    (*TMP is declared as Boolean variable*)

W_MB_ADR

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Write Boolean or Integer variable by using Modbus network address
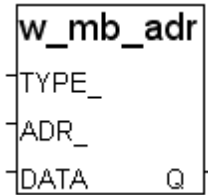
```
w_mb_adr
TYPE_
ADR_
DATA    Q
```

Input Parameters :

TYPE_	Integer	0: Boolean variable, 1: Integer variable
ADR_	Integer	The Modbus address to write to,
		Valid range for Wincon: 1~8191; others: 1~4095.
DATA_	Integer	The Integer to write (For Boolean, 0:False, 1:True.)

Return :

Q_	Boolean	TRUE : ok; FALSE: fail.

Note :

1. Please use W_MB_REL function to write "REAL" variable.
2. If no variable defined with the given Modbus address, no write action.
3. If TYPE_ is given as Integer however the related variable is "Real" typed, the related 32-bit is written. It is better to use "W_MB_REL" to write Real variable.
4. If TYPE_ is given as Integer however the related variable is "Boolean" typed, no write action.
5. If TYPE_ is given as Boolean however the related variable is not "Boolean" typed, no write action.
6. If Long Integer value (32-bit integer) is to be delivered to HMI via Modbus protocol, it should occupy 2 Modbus address number. Please refer to Section 4.2 of "User's Manual Of The ISaGRAF PAC".

W_MB_REL

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

**Description : C_Function**

Write REAL variable by using its network address number
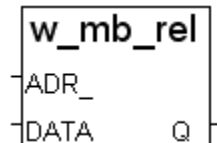
```
w_mb_rel
ADR_
DATA    Q
```

Input Parameters :

ADR_	Integer	The Modbus address to write to,
		Valid range for Wincon: 1~8191, others: 1~4095
DATA_	Real	The REAL to write

Return :

Q_	Boolean	TRUE : ok, FALSE : fail

Note :

1. Please check the related variable is "Real" type. If it is "Integer", please use "W_MB_ADR" function.
2. If the given TYPE_ is not "Analog" (Real or Integer), no write action.
3. If no variable defined with the given Modbus address, no write action.
4. If REAL value is to be delivered to HMI via Modbus protocol, it should occupy 2 Modbus address number. Please refer to Section 4.2 of "User's Manual Of The ISaGRAF PAC".

WD_BIT

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function Block

Convert a Word value (signed 16-bit) to 16 Boolean values

Input Parameters :

VAL_   Integer   The Word to be converted.
                             (only the lowest 16-bit are used)

Return :

ENO_   Boolean   Reserved; no usage.
B1_ ~ B16_   Boolean   The 16 Boolean values after conversion
                             Ex: If VAL_ is 4, B3_ will be TRUE
                                 and others will be FALSE.

```
wd_bit
        ENO
        B1_
        B2_
        B3_
        B4_
        B5_
        B6_
        B7_
        B8_
        B9_
        B10_
        B11_
        B12_
        B13_
        B14_
        B15_
VAL     B16_
```
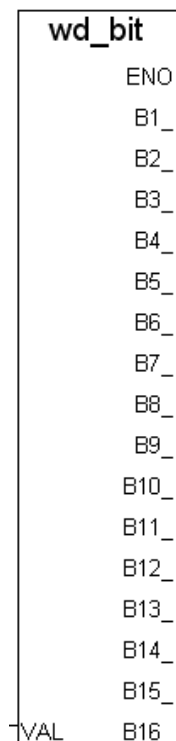
WD_LONG

■ I-8x17/iP-8x17 ■ I-8x37/iP-8x47 ■ I-7188EG/7186EG ■ I-7188XG ■ ISaGRAF WinCE PAC

Description : C_Function

Convert two Words (signed 16-bit) to one Long Integer (signed 32-bit)

```
wd_long
Lo_
Hi      Long
```

Input Parameters :

Lo_   Integer   Low Word to be converted (only the lowest 16-bit is used)
Hi_   Integer   High Word to be converted (only the lowest 16-bit is used)

Return :

Long_   Integer   The 32-bit Integer composed by Lo_ and Hi_ word
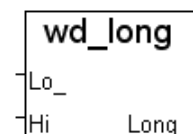
Example:

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

Appendix B : Setting The IP, Mask & Gateway In The I-8x37/iP-8x47, I-7188EG & μPAC-7186EG

This document describe the proper way to set the IP address, address mask and gateway address of the I-8x37/iP-8x47, I-7188EG & μPAC-7186EG controllers.

For the setting of WinCon-8xx7/8xx6, please refer to "Getting Started Manual" delivered with Wincon or at http://www.icpdas.com/products/PAC/i-8000/getting_started_manual.htm .

****
EACH I-8x37/iP-8x47, I-7188EG & μPAC-7186EG USES TCP/IP PORT NO. 502 TO TALK TO THE HMI AND ISAGRAF WORKBENCH. A MAXIMUM NUMBER OF 4 PCS CAN TALK TO THE I-8x37/iP-8x47, I-7188EG & μPAC-7186EG THROUGH MODBUS TCP/IP PROTOCOL.
****

The setting steps for I-7188EG & μPAC-7186EG controller:

1. Create a file folder named "7188" in your hard drive. For example: "c:\7188".

2. Copy \Napdos\ISaGRAF\7188EG\Driver\7188xw.exe, 7188xw.ini, from the CD_ROM into your "7188" folder.

3. Run "\7188\7188xw.exe"   (For Windows NT, Windows 2000 & Windows XP)

4. Link from COM1/COM2 of your PC to COM1 of the I-7188EG/μPAC-7186EG by a RS232 cable (CA0910).
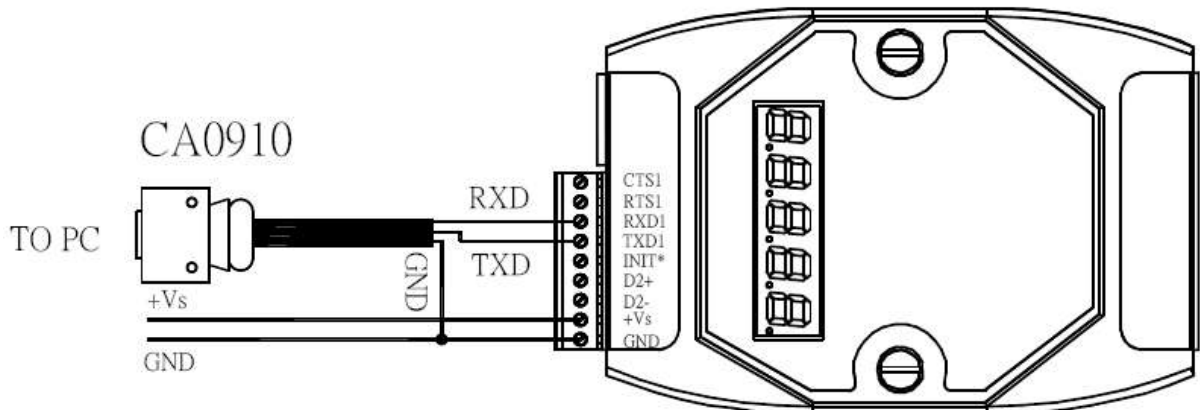
If your computer has no COM1/COM2 or you use other COM (like COM5) to link the I-7188, you can change the "C number" in the first line of "7188xw.ini" file.

EX: Using computer's COM5 to link to I-7188

```
C1 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```

➡

```
C5 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```

5. Power off the I-7188EG/μPAC-7186EG, connect pin "INIT*" to "GND", then power it up.

6. If the connection is Ok, "i7188E>" messages will appear on the 7188xw screen.

7. Type "ip" to see the current IP address of the I-7188EG

8. Type "ip xxx.xxx.xxx.xxx" to set to a new IP address.
   Ex: i7188E> ip 192.168.1.200

9. Type "mask" to see the current address mask of the I-7188EG.

10. Type "mask xxx.xxx.xxx.xxx" to set to a new address mask.
   Ex: i7188E> mask 255.255.255.0

11. Type "gateway" to see the current gateway address.
   i7188E> gateway

12. Type "gateway xxx.xxx.xxx.xxx" to set to a new gateway address.
   i7188E> gateway 192.168.1.1

13. Press ALT_X to exit "7188xw", or COM1/COM2 of the PC will be occupied.

14. Remove the connection between "INIT" - "GND", recycle the power of the
   I-7188EG/μPAC-7186EG controller.

```
i7188E>ip
IP=192.168.255.1
i7188E>ip 192.168.1.200
Set IP=192.168.1.200
[ReadBack]IP=192.168.1.200
i7188E>mask
MASK=255.255.0.0
i7188E>mask 255.255.255.0
Set MASK=255.255.255.0
[ReadBack]MASK=255.255.255.0
i7188E>gateway
Gateway=192.168.0.1
i7188E>gateway 192.168.1.1
Set GATEWAY=192.168.1.1
[ReadBack]Gateway=192.168.1.1
i7188E>_
```

The setting steps for I-8x37/iP-8x47 controller:

1. Create a file folder named "8000" in your hard drive. For example, "c:\8000".

2. Copy CD-ROM: \Napdos\ISaGRAF\8000\Driver\....\7188xw.exe, 7188xw.ini from the CD_ROM into your "8000" folder.

3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen will appear.

4. Link from COM1 or COM2 of PC to COM1 of the I-8437 / 8837 controller by a RS232 cable.

   If you use other COM port (ex.COM5), please modify the first line of "7188xw.ini".
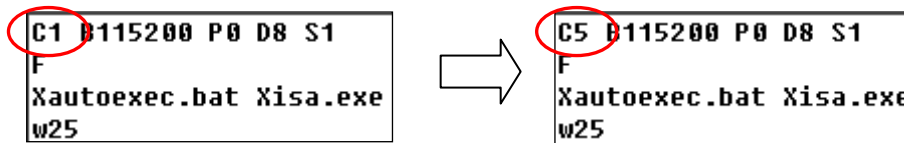
5. Power off the I-8x37/iP-8x47 controller, connect pin "INIT" to "INIT COM", then power it up.

6. If the connection is Ok, messages will appear on the 7188xw screen.
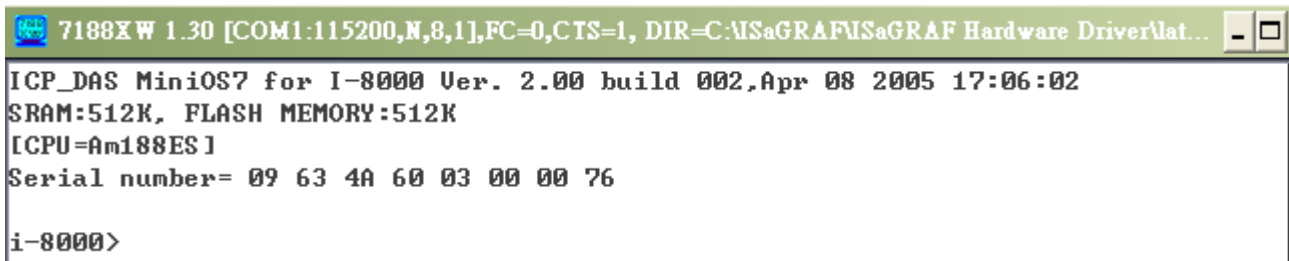
7. Use dos command (ipconfig) to search out the setup of network.
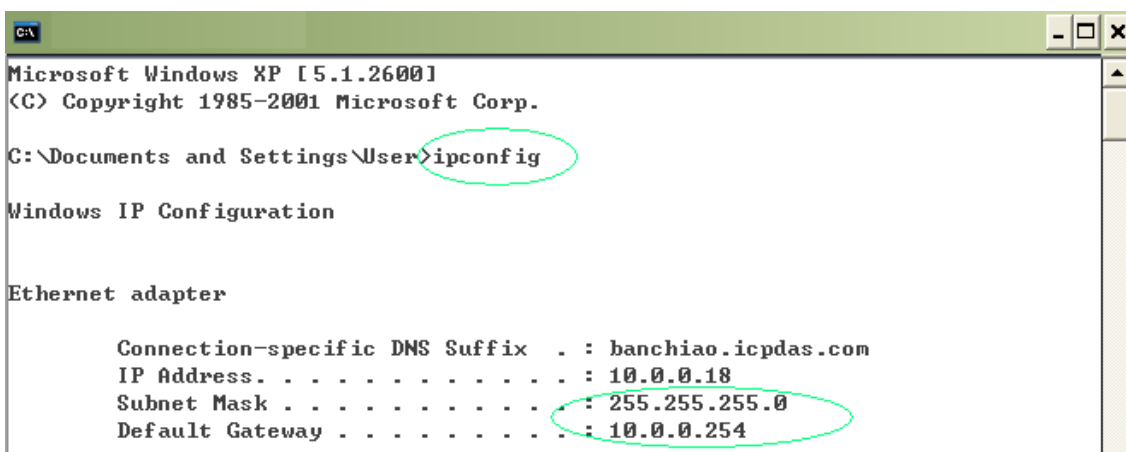
★ Please according as your computer to following setup (IP/MASK/GATEWAY)

8. Type "ip" to see the current IP address of the I-8x37/iP-8x47.

Type "**ip 10.0.0.xxx**" to setup a new IP address

```
7188XW 1.30 [COM1:115200,N,8,1],FC=0,CTS=1, DIR=C:\ISaGRAF\ISaGRAF Hardware Driver\lat...  _ □ ×
7188x for WIN32 version 1.30 (2005/11/29)[By ICPDAS. Tim Tsai.]
[Begin Key Thread...]Current set: Use COM1 115200,N,8,1
AutoRun:
Autodownload files: autoexec.bat isa.exe
Current work directory="C:\ISaGRAF\ISaGRAF Hardware Driver\latest_8k\latest_8k\3
.16"
original baudrate = 115200!
now baudrate = 115200!


i-8000>ip
IP=10.0.0.123
i-8000>ip 10.0.0.123
Set IP=10.0.0.123
[ReadBack]IP=10.0.0.123
i-8000>_
```

9. Type "mask" to see the current address mask of the I-8x37/iP-8x47.
Type "**mask 255.255.255.0**" to setup a new address mask.

```
i-8000>mask
MASK=255.255.255.0
i-8000>mask 255.255.255.0
Set MASK=255.255.255.0
[ReadBack]MASK=255.255.255.0
i-8000>
```

10. Type "gateway" to see the current gateway address.
Type "**gateway 10.0.0.254**" to setup a new gateway address.

```
i-8000>gateway
Gateway=10.0.0.254
i-8000>gateway 10.0.0.254
Set GATEWAY=10.0.0.254
[ReadBack]Gateway=10.0.0.254
i-8000>
```

11. Press **ALT_X** to exit "7188xw", or COM1/COM2 of the PC will be occupied.

12. Remove the connection between "INIT" - "INIT COM", recycle the power of I-8x37/iP-8x47 controller.

Appendix C : Update The I-8417/8817/8437/8837 Controller to New Hardware Driver

The ISaGRAF embedded driver is firmware burned into the flash memory of the I-8417 / 8817 / 8437 / 8837 controller. It can be easily upgraded by the user.

For updating driver of the I-7188EG/XG, µPAC-7186EG and ISaGRAF WinCE PAC, please refer to the "Getting Started" Manual in their respective folder of CD_ROM : Napdos\ISaGRAF\ or at http://www.icpdas.com/products/PAC/i-8000/getting_started_manual.htm .

Our newly released driver can be obtained from the following websites:
    Note: The file name will be different (*.img file) for different version.
    http://www.icpdas.com/products/PAC/i-8000/isagraf.htm or
    http://www.icpdas.com/products/PAC/i-8000/isagraf-link.htm (please extract the zip file).

Warning:
    The copyright of the firmware and the ISaGRAF embedded driver belongs to ICP DAS CO., LTD. Only the I-8417/8817/8437/8837, I-7188EG/XG, µPAC-7186EG & ISaGRAF WinCE PAC have registered a legal ISaGRAF Target license. To burn an ISaGRAF embedded driver into other controllers is absolutely illegal and may be punished by law.

Note: Make sure of your current OS & driver version before you upgrade it.

To Know The Current Driver Version: (We use driver 3.16 as an example)

1. Create a file folder named "8000" in your hard drive .For example, "**c:\8000**".

2. Copy Napdos\ISaGRAF\8000\Driver\40m\3.16\Napdos\ISaGRAF\8000\Driver\40m\3.16\
    ① **7188xw.exe,** ② **7188xw.f4,** ③ **7188xw.ini,** ④ **8k050408.img,** ⑤ **autoexec.bat,**
    ⑥**isa.exe**, ⑦ **isa_data.exe** from the CD_ROM into your "**8000**" folder.

3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen will appear. (F1: help).

4. Link COM1 or COM2 of your PC to COM1 of the controller through a RS232 cable.
   If you use other COM port (ex.COM5), please modify the first line of "7188xw.ini".
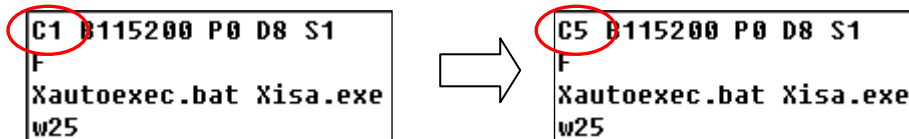
5. Power off I-8X417/8X37, connect pin "INIT" to "INIT COM" and then power it up.

6. If the connection is Ok, "i-8000>" messages will appear on the 7188xw screen.

7. Type "ver" to see the current OS version & date.

8. Type "isa *p=" to see the <u>current driver version No</u>. & setting of the controller.

```
i-8000>ver

ICP_DAS MiniOS7 for I-8000 Ver. 2.00 build 002,Apr 08 2005 17:06:02
SRAM:512K, FLASH MEMORY:512K
[CPU=Am188ES]
Serial number= 09 63 4A 60 03 00 00 76

i-8000>isa *p=

 Driver  : I-8xx7 : isa.exe - 3.16,Oct.25,2006
 MiniOS7 : Must use 8k050408.img
 isa_data.exe - 1.8,Oct.25,2006
 NED-ID  : 1
 COM1 is Modbus RTU slave port,19200,8,N,1
 COM3 is Modbus RTU slave port,19200,8,N,1
 Use 'isa *f=1' to free COM1, 'isa *f=0' to set COM1 as Modbus RTU

(C)Copyright:ICP DAS  CO. , LTD.     Taiwan Id:84517297_
```

To Upgrade An ISaGRAF Embedded Driver:

9. Power off the controller, connect pin "INIT" to "INIT COM" and then power it up.

10. Press "**F4**" to auto download the following files and reboot system.
(isa_data.exe, autoexec.bat, isa.exe, 8k050408.img)

```
i-8000>del /y
Total File number is 2, do you really want to delete(y/n)?

i-8000>LOAD
File will save to 8000:0000
StartAddr-->7000:FFFF
Press ALT_E to download file!
Load file:isa_data.exe[crc=E70F,0000]
Send file info. total 287 blocks
Block 287
Transfer time is: 12.844000 seconds
```

⏱ **Wait about 60 sec. to update ISaGRAF system & DO NOT REMOVE THE POWER**

```
i-8000>bios1
MiniOs7 for 8000 Ver 2.00.002, date=04/08/2005
Checking CRC-16...OK.
Update the OS code. Please wait the message <<Write Finished>>
Erase Flash [F000]
Write Flash
[FF]
<<Write Finished>>OK
Wait WDT reset system...
ICP_DAS MiniOS7 for I-8000 Ver. 2.00 build 002,Apr 08 2005 17:06:02
SRAM:512K, FLASH MEMORY:512K
[CPU=RDC 8820-D]
Serial number= 5A 5A 5A 5A 5A 5A 5A 5A
```

11. Type "dir" to make sure "autoexec.bat" and "isa.exe" are well burned

```
i-8000>DIR

0)autoexec.bat 05/21/2003 06:40:00      22[00016]8002:0000-8003:0006
1)isa.exe      10/25/2006 10:28:00 180678[2C1C6]8005:0006-AC21:000C
Total File number is 2  Free space=277956 bytes
```

12. Press ALT+X to exit "7188xw".

13. Remove the connection between "INIT" - "INIT COM", recycle the power of the controller.

Appendix C.1: Setting I-8xx7 & I-7188EG's COM1 As None-Modbus-Slave port

COM1 of the I-8417/8817/8437/8837, I-7188EG or µPAC-7186EG supports Modbus RTU Slave protocol by default. User may change it to a None-Modbus-Slave port for other usage. For example, user may write his own defined protocol on COM1 or use COM1 as a Modbus Master port.

Note : For **7188XG**, COM1 is for Modbus RTU Slave protocol **ONLY, can't be free**.

Steps: ** We use I-7188EG as an example. (The description in the parentheses is for I-8000.)

1. Create a file folder named "7188" in your hard drive. Ex: "c:\7188". (**I-8000:** Ex. "c:\8000")

2. Copy \Napdos\ISaGRAF\7188EG\Driver\7188xw.exe, 7188xw.ini, from the CD_ROM into your "7188" folder. (**I-8000:** please copy to "8000" folder)

3. Run "\7188\7188xw.exe".   (For Windows NT, Windows 2000 & Windows XP)

4. Link from COM1 of your PC to COM1 of the I-7188EG/µPAC-7186EG by a RS232 cable.
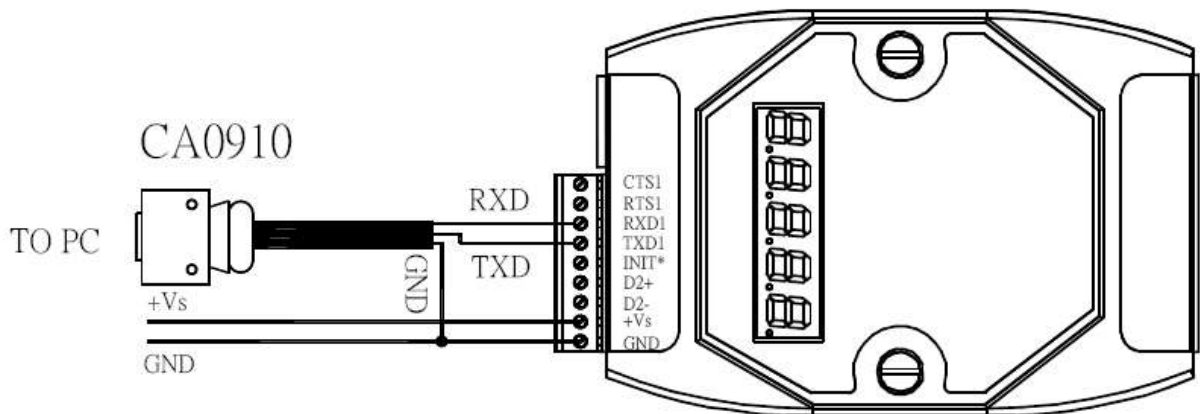
If your computer has no COM1/COM2 or you use other COM (like COM5) to link the I-7188, you can change the "C number" in the first line of "7188xw.ini" file.

EX: Using computer's COM5 to link to I-7188

```
C1 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```
⟹
```
C5 B115200 P0 D8 S1
F
Xautoexec.bat Xisa7188e.exe
w25
```
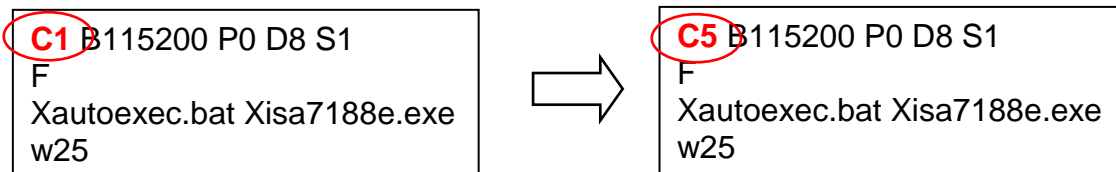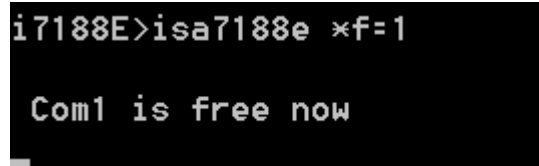
5. Power off 7188EG/7186EG, connect pin "INIT*" to "GND", then power up.
   (**I-8000:** connect "INIT" & " INIT COM")

6. If the connection is Ok, "i7188E>" messages will appear on the screen.
   (**I-8000:** "i-8000>")

7. Type "isa7188e *f=1" to free COM1 (set COM1 as none-Modbus-Slave port)

     i7188E> isa7188e *f=1   (for I-7188EG)
     i7188E> isa7186e *f=1   (for μPAC-7186EG)
     i-8000> isa *f=1   (for I-8000)

8. Press ALT_X to exit "7188xw", or COM1/COM2 of the PC will be occupied.

9. Remove the connection between "INIT" - "GND", recycle the power of the controller.
   (**I-8000:** remove the connection of "INIT"–"INIT COM")

```
i7188E>isa7188e *f=1

Com1 is free now
```

Important Note:

If user wants COM1 to be back to a Modbus RTU Slave port again, follow the same steps as above & then type "isa7188e *f=0" as below

     Ex1:   I-7188E> isa7188e *f=0   (for I-7188EG)
     Ex2:   I-7188E> isa7186e *f=0   (for μPAC-7186EG)
     Ex3:   I-8000> isa *f=0   (for I-8000)

Appendix D : Table of The Analog IO Value

I-87013, I-7013, I-7033, I-7015, M-7015, M-7033, I-87015

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
20 (Default)	Platinum 100 a = 0.00385 -100 ~ 100 °C	Temperature (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+138.50	+060.60
21	Platinum 100 a = 0.00385 0 ~ 100 °C	Temperature (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+138.50	+100.00
22	Platinum 100 a = 0.00385 0 ~ 200 °C	Temperature (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+175.84	+100.00
23	Platinum 100 a = 0.00385 0 ~ 600 °C	Temperature (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+313.59	+100.00
24	Platinum 100 a = 0.003916 -100 ~ 100 °C	Temperature (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+139.16	+060.60
25	Platinum 100 a = 0.003916 0 ~ 100 °C	Temperature (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+139.16	+100.00
26	Platinum 100 a = 0.003916 0 ~ 200 °C	Temperature (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+177.14	+100.00
27	Platinum 100 a = 0.003916 0 ~ 600 °C	Temperature (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+317.28	+100.00

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
28	Nickel 120 -80 ~ 100 °C	Temperature (Celsius)	+100.0	-80.0
		Decimal Value	+32767	-26214
		2's complement HEX	7FFF	999A
		Ohms	+200.64	+066.60
29	Nickel 120 0 ~ 100 °C	Temperature (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
		Ohms	+200.64	+120.60
2A	Platinum 1000 a = 0.00385 -200 ~ 600 °C	Temperature (Celsius)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+3137.1	+0185.2
2B*	Cu 100 a = 0.00421 -20 ~ +150 °C	Temperature (Celsius)	+150.0	-20.0
		Decimal Value	+32767	-4369
		2's complement HEX	7FFF	EEEF
		Ohms	+163.17	+091.56
2C*	Cu 100 a = 0.00427 0 ~ 200 °C	Temperature (Celsius)	+200.0	0.0
		Decimal Value	+32767	0
		2's complement HEX	7FFF	0
		Ohms	+167.75	+090.34
2D*	Cu 1000 a = 0.00421 -20 ~ 150 °C	Temperature (Celsius)	+150.0	-20.0
		Decimal Value	+32767	-4369
		2's complement HEX	7FFF	EEEF
		Ohms	+1631.7	+0915.6
2E	Platinum 100 a = 0.00385 -200 ~ 200 °C	Temperature (Celsius)	+200.0	-200.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+175.84	+018.49
2F	Platinum 100 a = 0.003916 -200 ~ 200 °C	Temperature (Celsius)	+200.0	-200.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
		Ohms	+177.14	+017.14
80	Platinum 100 a = 0.00385 -200 ~ 600 °C	Temperature (Celsius)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+313.59	+018.49

Range Type Code (Hex)	RTD Type	Data Format	Max Value	Min Value
81	Platinum 100 a = 0.003916 -200 ~ 600 °C	Temperature (Celsius)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556
		Ohms	+317.28	+017.14

* Range Type Code 2B, 2C and 2D are only supported by I-7015, M-7015 & I-87015.
* I-87015, I-7015 & M-7015: Each channel can be configured to different range ID.

I-8017H(8-ch), I-8017HS(16-ch)

Range Type Code (Hex)	Data Format	Max value	Min value
05	Input Range	+2.5 V	-2.5 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
06*	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
07	Input Range	+1.25 V	-1.25 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* I-8017H: Each channel can be configured to different range ID.

* Using Code 06, module needs to connect with an additional 125Ω resistor in each current input channel.

I-87017, I-87017R, I-7017, I-7017R, M-7017, M-7017R

Range Type Code (Hex)	Data Format	Max value	Min value
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0A	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0B	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0C	Input Range	+150.0 mV	-150.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0D*	Input Range (with 125 ohms resistor)	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* Using Code 0D, module needs to connect with an additional 125Ω resistor in each current input channel.

I-7017RC, M-7017RC, I-87017RC

Range Type Code (Hex)	Data Format	Max value	Min value
7	Input Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0
D	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
1A	Input Range	+20.0 mA	0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0

* I-7017RC, M-7017RC & I-87017RC modules do not need an additional resistor when using them to measure current.

I-87018Z, I-87018R, I-87018, I-87019R, I-7018Z, I-7018R, I-7018, M-7018, M-7018R, I-7019R, M-7019R (1)

Range Type Code (Hex)	Data Format	Max value	Min value
00	Input Range	-15.0 mV	-15.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
01	Input Range	+50.0 mV	-50.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
02	Input Range	+100.0 mV	-100.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
03	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
04	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
05 (Default)	Input Range	+2.5V	-2.5V
	Decimal Value	+100.00	-100.00
	2's Complement HEX	7FFF	8000
06*	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

* I-7019, I-7019R, M-7019, M-7019R & I-87019R measure current (Code 6) by jumper setting, but the other Modules need to add an additional 125Ω resistor for measuring current.

* I-87018Z & I-7018Z: 10 Channels, each channel can be configured to different Type Code.

➢ **I-87018Z & I-7018Z** also support the types listed below:

Range Type Code (Hex)	Data Format	Max value	Min value
7	Input Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0
1A	Input Range	+20.0 mA	0 mA
	Decimal Value	+32767	0
	2's Complement HEX	7FFF	0

I-87018Z, I-87018R, I-87018, I-87019R, I-7018Z, I-7018R, I-7018, M-7018, M-7018R, I-7019R, M-7019R (2)

Range Type Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
0E	J Type -210 ~ 760 °C	Temperature (Celsius)	+760.0	-210.0
		Decimal Value	+32767	-9054
		2's Complement HEX	7FFF	DCA2
0F	K Type -270 ~ 1372 °C	Temperature (Celsius)	+1372.0	-270.0
		Decimal Value	+32767	-6448
		2's Complement HEX	7FFF	E6D0
10	T Type -270 ~ 400 °C	Temperature (Celsius)	+400.0	-270.0
		Decimal Value	+32767	-22118
		2's Complement HEX	7FFF	A99A
11	E Type -270 ~ 1000 °C	Temperature (Celsius)	+1000.0	-270.0
		Decimal Value	+32767	-8847
		2's Complement HEX	7FFF	DD71
12	R Type 0 ~ 1768 °C	Temperature (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
13	S Type 0 ~ 1768 °C	Temperature (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
14	B Type 0 ~ 1820 °C	Temperature (Celsius)	+1820.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
15	N Type -270 ~ 1300 °C	Temperature (Celsius)	+1300.0	-270.0
		Decimal Value	+32767	-6805
		2's Complement HEX	7FFF	E56B
16	C Type 0 ~ 2320 °C	Temperature (Celsius)	+2320.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
17	L Type -200 ~ 800 °C	Temperature (Celsius)	+800.0	-200.0
		Decimal Value	+32767	-8192
		2's Complement HEX	7FFF	E000
18	M Type -200 ~ 100 °C	Temperature (Celsius)	+100.0	-200.0
		Decimal Value	+16384	-32768
		2's Complement HEX	4000	8000

➢ **I-87018Z & I-7018Z** also support the types listed below:

Range Type Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
19	L Type DIN43710 -200 ~ 900°C	Temperature (Celsius)	+900.0	-200.0
		Decimal Value	+32767	-7281
		2's Complement HEX	7FFF	E38F

➢ **I-7019, I-7019R, M-7019, M-7019R & I-87019R** also support the types:

Range Type Code (Hex)	Data Format	Max value	Min value
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0A	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0B	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0C	Input Range	+150.0 mV	-150.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0D*	Input Range (with 125 ohms resistor)	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
19	L Type DIN43710 -200 ~ 900°C — Temperature (Celsius)	+900.0	-200.0
	Decimal Value	+32767	-7281
	2's Complement HEX	7FFF	E38F

* I-7019, I-7019R, M-7019, M-7019R & I-87019R: each channel can be configured to different Type Code.
* I-7019, I-7019R, M-7019, M-7019R & I-87019R can measure Code 0D current by jumper, they don't need to add an additional 125Ω resistor.

I-7021

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
32 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

I-7022

Range Type Code (Hex)	Data Format	Max Value	Min Value
0	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
1	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
2 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

Range Type Code (Hex)	Thermistor Type	Data Format	Max Value	Min Value
60	PreCon Type III 10K @ 25 °C -35 ~ 115 °C	Temperature(Fahrenheit)	+240.00 °F	-030.00 °F
		2's complement HEX	7FFF	E000
		Ohms	+000539.4	+173600.0
61	Fenwell U 2K @ 25 °C -50 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000037.2	+134020.0
62	Fenwell U 2K @ 25 °C 0 ~ 150 °C	Temperature(Celsius)	+150.00 °C	+000.00 °C
		2's complement HEX	7FFF	0000
		Ohms	+000037.2	+006530.0
63	YSI L Mix 100 @ 25 °C -80 ~ 100 °C	Temperature(Celsius)	+100.00 °C	-080.00 °C
		2's complement HEX	7FFF	999A
		Ohms	+000014.3	+014470.0
64	YSI L Mix 300 @ 25 °C -80 ~ 100 °C	Temperature(Celsius)	+100.00 °C	-080.00 °C
		2's complement HEX	7FFF	999A
		Ohms	+000035.8	+067660.0
65	YSI L Mix 1000 @ 25 °C -70 ~ 100 °C	Temperature(Celsius)	+100.00 °C	-070.00 °C
		2's complement HEX	7FFF	A667
		Ohms	+000106.4	+132600.0
66	YSI B Mix 2252 @ 25 °C -50 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000041.8	+151000.0
67	YSI B Mix 3000 @ 25 °C -40 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-040.00 °C
		2's complement HEX	7FFF	DDDE
		Ohms	+000055.6	+101000.0
68	YSI B Mix 5000 @ 25 °C -40 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-040.00 °C
		2's complement HEX	7FFF	DDDE
		Ohms	+000092.7	+168300.0
69	YSI B Mix 6000 @ 25 °C -30 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000111.5	+106200.0

Range Type Code (Hex)	Thermistor Type	Data Format	Max Value	Min Value
6A	YSI B Mix 10K @ 25 °C -30 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000185.9	+177000.0
6B	YSI H Mix 10K @ 25 °C -30 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-030.00 °C
		2's complement HEX	7FFF	E667
		Ohms	+000237.0	+135200.0
6C	YSI H Mix 30K @ 25 °C -10 ~ 200 °C	Temperature(Celsius)	+200.00 °C	-010.00 °C
		2's complement HEX	7FFF	F99A
		Ohms	+000186.7	+158000.0
70 ~ 77	User-defined -50 ~ 150 °C	Temperature(Celsius)	+150.00 °C	-050.00 °C
		2's complement HEX	7FFF	D556
		Ohms	+000000.0	+000000.0

* When using the User-defined Type, if the resistor is larger than 180000 ohms, it will be recognized as "lower than the standard range".

* Please refer to the Section 1.11 of the "I-7005/M-7005 User's Manual".

I-8024

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
33	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768

* Each channel can be configured to different Range Type Code.

I-87024, I-7024

Range Type Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
32	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
33 (Default)	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
34	Output Range	+5.0 V	+0.0 V
	Decimal Value	+32767	+0
35	Output Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768

I-87022, I-87026

Range Type Code (Hex)	Data Format	Max Value	Min Value
0	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
1	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
2	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	0

* I-87022, I-87026: Each channel can be configured to different Range Type Code

Appendix E : LANGUAGE REFERENCE

copyright AlterSys
printed with permission

ISaGRAF

Version 3.46

LANGUAGE REFERENCE

AlterSys Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of **AlterSys Inc**. The software, which includes information contained in any databases, described in this document is furnished under a license agreement or nondisclosure agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of **AlterSys Inc**.

E.1 Project architecture

An ISaGRAF project is divided into several programming units called **programs**. The programs of the project are linked together in a tree-like architecture. Programs can be described using any of **SFC, FC (Flow Chart)**, **FBD**, **LD, ST** or **IL** graphic or literal languages.

E.1.1 Programs

A **program** is a logical programming unit, which describes operations between **variables** of the process. Programs describe either **sequential** or **cyclic** operations. Cyclic programs are executed at each target system cycle. The execution of sequential programs follows the dynamic rules of either the **SFC** language or the **FC** language.

Programs are linked together in a hierarchy tree. Programs placed on the top of the hierarchy are activated by the system. Sub-programs (lower level of the hierarchy) are activated by their father. A program can be described with any of the available graphic or literal following languages:

Sequential Function Chart (SFC) for high level programming
Flow Chart (FC) for high level programming
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
Instruction List (IL) for low level operations

The same program cannot mix several languages, except LD and FBD can be combined in one diagram.

E.1.2 Cyclic and sequential operations

The hierarchy of programs is divided into four main **sections** or groups:

Begin   programs executed at the beginning of each target cycle
Sequential programs following SFC or FC dynamic rules
End   programs executed at the end of each target cycle
Functions   set of non-dedicated sub-programs

Programs of the **'Begin'** or **'End'** sections describe cyclic operations, and are not time dependent. Programs of the '**Sequential**' section describe sequential operations, where the time variable explicitly synchronises basic operations. Main programs of the **'Begin'** section are systematically executed at the beginning of each run time cycle. Main programs of the **'End'** section are systematically executed at the end of each run time cycle. Main programs of the **'Sequential'** section are executed according to either the **SFC** or the **FC** dynamic rules.

Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. A program of the "**Function**" section can call another program of this section.

Main and child programs of the sequential section must be described with **SFC** or **FC** language. Programs of cyclic sections (**begin** and **end**) cannot be described with **SFC** or **FC** language. Any program of any section may own one or more sub-programs. Any program of the sequential section may own one or more **SFC** or **FC** child programs (according to its own programming language). Sub-programs cannot be described with **SFC** or **FC** language.

Programs of the **Begin** section are typically used to describe preliminary operations on input devices to build high level filtered variables. Such variables are frequently used by the programs of the **Sequential** section. Programs of the **End** section are typically used to describe security operations on the variables operated on by the **Sequential** section, before sending values to output devices.

E.1.3 Child SFC and FC programs

Any **SFC** program of the sequential section may control other **SFC** programs. Such low-level programs are called **child SFC programs**. A **child SFC program** is a parallel program that can be started, killed, frozen or restarted by its parent program. The parent program and child program must both be described with the **SFC** language. A child SFC program may have local variables and defined words.

When a parent program starts a child **SFC** program, it puts an SFC **token** (activates) into each initial step of the child program. This command is described with the **GSTART** statement. When a parent program kills a child **SFC** program, it clears all the tokens existing in the **steps** of the child. Such a command is described with the **GKILL** statement.

When a parent program freezes a child **SFC** program, it suspends its execution. The suspended program can then be restarted using the **GRST** statement.

Any **FC** program of the sequential section may control other **FC** sub-programs. An **FC** father program is blocked (waits) during execution of an FC sub-program. It is not possible that simultaneous operations are done in father FC program and one of its FC sub-programs.

E.1.4 Functions and sub-programs

A sub-program or a function execution is driven by its parent program. The execution of the parent program is suspended until the sub-program or the function ends:
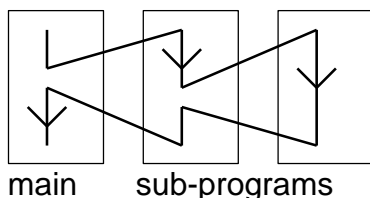
main   sub-programs

Any program of any section may have one or more sub-programs. A sub-program is owned by only one father program. A sub-program may have local variables and defines. Any language but **SFC** or **FC** can be used to describe a sub-program. Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. Unlike other sub-programs, they are not dedicated to one father program. A

program of the "**Function**" section can call another program of this section. A function can be located in the Library.

Warning: The ISaGRAF system does not support **recursive function calls**. A run time error will occur if a program of the "**Functions**" section is called by itself or by one of its called sub-program.

Warning: A function or sub-program does not "store" the local value of its local variables. A function or sub-program is not instantiated and so can not call function blocks.

The interface of a sub-program must be explicitly defined, with a **type** and a **unique name** for each of its calling or return parameter. In order to support the **ST** language convention, the return parameter must have the same name as the sub-program.

The following table shows how to set the value of the return parameter in the body of a sub-program, in the different languages:
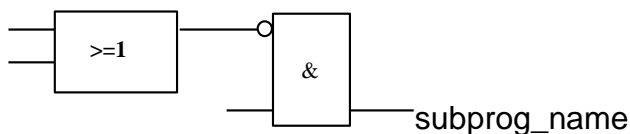
ST: assign the return parameter using its name
(the same name as the sub-program):

subprog_name := <expression>;
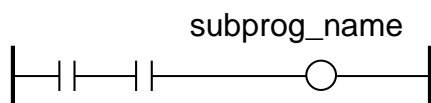
IL: the value of the current result (IL register)
at the end of the sequence is stored in the return parameter:

LD   10
ADD   20   (* return parameter value = 30 *)

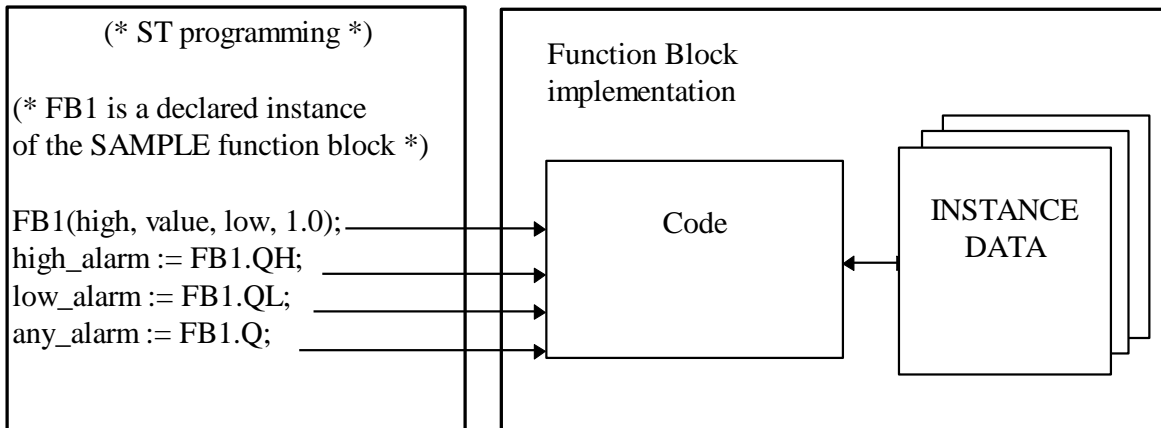**FBD:**   set the return parameter using its name:

LD: use a coil symbol with the name of the return parameter:
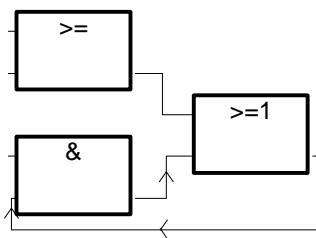

E.1.5 Function blocks

Function blocks can use the languages: LD, FBD, ST or IL. Function blocks are instantiated. It means local variables of a function block are copied for each instance. When calling a block in a program, you actually call the instance of the block: the same code is called but the data used are the one which have been allocated for the instance. Values of the variables of the instance are stored from one cycle to the other.

```
(* ST programming *)

(* FB1 is a declared instance
of the SAMPLE function block *)

FB1(high, value, low, 1.0);
high_alarm := FB1.QH;
low_alarm := FB1.QL;
any_alarm := FB1.Q;
```

Function Block
implementation

Code

INSTANCE
DATA

Warnings:
- A function block written with one of the IEC languages can not call other function blocks: the instantiation mechanism only manages the local variables of the block itself. Here is the list of standard function blocks that you cannot use inside an IEC function block:
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALRM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

- For the same reason, you can not use Positive or Negative contact or coils, or Set and Reset coils.

- TSTART and TSTOP functions to start and stop timers cannot be used in a function block for 3.0x targets. It works since the 3.20 target.

- When you need loop in your function block, you must use local variable before doing the loop. See the example below:

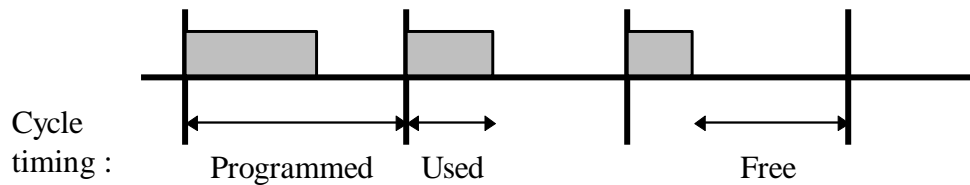This will not work: This is OK:
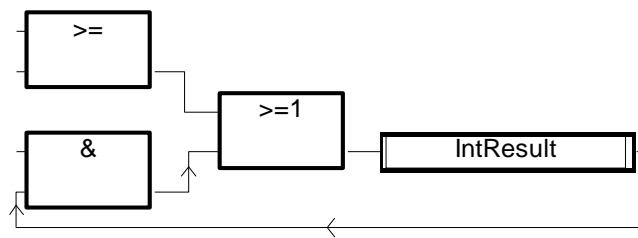
E.1.6 Description language

A program can be described with any of the following graphic or literal languages:

Sequential Function Chart (SFC) for high level operations
Flow Chart (FC) for high level operations
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
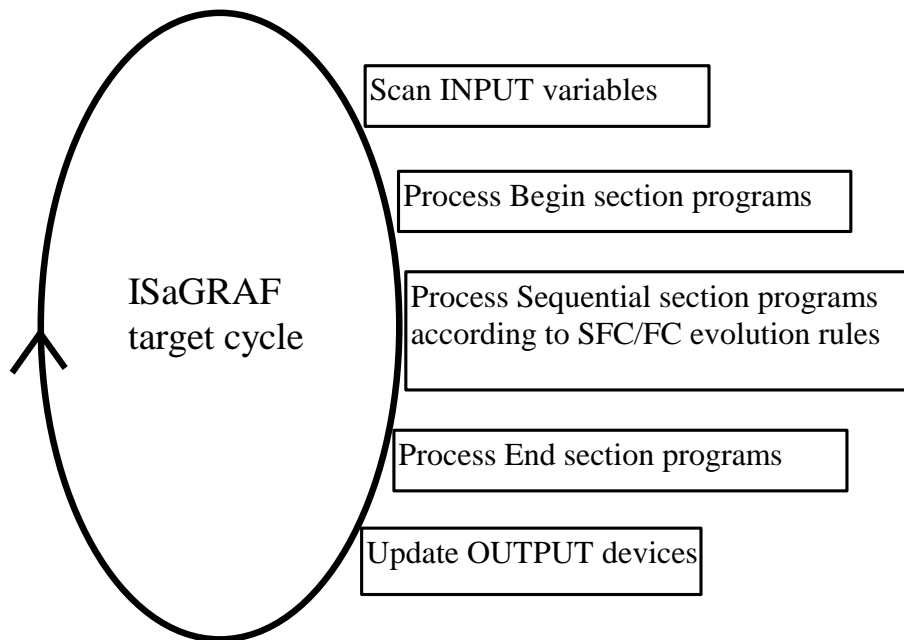**Instruction List** (IL) for low level operations

The same program cannot mix several languages. The language used to describe a program is chosen when the program is created, and cannot be changed later on. The exception is that it is possible to combine FBD and LD in a single program.

E.1.7 Execution rules

ISaGRAF is a **synchronous** system. All the operations are triggered by a clock. The basic duration of the clock is called the cycle timing:

Cycle timing :

             Programmed   Used   Free

Basic operations processed during a target cycle are:

ISaGRAF target cycle

Scan INPUT variables

Process Begin section programs

Process Sequential section programs according to SFC/FC evolution rules

Process End section programs

Update OUTPUT devices

This system makes it possible to:

- guarantee that an input variable keeps the same value within a cycle,
- guarantee that an output device is not updated more than once in a cycle,
- work safely on the same global variable from different programs,
- estimate and control the response time of the complete application.

E.2 Common objects

These are main features and common **objects** of the ISaGRAF programming database. Such objects can be used in any program written with any of the **SFC**, **FC**, **FBD**, **LD**, **ST** or **IL** languages.

E.2.1 Basic types

Any constant, expression or variable used in a program (written in any language) must be characterised by a type. Type coherence must be followed in graphic operations and literal statements. These are the available basic types for programming objects:

BOOLEAN: logic (true or false) value
ANALOG: integer or real (floating) continuous value
TIMER: time value
MESSAGE: character string

Note: Timers contain values less than one day and cannot be used to store dates.

E.2.2 Constant expressions

Constant expressions are relative to one type. The same notation cannot be used to represent constant expressions of different types.

E.2.2.1 Boolean constant expressions

There are only two boolean constant expressions:

TRUE is equivalent to the integer value 1
FALSE is equivalent to the integer value 0

"True" and "False" keywords are case insensitive.

E.2.2.2 Integer analog constant expressions

Integer constant expressions represent signed long integer (32 bit) values: from **-2147483647** to **+2147483647**. Integer analog constants may be expressed with one of the following **bases**. Integer constants must begin with a **prefix** that identifies the bases used:

Base	Prefix	Example
DECIMAL	(none)	-908
HEXADECIM AL	"16#"	16#1A2B3C4D
OCTAL	"8#"	8#1756402
BINARY	"2#"	2#1101_0001_0101_110 1

The underscore character ('_') may be used to separate groups of digits. It has no particular significance, and is used to increase constant expression readability.

E.2.2.3 Real analog constant expressions

Real analog constant expressions can be written with either **decimal** or **scientific** representation. The **decimal point** ('**.**') separates the integer and decimal parts. The decimal point must be used to differentiate a real constant expression from an integer one. The scientific representation uses the '**E**' or '**F**' letter to separate the **mantissa** part and the **exponent**. Exponent part of a real scientific expression must be a signed integer value from **-37** to **+37**. Below are examples of real analog constant expressions:

```
3.14159  -1.0E+12
+1.0      1.0F-15
-789.56  +1.0E-37
```

The expression "**123**" does not represent a real constant expression. Its correct real representation is "**123.0**".

E.2.2.4 Timer constant expressions

Timer constant expressions represent time values from **0 second** to **23h59m59s999ms**. The lowest allowed unit is a millisecond. Standard time units used in constant expressions are:

Hour   The "**h**" letter must follow the number of hours
Minute   The "**m**" letter must follow the number of minutes
Second   The "**s**" letter must follow the number of seconds
Millisecond The "**ms**" letters must follow the number of milliseconds

The time constant expression must begin with "**T#**" or "**TIME#**" prefix. Prefixes and unit letters are case insensitive. Some units may not appear. These are examples of timer constant expressions:

T#1H450MS   1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes

The expression "0" does not represent a time value, but an analog constant.

E.2.2.5 Message string constant expressions

String or message constant expressions represent character strings. Characters must be preceded by a quote and followed by an apostrophe. For example:

'THIS IS A MESSAGE'

Warning: The apostrophe '**'**' character cannot be used within a string constant expression. A string constant expression must be expressed on one line of the program source code. Its length cannot exceed 255 characters, including spaces.

Empty string constant expression is represented by two apostrophes, with no space or tab
character between them:

''   (* this is an empty string *)

The special character dollar ('**$**'), followed by other special characters, can be used in a string
constant expression to represent a non-printable character:

Sequence	Meaning	ASCII (hexa)	Example
$$	'$' character	16#24	'I paid $$5 for this'
$'	apostrophe	16#27	'Enter $'Y$' for YES'
$L	line feed	16#0a	'next $L line'
$R	carriage return	16#0d	' llo $R He'
$N	new line	16#0d0a	'This is a line$N'
$P	new page	16#0c	'lastline $P first line'
$T	tabulation	16#09	'name$Tsize$Tdate'
$hh (*)	any character	16#hh	'ABCD = $41$42$43$44'

(*) "**hh**" is the hexadecimal value of the ASCII code for the expressed character.

E.2.3 Variables

Variables can be **LOCAL** to one program, or **GLOBAL**. Local variables can be used by one
program only. Global variables can be used in any program of the project. Variable names
must conform to the following rules:

name cannot exceed **16** characters
first character must be a **letter**
following characters can be **letters**, **digits** or the underscore character

E.2.3.1 Reserved keywords

A list of the reserved keywords is shown below. Such identifiers cannot be used to name a
program, a variable or a "C" function or function block:

A   ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B   BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME,
         BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL,
         BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,
C   CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D   DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
E   ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM,
         END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR,
         END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F   FALSE, FEDGE, FIND, FOR, FUNCTION,
G   GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,

I   IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING,
         INT_TO_TIME,
J   JMP, JMPC, JMPCN, JMPN, JMPNC,
L   LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
M   MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
N   NE, NOT,
O   OF, ON, OPERATE, OR, OR_MASK, ORN,
P   PROGRAM
R   R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL,
         REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT,
         REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC,
         RETURN, RIGHT, ROL, ROR,
S   S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD,
         STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME,
         STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL,
         SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA,
         SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO,
         SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM,
         SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,
T   TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL,
         TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE,
         TSTART, TSTOP, TYPE,
U   UDINT, UINT, ULINT, UNTIL, USINT,
V   VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT,
         VAR_INPUT, ,VAR_OUTPUT,
W   WHILE, WITH, WORD,
X   XOR, XOR_MASK, XORN

All keywords beginning with an underscore ('_') character are internal keywords and must not
    be used in textual instructions.

E.2.3.2 Directly represented variables

ISaGRAF enables the use of **directly represented variables** in the source of the programs to
    represent a free channel. Free channels are the ones which are not linked to a declared I/O
    variable. The identifier of a directly represented variable always begins with "**%**" character.

Below are the naming conventions of a directly represented variable for a channel of a single
    board. "**s**" is the slot number of the board. "**c**" is the number of the channel.

`%IXs.c`   free channel of a boolean input board
`%IDs.c`   free channel of an integer input board
`%ISs.c`   free channel of a message input board
`%QXs.c`   free channel of a boolean output board
`%QDs.c`   free channel of an integer output board
`%QSs.c`   free channel of a message output board

Below are the naming conventions of a directly represented variable for a channel of a complex equipment. "*s*" is the slot number of the equipment. "*b*" is the index of the single board within the complex equipment. "*c*" is the number of the channel.

`%IXs.b.c` free channel of a boolean input board
`%IDs.b.c` free channel of an integer input board
`%ISs.b.c` free channel of a message input board
`%QXs.b.c` free channel of a boolean output board
`%QDs.b.c` free channel of an integer output board
`%QSs.b.c` free channel of a message output board

Below are examples:

`%QX1.6`   6th channel of the board #1 (boolean output)
`%ID2.1.7` 7th channel of the board #1 in the equipment #2 (integer input)

A directly represented variable cannot have the "**real**" data type.

E.2.3.3 Boolean variables

Boolean means **logic**. Such variables can take one of the boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in boolean expressions. Boolean variables can have one of the following **attributes**:

Internal: memory variable updated by the program
Constant: read-only memory variable with an initial value
Input: variable connected to an input device (refreshed by the system)
Output: variable connected to an output device

Warning: When declaring a boolean variable, strings can be defined to replace 'true' and 'false' values during debug. Those strings cannot be used in the programs unless entered as **'defined words'** for the language.

E.2.3.4 Analog variables

Analog means **continuous**. Such variables have signed integer or real (floating) values. Available formats for an analog variable are:

Integer   32 bit signed integer: from **-2147483647** to **+2147483647**
Real   standard IEEE 32 bit floating value (single precision)
    1 sign bit + 23 mantissa bits + 8 exponent bits

REAL analog exponent value cannot be less than **-37** or greater than **+37**. Analog variables can have one of the following **attributes**:

Internal   memory variable updated by the program
Constant: read-only memory variable with an initial value
Input   variable connected to an input device (refreshed by the system)

Output   variable connected to an output device

Note: When a real variable is connected to an I/O device, the corresponding I/O driver operates the equivalent integer value.

Warning: Integer and real analog variables or constant expressions cannot be mixed in the same analog expression.

E.2.3.5 Timer variables

Timer means **clock** or **counter**. Such variables have time values and are typically used in time expressions. A timer value cannot exceed **23h59m59s999ms** and cannot be negative. Timer variables are stored in 32 bit words. The internal representation is a positive number of milliseconds.
Timer variables can have one of the following **attributes**:

Internal   memory variable managed by the program, refreshed by ISaGRAF system
Constant: read-only memory variable with an initial value

Warning: Timer variables cannot have the INPUT or OUTPUT attributes.

Timer variables can be automatically refreshed by the ISaGRAF system. When a timer is **active**, its value is automatically increased according to the target system real time clock. The following statements of the **ST** language can be used to control a timer:

TSTART   starts automatic refresh of a timer
TSTOP   stops automatic refresh of a timer

E.2.3.6 Message string variables

Message or string variables contain character strings. The length of the string can change during process operations. The length of a message variable cannot exceed the capacity (maximum length) specified when the variable is declared. Message capacity is limited to 255 characters. Message variables can have one of the following **attributes**:

Internal   memory variable updated by the program
Constant: read-only memory variable with an initial value
Input   variable connected to an input device (refreshed by the system)
Output   variable connected to an output device

String variables can contain any character of the standard ASCII table (ASCII code from **0** to **255**). The null character can exist in a character string. Some "C" functions of the standard ISaGRAF library will not correctly operate messages which contain null (**0**) characters.

E.2.4 Comments

Comments may be freely inserted in literal languages such as **ST** and **IL**. A comment must begin with the special characters "**(***" and terminate with the characters "***)**". Comments can be inserted anywhere in a **ST** program, and can be written on more than one line.

These are examples of comments:

counter := ivalue;   (* assigns the main counter *)
(* this is a comment expressed
on two lines *)
c := counter (* you can put comments anywhere *) + base_value + 1;

Interleave comments cannot be used. This means that the "**(***" characters cannot be used within a comment.

Warning: The IL language only accepts comments as the last component of an instruction line.

E.2.5 Defined words

The ISaGRAF system allows the re-definition of constant expressions, true and false boolean expressions, keywords or complex **ST** expressions. To achieve this, an **identifier** name has to be given to the corresponding expression. For example:

YES   is   TRUE
PI   is   3.14159
OK   is   (auto_mode AND NOT (alarm))

When such equivalence is defined, its **identifier** can be used anywhere in an **ST** program to replace the attached expression. This is an example of **ST** programming using defines:

If OK Then
  angle := PI / 2.0;
  isdone := YES;
End_if;

Defined words can be **LOCAL** to one program, **GLOBAL**, or **COMMON**.
Local defined words can be used by only one program.
Global defined words can be used in any program of the project.
Common defined words can be used in any program of any project.
Note that common defined can be stored separately with the Archive manager.

Warning: When the same identifier is defined twice with different **ST** equivalencies, the last defined expression is used. For example:

Define: **OPEN**   is   **FALSE**
    OPEN   is   TRUE

means: **OPEN**   is   **TRUE**

Naming defined words must conform to following rules:
- name cannot exceed **16** characters
- first character must be a **letter**
- following characters can be **letters**, **digits** or underscore ('_') character

Warning: A defined word can not use a defined word in its definition, for example, you can not
   have:
   PI   is   3.14159
   PI2   is   PI*2
write the complete equivalence using constants or variables and operations:
   PI2   is   6.28318

E.3 SFC language

Sequential Function Chart (SFC) is a **graphic** language used to describe **sequential operations**. The process is represented as a set of well-defined **steps**, linked by **transitions**. A **boolean condition** is attached to each transition. **Actions** within the steps are detailed by using other languages (**ST**, **IL**, **LD** and **FDB**).

E.3.1 SFC chart main format

An SFC program is a graphic set of **steps** and **transitions**, linked together by **oriented links**. Multiple connection links are used to represent divergences and convergences. Some parts of the complete program may be separated and represented in the main chart by a single symbol, called **macro steps**. The basic **graphic rules** of the SFC are:
- A step cannot be followed by another step
A transition cannot be followed by another transition

E.3.2 SFC basic components

The basic components (graphic symbols) of the SFC language are: steps and initial steps, transitions, oriented links, and jumps to a step.

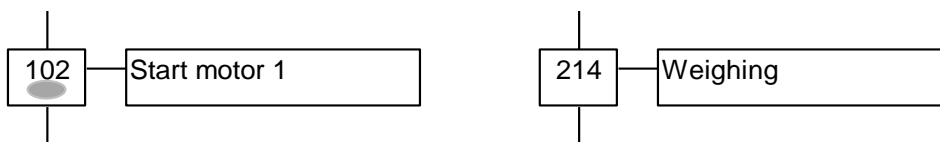### E.3.2.1 Steps and initial steps

A step is represented by a single **square**. Each step is **referenced** by a number, written in the step square symbol. A main description of the step is written in a rectangle linked to the step symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the step:
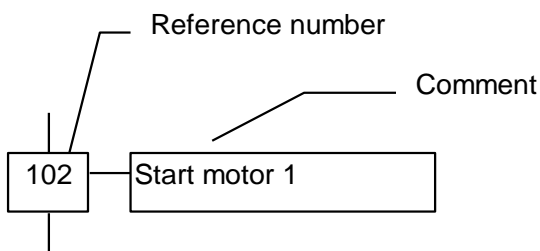
At run time, a token indicates that the step is active:
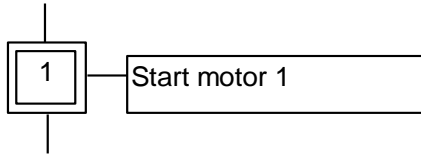
Active step: Inactive step:

The **initial situation** of an SFC program is expressed with **initial steps**. An initial step has a **double-bordered** graphic symbol. A token is automatically placed in each initial step when the program is started.

Initial step:

An SFC program must contain **at least one** initial step.

These are the attributes of a step. Such fields may be used in any of the other languages:
**GSnnn.x**   activity of the step (boolean value)
GSnnn.t   activation duration of the step (time value)
(where **nnn** is the reference number of the step)

E.3.2.2 Transitions

Transitions are represented by a small horizontal bar that crosses the connection link. Each
   transition is **referenced** by a number, written next to the transition symbol. A main
   description of the transition is written on the right side of the transition symbol. This
   description is a **free comment** (not part of the programming language). The above
   information is called the **Level 1** of the transition:
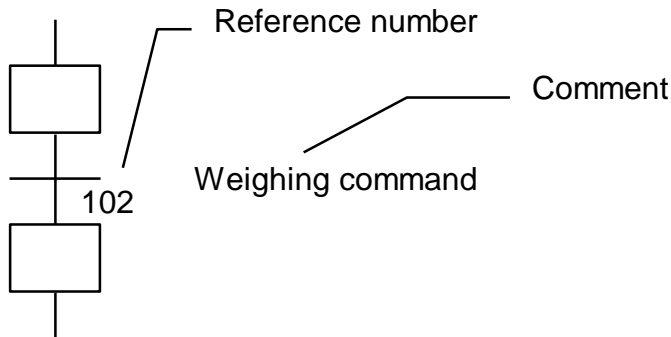
E.3.2.3 Oriented links

Single lines are used to link steps and transitions. These are oriented links. When the
   orientation is not explicitly given, the link is oriented from the top to the bottom.
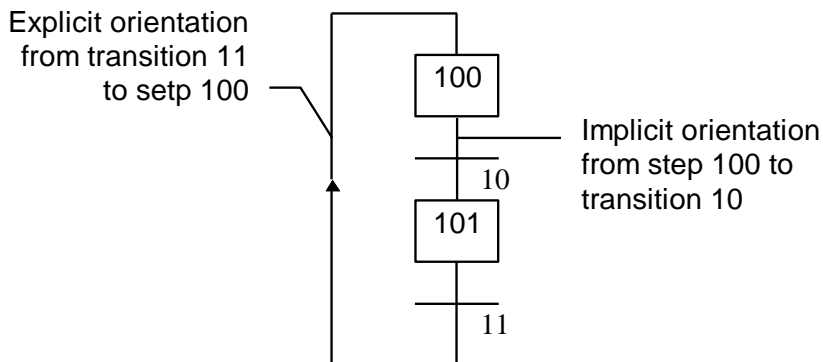
E.3.2.4 Jump to a step

Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the number of the destination step:

Jump to step 102

102
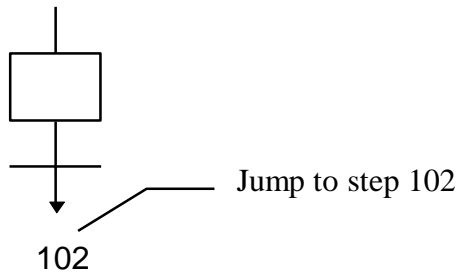
A jump symbol cannot be used to represent a link from a step to a transition. Examples of jumps - the following charts are equivalent:
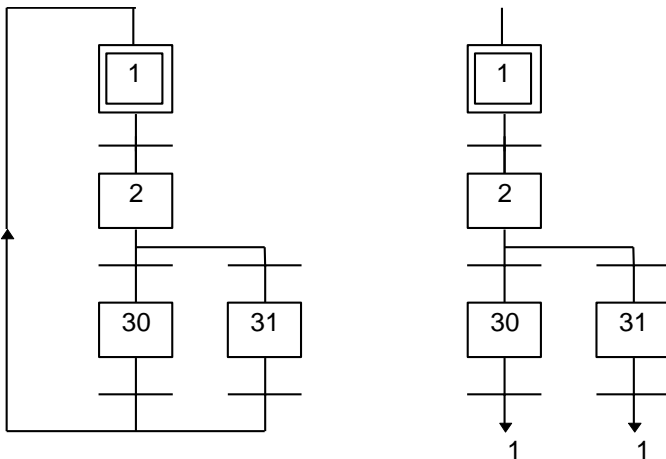

E.3.3 Divergences and convergences

Divergences are **multiple connection links** from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

E.3.3.1 single divergences

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence. Single divergences and convergences are represented by single horizontal lines.
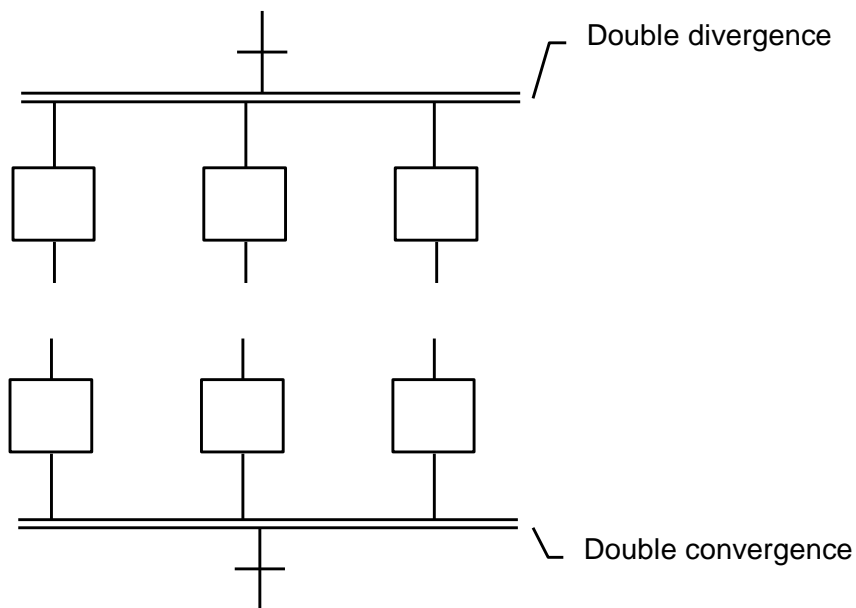
Single divergence

Single convergence
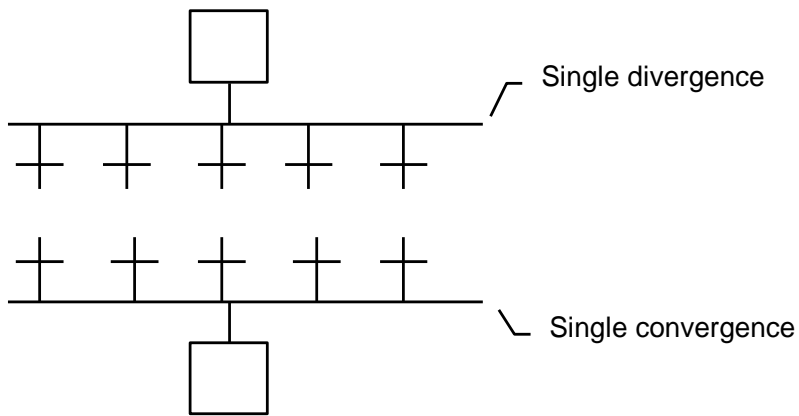
Warning: The conditions attached to the different transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of the transitions to ensure that only one token progresses in one branch of the divergence at run time. Below is an example of single divergence and convergence:

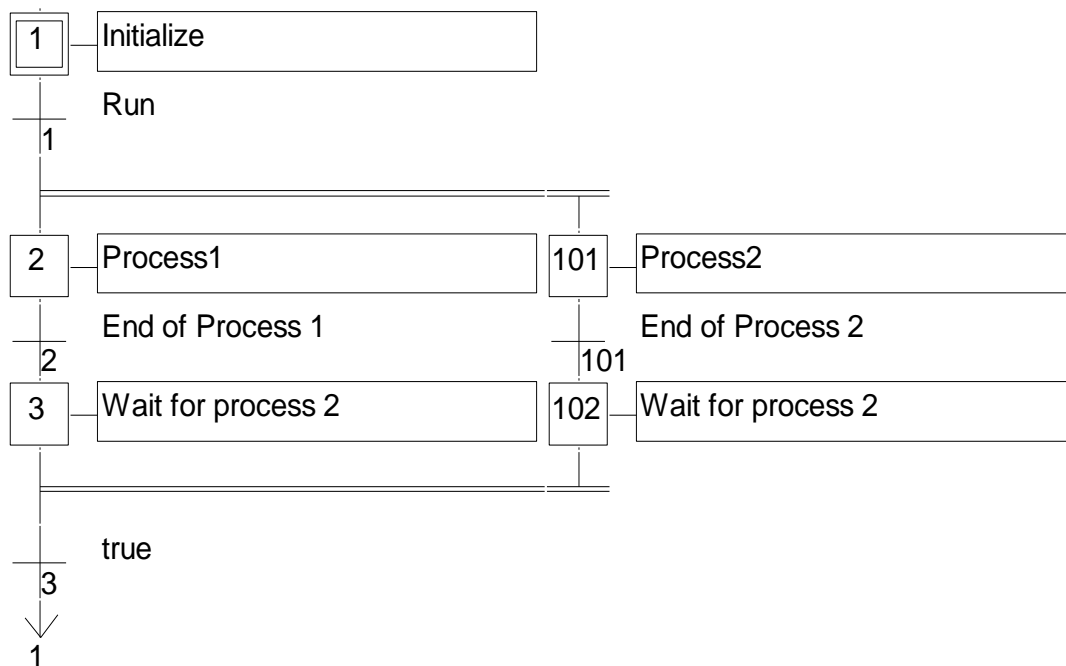(* SFC program with single divergence and convergence *)
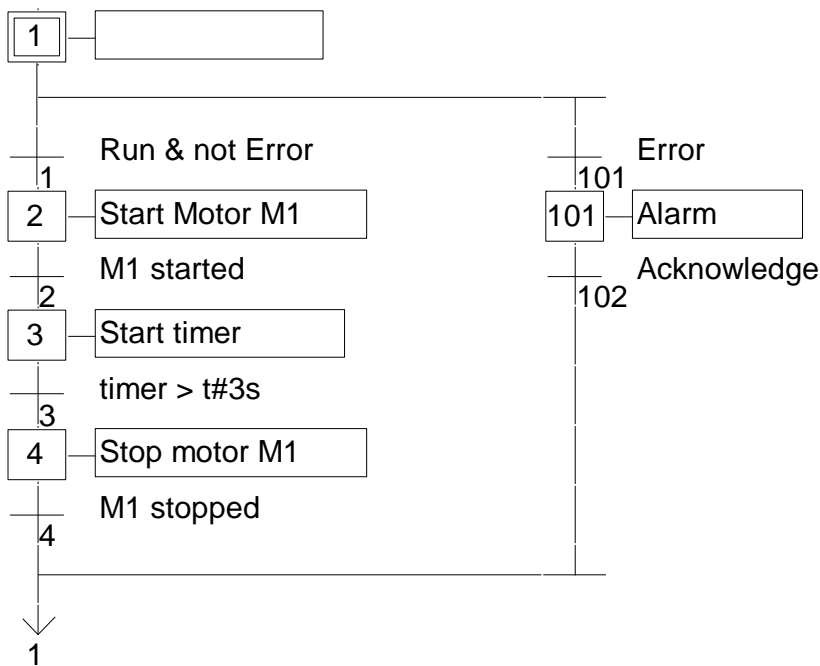
E.3.3.2 Double divergences

A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence. Double divergences and convergences are represented by double horizontal lines.

Double divergence

Double convergence

Example of double divergence and convergence:

(* SFC program with double divergence and convergence *)

| 1 | Initialize |

Run
1

| 2 | Process1 | | 101 | Process2 |

End of Process 1   End of Process 2
2   101

| 3 | Wait for process 2 | | 102 | Wait for process 2 |

true
3

1

E.3.4 Macro steps

A macro step is a unique representation of a unique group of steps and transitions. The body of the macro step is described separately, elsewhere in the same SFC program. It appears as a single symbol in the main SFC chart. This is the symbol used for a macro step:

Reference number

Comment

102 | Process A
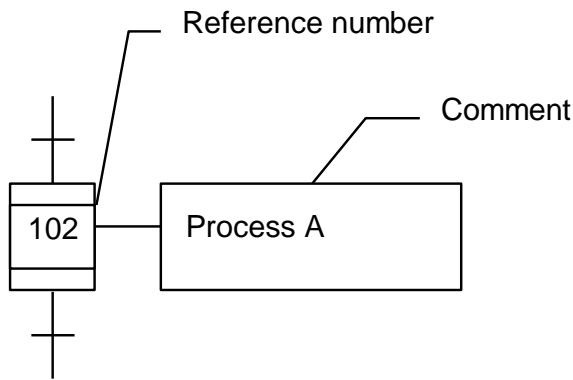
The reference number written in the macro step symbol is the reference number of the first step in the body of the macro step. The macro step body must begin with a **beginning step** and terminate with an **ending step**. The chart must be self-contained. A beginning step has no upper link (no backward transition). An ending step has no lower link (no forward transition). A macro step symbol may be put in the body of another macro step.

Warning: Because macro step is a **unique** set of steps and transitions, the same macro step cannot be used more than once in an SFC program.

Example of macro step:
(* SFC program with macro step *)
(* Main chart *) (* Body of the macro step *)
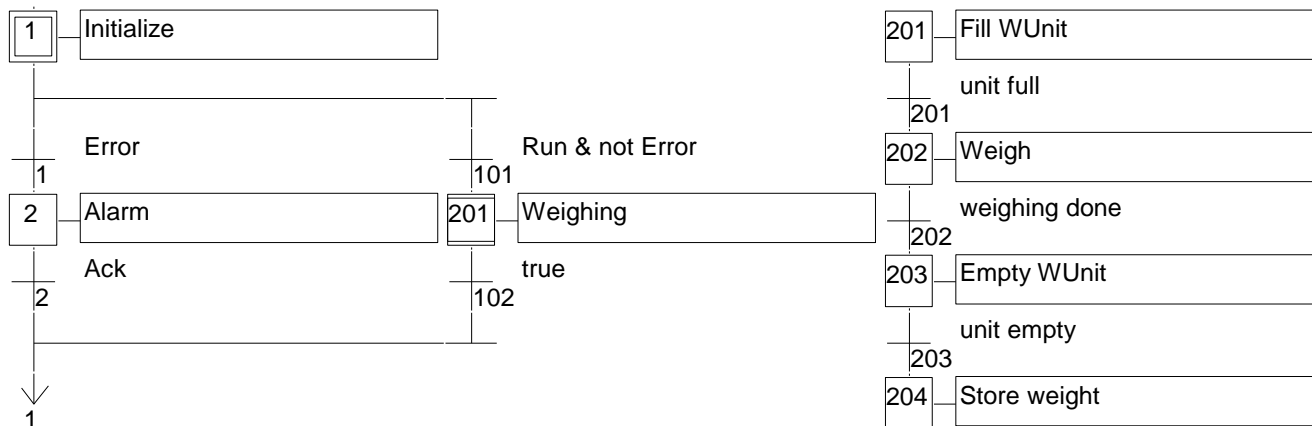
E.3.5 Actions within the steps

The **level 2** of an SFC step is the detailed description of the **actions** executed **during the step activity**. This description is made by using **SFC literal features**, and other languages such as Structured Text (**ST**). The basic types of actions are:
- Boolean actions
- Pulse actions programmed in ST
- Non-stored actions programmed in ST
- SFC actions

Several actions (with same or different types) can be described in the same step. The special features that enable the use of any of the other languages are:
- Calling sub-programs
- Instruction List (IL) language convention

E.3.5.1 Boolean actions

Boolean actions assign a boolean variable with the activity of the step. The boolean variable can be an output or an internal. It is assigned each time the step activity starts or stops. This is the syntax of the basic boolean actions:

<boolean_variable> (N) ; assigns the step activity signal to the variable
<boolean_variable> ; same effect (N attribute is optional)
**/ <boolean_variable> ;** assigns the negation of the step activity signal to the variable

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

<boolean_variable> (S) ; sets the variable to TRUE when the step activity signal becomes TRUE
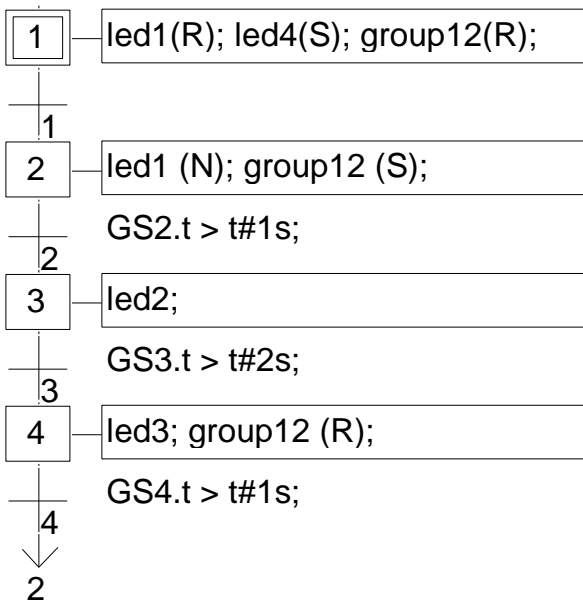**<boolean_variable> (R) ;** resets the variable to FALSE when the step activity signal becomes TRUE

The boolean variable must be an OUTPUT or an INTERNAL. The following SFC programming leads to the following behaviour:
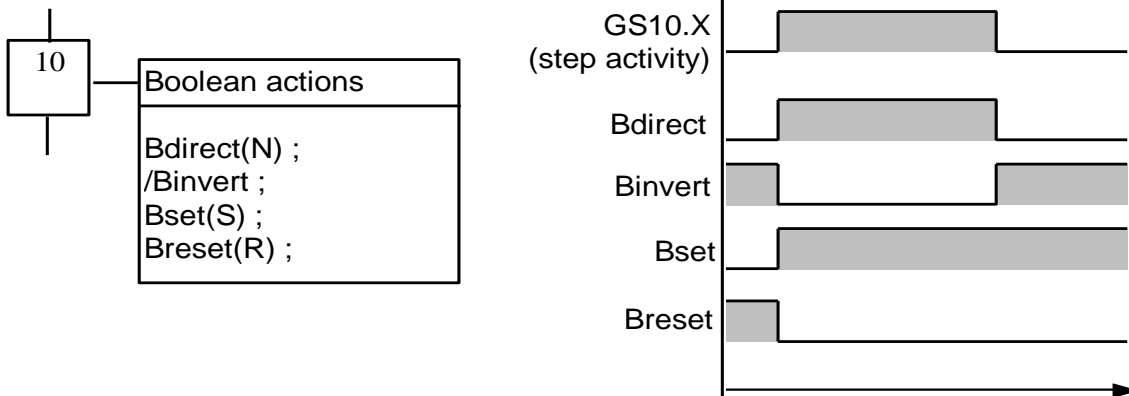
Example of boolean actions:

(* SFC program using BOOLEAN actions *)

```
┌───┐    ┌──────────────────────────────────────┐
│┌─┐│────│led1(R); led4(S); group12(R);         │
│└─┘│    └──────────────────────────────────────┘
└───┘
  │
──┼── 1
  │
┌───┐    ┌──────────────────────────────────────┐
│ 2 │────│led1 (N); group12 (S);                │
└───┘    └──────────────────────────────────────┘
  │         GS2.t > t#1s;
──┼──
  │ 2
┌───┐    ┌──────────────────────────────────────┐
│ 3 │────│led2;                                 │
└───┘    └──────────────────────────────────────┘
  │         GS3.t > t#2s;
──┼──
  │ 3
┌───┐    ┌──────────────────────────────────────┐
│ 4 │────│led3; group12 (R);                    │
└───┘    └──────────────────────────────────────┘
  │         GS4.t > t#1s;
──┼──
  │ 4
  ▼
  2
```
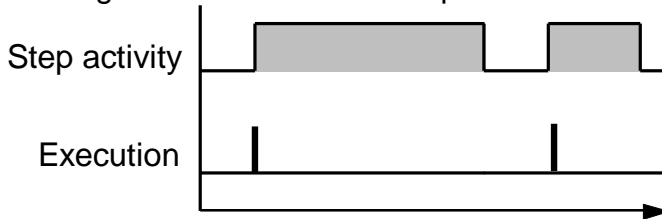
E.3.5.2 Pulse actions

A pulse action is a list of ST or IL instructions, which are executed only **once** at the **activation** of the step. Instructions are written according to the following SFC syntax:
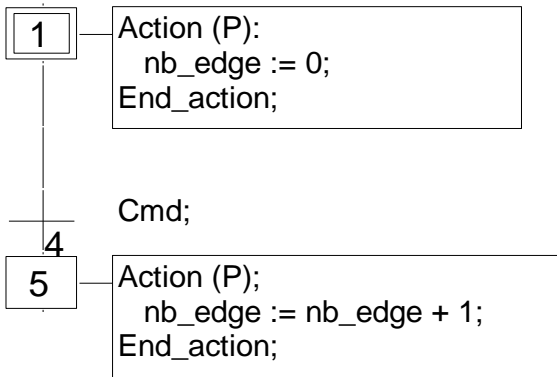
```
ACTION (P) :
 (* ST statements *)
END_ACTION ;
```

The following shows the results of a pulse action:

```
Step activity │   ┌───────────────┐   ┌──────┐
              │___│               │___│      │___
              │
    Execution │   │                   │
              │___│_____│_____
                                               →
```

Example of pulse action:

```
┌───┐    ┌──────────────────────────────────┐
│┌─┐│────│Action (P):                       │
│└─┘│    │   nb_edge := 0;                  │
└───┘    │End_action;                       │
  │      └──────────────────────────────────┘
  │
  │
  │         Cmd;
──┼──
  │ 4
┌───┐    ┌──────────────────────────────────┐
│ 5 │────│Action (P);                       │
└───┘    │   nb_edge := nb_edge + 1;        │
         │End_action;                       │
         └──────────────────────────────────┘
```
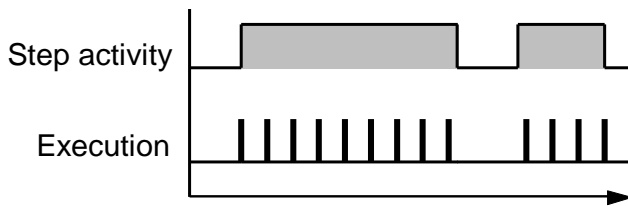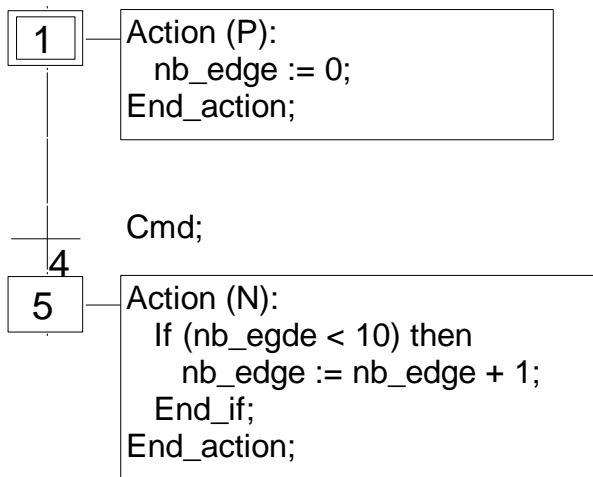
E.3.5.3 Non-stored actions

A non-stored (normal) action is a list of ST or IL instructions which are executed **at each cycle** during the whole **active** period of the step. Instructions are written according to the following SFC syntax:

```
ACTION (N) :
 (* ST statements *)
END_ACTION ;
```

The following is the results of a non-stored action:

Example of non-stored action:


```
1 ─ Action (P):
      nb_edge := 0;
    End_action;


    Cmd;
4

5 ─ Action (N):
      If (nb_egde < 10) then
        nb_edge := nb_edge + 1;
      End_if;
    End_action;
```
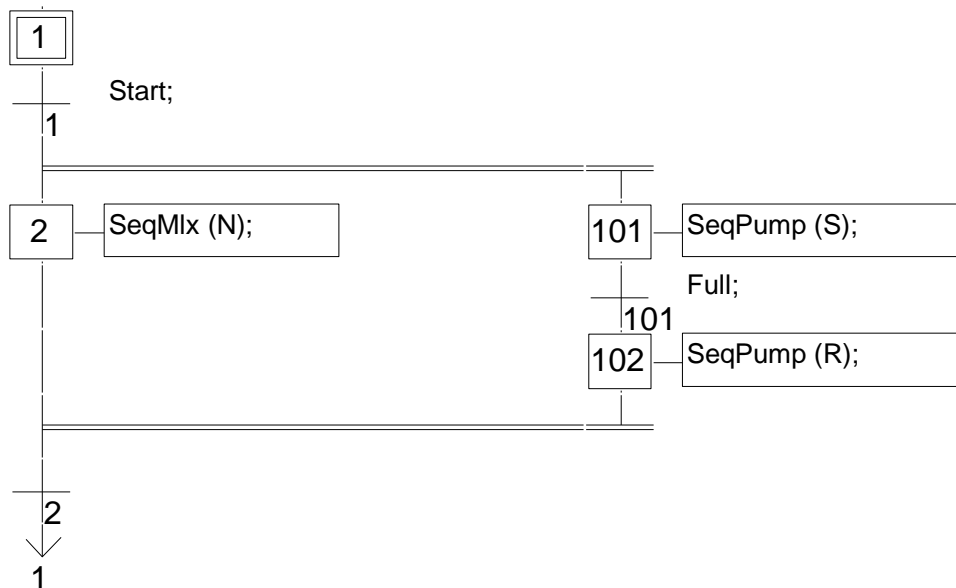
E.3.5.4 SFC actions

An SFC action is a child SFC sequence, started or killed according to the change of the step activity signal. An SFC action can have the **N** (Non stored), **S** (Set), or **R** (Reset) qualifier. This is the syntax of the basic SFC actions:

<child_prog> (N); starts the child sequence when the step becomes active, and kills the child sequence when the step becomes inactive
<child_prog> ; same effect (N attribute is optional)
<child_prog> (S); starts the child sequence when the step becomes active. Nothing is done when the step becomes inactive
<child_prog> (R); kills the child sequence when the step becomes active. Nothing is done when the step becomes inactive

The SFC sequence specified as an action must be a **child SFC program** of the program currently being edited. Note that using the **S** (Set) or **R** (Reset) qualifiers for an SFC action has exactly the same effect as the **GSTART** and **GKILL** statements, programmed in an **ST** pulse action.

Below is an example of an SFC action. The main SFC program is named **Father**. It has two SFC children, called **SeqMlx** and **SeqPump**. The SFC programming of the father SFC program is:

(* SFC program using SFC actions *)

E.3.5.5 Calling function and function blocks from an action

Sub-programs, functions or function blocks (written in ST, IL, LD or FBD language) or "C" functions and "C" function blocks, can be directly called from an SFC action block, based on the following syntax:

For sub-programs, functions and "C" functions:
```
ACTION (P) :
 result := sub_program ( ) ;
END_ACTION;
```

or

```
ACTION (N) :
 result := sub_program ( ) ;
END_ACTION;
```

For function blocks in "C" or in ST, IL, LD, FBD:
```
ACTION (P) :
 Fbinst(in1, in2);
 result1 := Fbinst.out1;
 result2 := Fbinst.out2;
```
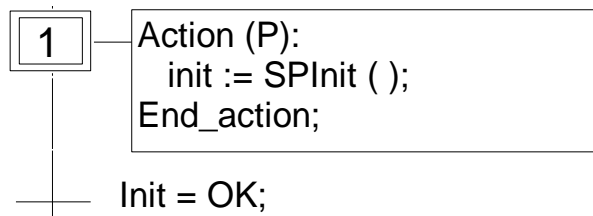
```
        END_ACTION;

or

        ACTION (N) :
         Fbinst(in1, in2);
         result1 := Fbinst.out1;
         result2 := Fbinst.out2;
        END_ACTION;
```

Detailed syntax can be found in the ST language section.
Example of a sub-program call in action blocks:

(* SFC program with a sub-program call in an action block *)

```
 ┌───┐   ┌──────────────────────────────┐
 │ 1 │───│Action (P):                   │
 └───┘   │    init := SPInit ( );       │
   │     │End_action;                   │
   │     └──────────────────────────────┘
   │
 ──┼──      Init = OK;
```
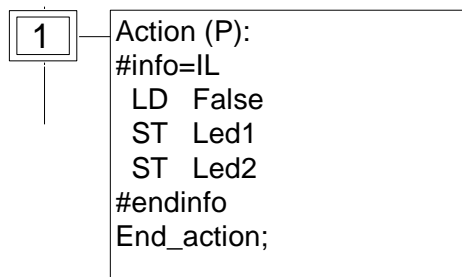
E.3.5.6 IL convention

Instruction List (IL) programming may be directly entered in an SFC action block, based on the
    following syntax:

```
        ACTION (P) :               (* or N *)
        #info=IL
            <instruction>
            <instruction>
            ....
        #endinfo
        END_ACTION;
```

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case
    sensitive**. Space or tab characters cannot be inserted into, after or before the keywords.
    Below is an example of an IL program in an action block:

(* SFC program with an IL sequence in an action block *)

```
 ┌───┐   ┌──────────────────────────────┐
 │ 1 │───│Action (P):                   │
 └───┘   │#info=IL                      │
   │     │ LD   False                   │
   │     │ ST   Led1                    │
   │     │ ST   Led2                    │
   │     │#endinfo                      │
   │     │End_action;                   │
         │                              │
         │                              │
         └──────────────────────────────┘
```

E.3.6 Conditions attached to transitions

At each transition, a **boolean expression** is attached that conditions the clearing of the transition. The condition is usually expressed with ST language or using the LD language (Quick LD editor). This is the **Level 2** of the transition. Other structures may, however, be used:

- ST language convention
- LD language convention
- IL language convention
- Calling function from a transition

<u>Warning</u>: When no expression is attached to the transition, the default condition is **TRUE**.

E.3.6.1 ST convention

The **Structured Text** (ST) language can be used to describe the **condition** attached to a transition. The complete expression must have **boolean** type and must be terminated by a **semicolon**, according to the following syntax:

< boolean_expression > ;

The expression may be a TRUE or FALSE constant expression, a single input or an internal boolean variable, or a combination of variables that leads to a boolean value. Below is an example of ST programming for transitions:
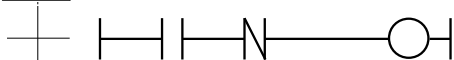
(* SFC program with ST programming for transitions *)

```
┌───┐
│ 1 │
└───┘
  │
──┼──   Run & not Error;
  │
```

E.3.6.2 LD convention

The **Ladder Diagram** (LD) language can be used to describe the **condition** attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transition value. Below is an example of LD programming for transitions:

```
┌─┐      Run   Error
│1│
└─┘
 ┼    ─┤ ├─┤N├────( )─
```

E.3.6.3 IL convention

Instruction List (IL) programming may be directly used to describe an SFC transition, according to the following syntax:

```
   #info=IL
    <instruction>
    <instruction>
    ....
   #endinfo
```
The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the transition:

   current result = 0   ➔   condition is **FALSE**
   current result <> 0   ➔   condition is **TRUE**

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords. Below is an example of IL programming for transitions:

(* SFC program with an IL program for transitions *)

```
┌─────┐
│  1  │
└─────┘
   :
   ┼── #info=IL
         LD   Run
         &N  Error
      #endinfo
```
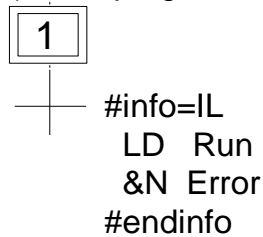
E.3.6.4 Calling functions from a transition

Any sub-program or a function (written in FBD, LD, ST or IL language), or a "C" function can be called to evaluate the condition attached to a transition, according to the following syntax:

   < sub_program > ( ) ;

The value returned by the sub-program or the function must be boolean and yields the resulting condition:

   return value = FALSE ➔   condition is **FALSE**
   return value = TRUE   ➔   condition is **TRUE**

Example of a sub-program called in a transition:

(* SFC program with sub-program call for transitions *)

```
┌─────┐
│  1  │
└─────┘
   ┼── EvalCond ( );
```

E.3.7 SFC dynamic rules

The **five** dynamic rules of the SFC language are:

▣ Initial situation
The initial situation is characterised by the **initial steps** which are, by definition, in the active
state at the beginning of the operation. **At least one** initial step must be present in each
SFC program.

▣ Clearing of a transition
A transition is either **enabled** or **disabled**. It is said to be enabled when all immediately
preceding steps linked to its corresponding transition symbol are **active**, otherwise it is
disabled. A transition cannot be **cleared** unless:
  - it is enabled, and
  - the associated transition condition is true.

▣ Changing of state of active steps
The clearing of a transition simultaneously leads to the active state of the immediately following
steps and to the inactive state of the immediately preceding steps.

▣ Simultaneous clearing of transitions
Double lines may be used to indicate transitions which have to be cleared simultaneously. If
such transitions are shown separately, the activity state of preceding steps (GSnnn.x) can
be used to express their conditions.

▣ Simultaneous activation and deactivation of a step
If, during operation, a step is simultaneously activated and deactivated, priority is given to the
activation.

E.3.8 SFC program hierarchy

The ISaGRAF system enables the description of the vertical structure of SFC programs. SFC
programs are organised in a **hierarchy tree**. Each SFC program can control (start, kill...)
other SFC programs. Such programs are called **children** of the SFC program which
controls them. SFC programs are linked together into a main **hierarchy tree**, using a
"**father - child**" relation:

FATHER program
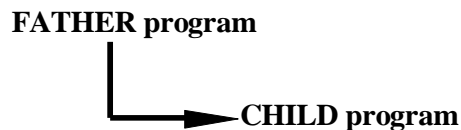
              → **CHILD program**

The basic rules implied by the hierarchy structure are:
- SFC programs which have no father are called "**main**" SFC programs
- Main SFC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can only be controlled by its father
- A program cannot control the children of one of its own children

The basic actions that a father SFC program can take to control its child program are:

Start (**GSTART**) Starts the child program: activates each of its initial steps. Children of this child program are not automatically started.

Kill (**GKILL**) Kills the child program by deactivating each of its active steps. All the children of the child program are also killed.

Freeze (**GFREEZE**) Suspends the execution of the program (deactivates actions of each of the active steps and suspend transition calculation), and memorises the status of the program steps so the program can be restarted. All the children of the child program are also frozen.

Restart (**GRST**) Restarts a frozen SFC program by reactivating all the suspended steps. Children of the program are not automatically restarted.

Get status (**GSTATUS**) Gets the current status (active, inactive or frozen) of a child program.

E.4 Flow Chart language

Flow Chart (**FC**) is a graphic language used to describe **sequential operations**. A Flow Chart diagram is composed of **Actions** and **Tests**. Between Actions and test are **oriented links** representing data flow. Multiple connection links are used to represents divergences and convergences. Actions and Tests can be described with ST, LD or IL languages. Functions and Function blocks of any language (except SFC) can be called from actions and tests. A Flow Chart program can call another Flow Chart program. The called FC program is a **sub-program** of the calling FC program.

E.4.1 FC components

Below are graphic components of the Flow Chart language:

▭ *Beginning of FC chart*

A "**begin**" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated. Below is the drawing of a "begin" symbol:
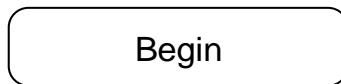
Begin

The "Begin" symbol always has a connection (on the bottom) to the other objects of the chart. A flow chart is not valid if no connection is drawn from "Begin" to another object.

▭ *Ending of FC chart*

An "**end**" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It is possible that no connection is drawn to the "End" symbol (always looping chart), but "End" symbol is still drawn anyway at the bottom of the chart. It represents the final state of the chart, when its execution has been completed. Below is the drawing of an "end" symbol:

End
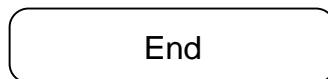
The "End" symbol generally has a connection (on the top) to the other objects of the chart. A flow chart may have no connection to the "End" object (always looping chart). The "End" object is still visible at the bottom of the chart in this case.

▭ *FC flow links*

A flow **link** is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Below is the drawing of a flow link:
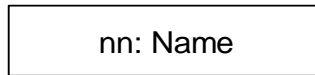
Two links cannot start from the same source connection point.

▭ *FC actions*

An **action** symbol represents actions to be performed. An action is identified by a number and a name. Below is the drawing of an "action" symbol:

```
┌─────────────────────┐
│      nn: Name       │
└─────────────────────┘
```

Two different objects of the same chart cannot have the same name or logical number. Programming language for an action can be ST, LD or IL. An action is always connected with links, one arriving to it, one starting from it.

◘ *FC conditions*

A **condition** represents a boolean **test**. A condition is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. Below are the possible drawings for a condition symbol:
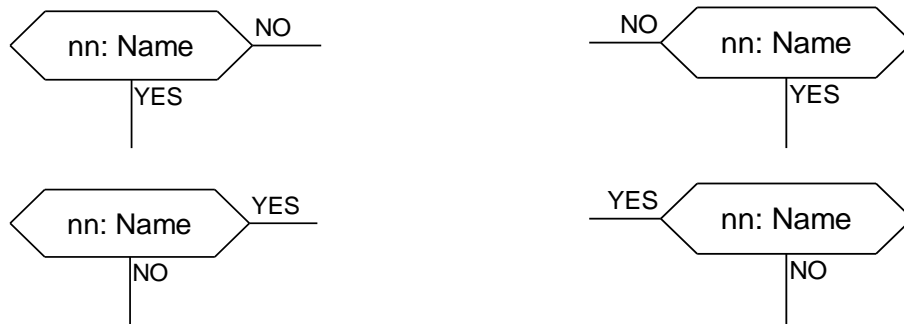
Two different objects of the same chart cannot have the same name or logical number. The programming of a test is either
- an expression in ST, or
- a single rung in LD, with no symbol attached to the unique coil, or
- several instructions in IL. The IL register (or current result) is used to evaluate the condition.

When programmed in ST text, the expression may optionally be followed by a semicolon. When programmed in LD, the unique coil represents the condition value. A condition equal to:
- 0 or FALSE directs the flow to NO
- 1 or TRUE directs the flow to YES

A test is always connected with an arriving link, and both forward connections must be defined.

◘ *FC sub-program*

The system enables the description of the vertical structure of FC programs. FC programs are organised in a **hierarchy tree**. Each FC program can call other FC programs. Such a program is called a **child program** of the FC program which calls them. FC programs which call FC sub-programs are called **father program**. FC programs are linked together into a main hierarchy tree, using a "father - child" relation:

FATHER program

```
     │
     │
     └──────────▶ CHILD program
```
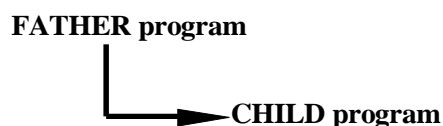
A **sub-program** symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete. A Flow Chart sub-program is identified by a number and a name, as other programs, functions or function blocks. Below is the drawing of a "sub-program call" symbol:

nn: SpName

Two different objects of the same chart cannot have the same logical number. The basic rules implied by the FC hierarchy structure are:
- FC programs which have no father are called main FC programs.
- Main FC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can be called only by its father
- A program cannot call the children of one of its own children

The same sub-program may appear several times in the father chart. A Flow Chart sub-program call represents the complete execution of the sub chart. The father chart execution is suspended during the child chart is performed. The sub-program calling blocks must follow the same connection rules as the ones defined for action.

▭ *FC I/O specific action*

An **I/O specific action** symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. Below is the drawing of an "I/O specific action" symbol:

nn: Name
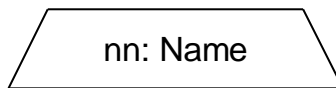
I/O specific blocks have exactly the same behaviour as standard actions. This covers their properties, ST, LD or IL programming, and connection rules.

▭ *FC connectors*

Connectors are used to represent a link between two points of the diagram without drawing it. A connector is represented as a circle and is connected to the source of the flow. The drawing of the connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). Below is the standard drawing of a connector:

nn: Name

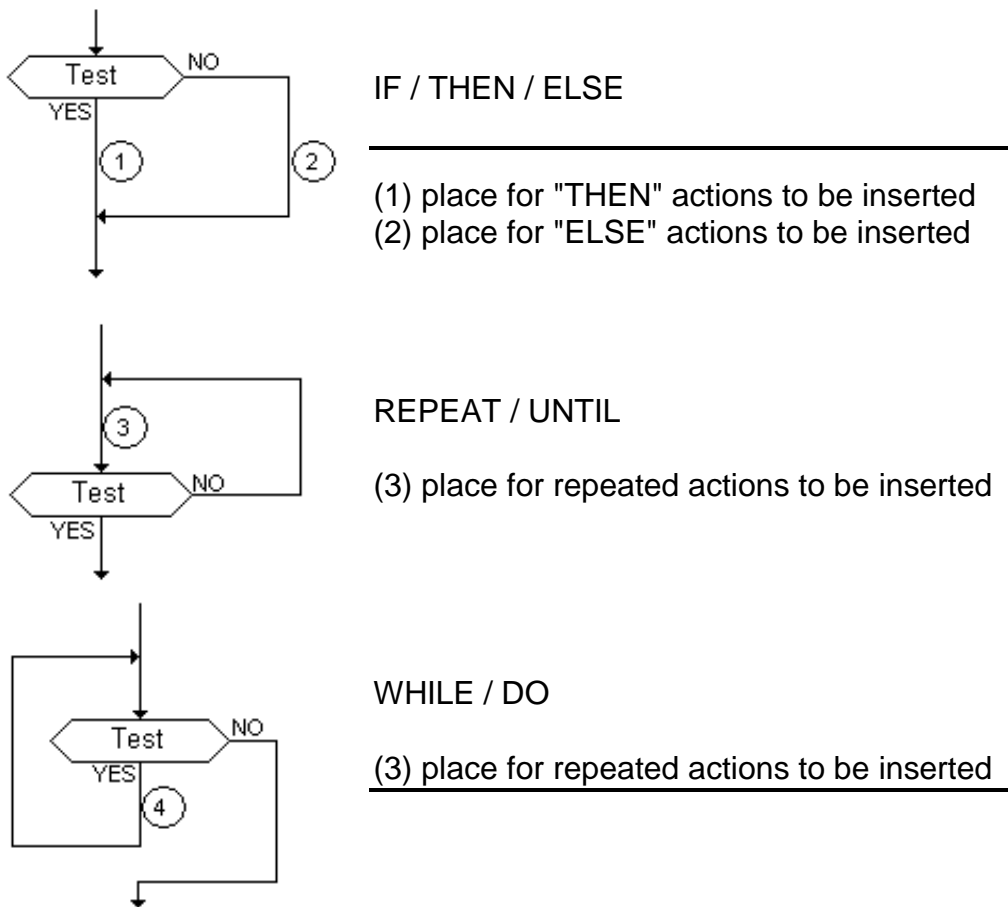A connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

▭ *FC comments*

A **comment** block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on an unused space of the Flow Chart document window, and is used to document the program. Below is the drawing of a "comment" symbol:

comment text can
be on several lines...
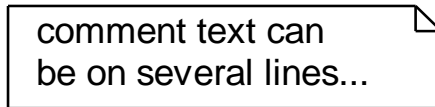
E.4.2 FC complex structures

This section shows **complex structure** examples that can be defined in a Flow Chart diagram. Such structures are combinations of basic objects linked together.

IF / THEN / ELSE

(1) place for "THEN" actions to be inserted
(2) place for "ELSE" actions to be inserted

REPEAT / UNTIL

(3) place for repeated actions to be inserted

WHILE / DO

(3) place for repeated actions to be inserted

E.4.3 FC dynamic behaviour

The **execution** of a Flow Chart diagram can be explained as follows:

- The Begin symbol takes one target cycle
- The End symbol takes one target cycle and ends the execution of the chart. After this symbol is reached, no more actions of the chart are executed.

- The flow is broken each time an item (action, decision) is encountered that has already been reached in the same cycle. In such a case the flow will continue on the next cycle.

Note: Contrary to SFC, an action is not a stable state. There is no repetition of instructions while the action symbol is highlighted.

E.4.4 FC checking

Apart of attached ST, LD or IL programming, some other **syntactic rules** apply to flow chart itself. Below is the list of main rules:
- All "connection" points of all symbols must be wired. (connection to "End" symbol may be omitted)
- All symbols must be linked together (no isolated part should appear)
- All connectors should have valid destination

Other minor syntax errors can be reported:
- Empty actions (no programming) are considered as steps during run time scheduling
- Empty tests (no programming) are considered as "always true"

E.5 FBD language

The **Functional Block Diagram** (FBD) is a graphic language. It allows the programmer to build complex procedures by taking existing **functions** from the ISaGRAF library and **wiring** them together in the graphic diagram area.

E.5.1 FBD diagram main format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of **elementary function blocks**. Input and output variables are connected to blocks by **connection lines**. An output of a function block may also be connected to an input of another block.
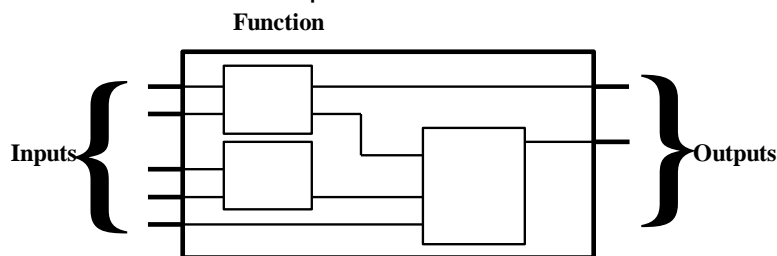
An entire function operated by an FBD program is built with standard **elementary** function blocks from the ISaGRAF library. Each function block has a fixed number of **input connection points** and a fixed number of **output connection points**. A function block is represented by a single **rectangle**. The inputs are connected on its **left** border. The outputs are connected on its **right** border. An elementary function block performs a single **function** between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well-defined **type**.
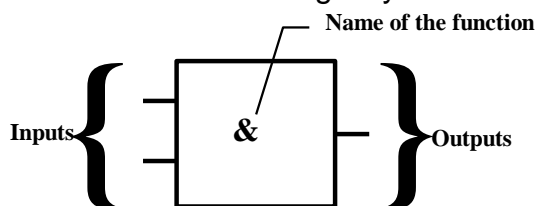
Input variables of an FBD program must be connected to input connection points of function blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a **constant** expression, any **internal** or **input** variable, or an **output** variable.

Output variables of an FBD program must be connected to output connection points of function blocks. The type of each variable must be the same as the type expected for the associated block output. An Output for FBD diagram can be any **internal** or **output** variable, or the name of the program (for **sub-programs** only). When an output is the name of the currently edited sub-program, it represents the assignment of the return value for the sub-program (returned to the calling program).

Input and output variables, inputs and outputs of the function blocks are wired together with **connection lines**. Single lines may be used to **connect** two logical points of the diagram:
- An input variable and an input of a function block
- An output of a function block and an input of another block
- An output of a function block and an output variable

The connection is **oriented**, meaning that the line carries associated data from the left extremity to the right extremity. The left and right extremities of the connection line must be of the **same type**.

Multiple right connection can be used to broadcast an information from its left extremity to each of its right extremities. All the extremities of the connection must be of the same type.

E.5.2 RETURN statement

The "**<RETURN>**" keyword may occur as a diagram output. It must be connected to a boolean output connection point of a function block. The RETURN statement represents a **conditional end** of the program: if the output of the box connected to the statement has the boolean value **TRUE**, the end (remaining part) of the diagram is not executed.

(* Example of an FBD program using RETURN statement *)
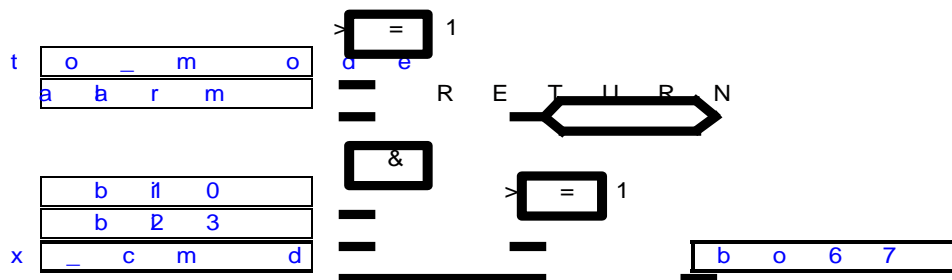
(* ST equivalence: *)
If auto_mode OR alarm Then
   Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;

E.5.3 Jumps and labels

Labels and jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notations are used:

>>LAB   jump to a label (label name is "LAB")
LAB: definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the boolean state **TRUE**, the execution of the program directly jumps after the corresponding label symbol.

(* Example of an FBD program using labels and jumps *)
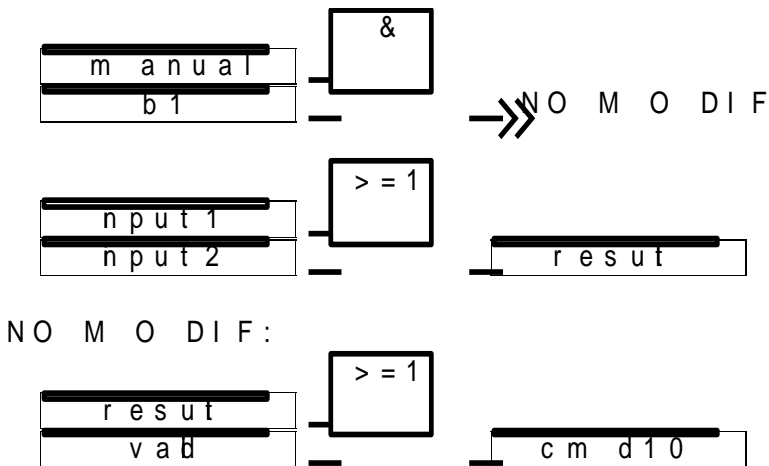
NO M O DIF:

(* IL Equivalence: *)
```
        ld        manual
        and       b1
        jmpc      NOMODIF
        ld        input1
        or        input2
        st        result
NOMODIF: ld   result
        or        valid
        st        cmd10
```
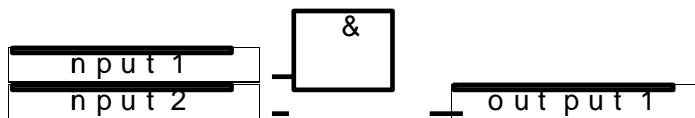
E.5.4 Boolean negation

A single connection line with its right extremity connected to an input of a function block can be terminated by a **boolean negation**. The negation is represented by a small circle. When a boolean negation is used, the left and right extremities of the connection line must have the **BOOLEAN** type.

(* Example of an FBD program using a boolean negation *)
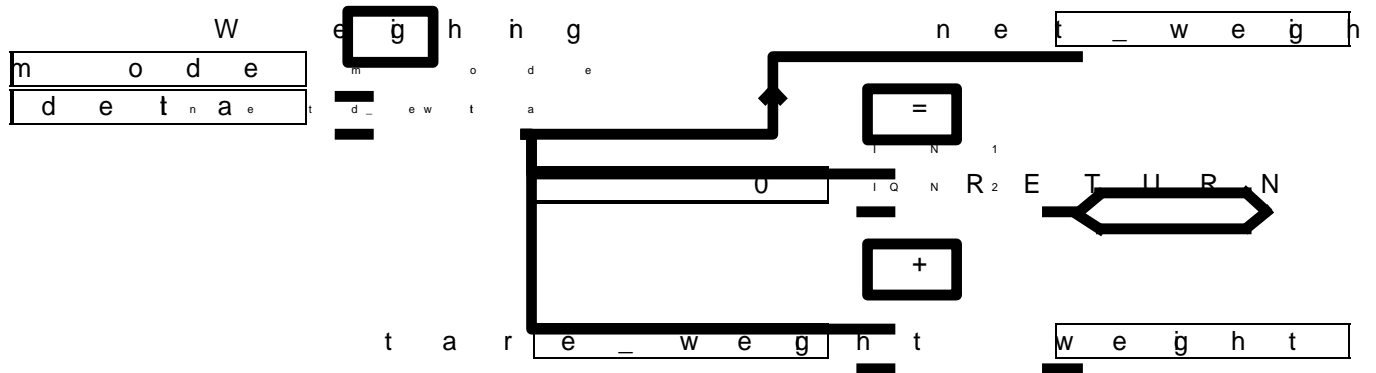

(* ST equivalence: *)
output1 := input1 AND NOT (input2);

E.5.5 Calling function or function blocks from the FBD

The FBD language enables the calling of sub-programs, functions or function blocks. A sub-program, or function or function block is represented by a function box. The names written in the boxes are the name of the sub-program or function or function blocks.

In case of a sub-program or a function, the return value is the only output of the function box.
A function block can have more than one output.

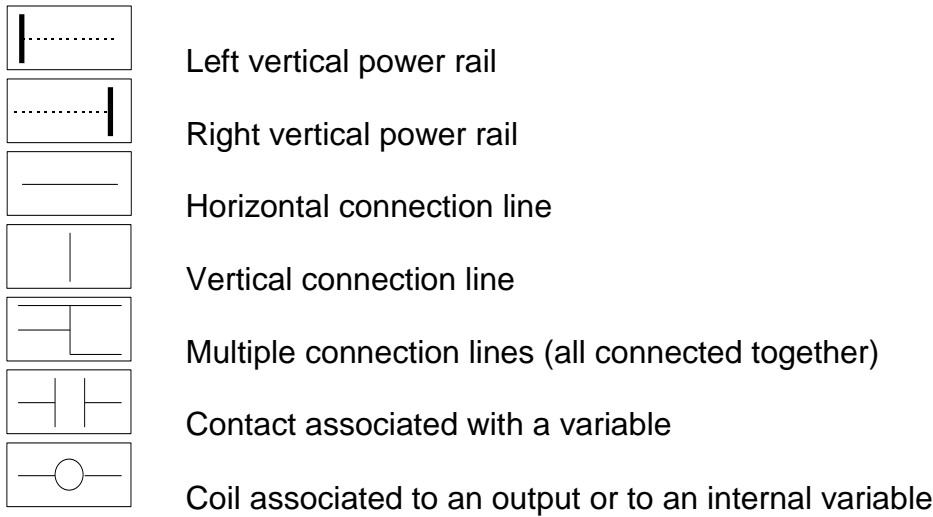(* Example of an FBD program using SUB PROGRAM block *)

(* ST Equivalence *)
net_weight := Weighing (mode, delta); (* call sub-program *)
If (net_weight = 0) Then Return; End_if;
weight := net_weight + tare_weight;

E.6 LD language

Ladder Diagram (LD) is a graphic representation of boolean equations, combining **contacts** (input arguments) with **coils** (output results). The LD language enables the description of tests and modifications of **boolean** data by placing **graphic symbols** into the program chart. LD graphic symbols are organized within the chart exactly as an electric contact diagram. LD diagrams are connected on the left side and on the right side to vertical **power rails**. These are basic graphic components of an LD diagram:

Left vertical power rail

Right vertical power rail

Horizontal connection line

Vertical connection line

Multiple connection lines (all connected together)

Contact associated with a variable
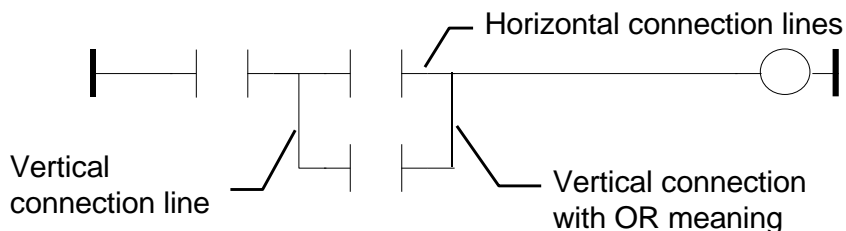
Coil associated to an output or to an internal variable

E.6.1 Power rails and connection lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.
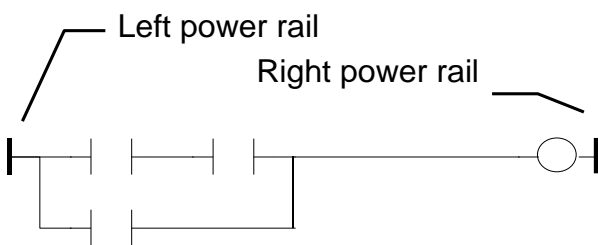
LD diagram graphic symbols are connected to power rails or to other symbols by **connection lines**. Connection lines are horizontal or vertical.

Each line segment has a boolean state **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

E.6.2 Multiple connection

The boolean state given to a single horizontal connection line is the same on the left and on the right extremities of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The boolean state of the extremities of a multiple connection follows logic rules.

A **multiple connection on the left** combines **more than one** horizontal lines connected on the **left** side of a vertical line, and **one** line connected on its **right** side. The boolean state of the right extremity is the **LOGICAL OR** between all the left extremities.

(* Example of multiple LEFT connection *)
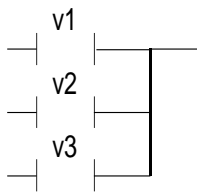
(* right extremity state is (v1 OR v2 OR v3) *)

A **multiple connection on the right** combines **one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of the left extremity is propagated into each of the right extremities.

(* Example of multiple RIGHT connection *)

(* ST equivalence: *)
output1 := input1;
output2 := input1;

A **multiple connection on the left and on the right** combines **more than one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of each of the right extremities is the **LOGICAL OR** between all the left extremities
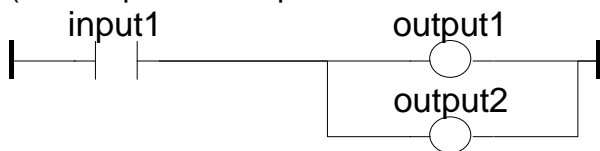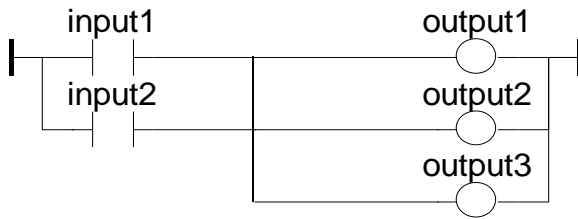
(* Example of multiple LEFT and RIGHT connection *)

(* ST Equivalence: *)
output1 := input1 OR input2;
output2 := input1 OR input2;
output3 := input1 OR input2;

E.6.3 Basic LD contacts and coils

There are several symbols available for input contacts:
- Direct contact
- Inverted contact
- Contacts with edge detection

There are several symbols available for output coils:
- Direct coil
- Inverted coil
- SET coil
- RESET coil
- Coils with edge detection

The name of the variable is written above any of these graphic symbols:

▭ **Direct contact**

A direct contact enables a **boolean operation** between a **connection line** state and a boolean **variable**.
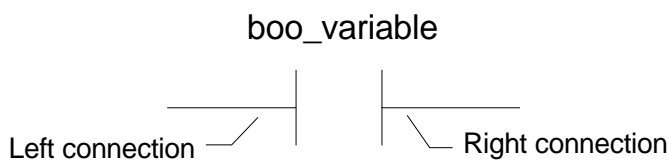
The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the value of the variable associated with the contact.

(* Example using DIRECT contacts *)
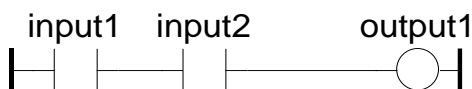
(* ST Equivalence: *)
output1 := input1 AND input2;

▭ **Inverted contact**

An inverted contact enables a **boolean operation** between a **connection line** state and the boolean negation of a boolean **variable**.
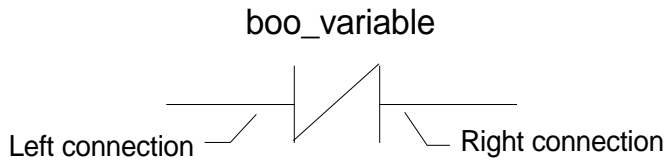
boo_variable

Left connection —/ \/ \— Right connection

The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the **boolean negation** of the value of the variable associated with the contact.

(* Example using INVERTED contacts *)

input1   input2   output1
├─/├──────/├───────────( )─┤
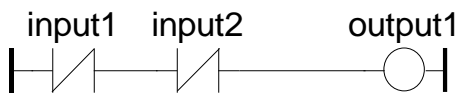
(* ST Equivalence: *)
output1 := NOT (input1) AND NOT (input2);

▬   *Contact with rising edge detection*

This contact (positive) enables a **boolean operation** between a **connection line** state and the rising edge of a boolean **variable**.
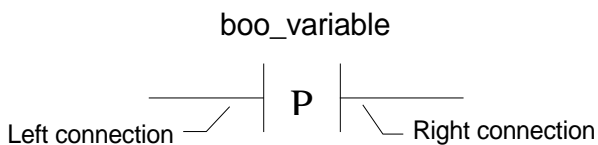
boo_variable

Left connection —/ | P | \— Right connection

The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **rises** from FALSE to TRUE. It is reset to FALSE in all other cases.

(* Example using RISING EDGE contacts *)

input1   input2   output1
├─┤├──────┤ P ├──────────( )─┤
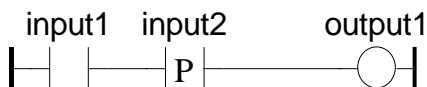
(* ST Equivalence: *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)

▬   *Contact with falling edge detection*

This contact (negative) enables a **boolean operation** between a **connection line** state and the falling edge of a boolean **variable**.
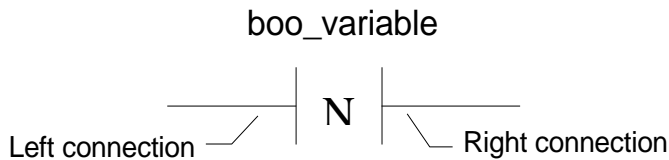
boo_variable

The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **falls** from TRUE to FALSE. It is reset to FALSE in all other cases.

(* Example using FALLING EDGE contacts *)
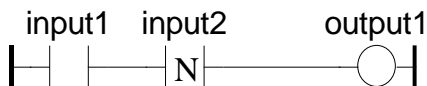
(* ST Equivalence: *)
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)

▭ *Direct coil*

Direct coils enable a **boolean output** of a **connection line** boolean state.
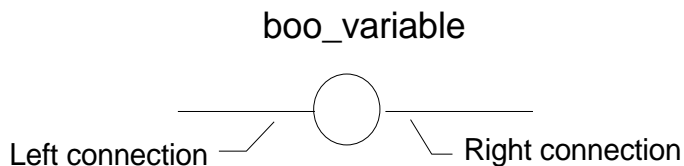
boo_variable

The associated variable is assigned with the boolean **state of the left connection**. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

(* Example using DIRECT coils *)
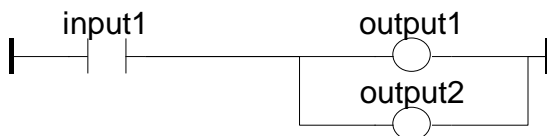
(* ST Equivalence: *)
output1 := input1;
output2 := input1;

▭ *Inverted coil*

Inverted coils enable a **boolean output** according to the boolean **negation** of a **connection line** state.

boo_variable

Left connection — — Right connection
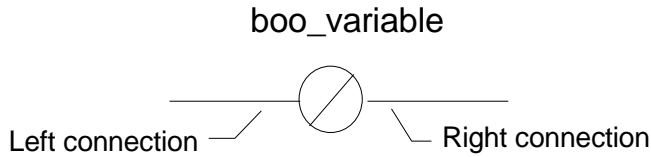
The associated variable is assigned with the boolean **negation** of the **state of the left connection**. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

(* Example using INVERTED coils *)
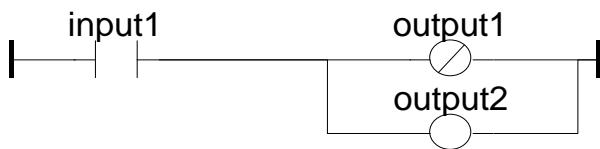
(* ST Equivalence: *)
output1 := NOT (input1);
output2 := input1;

▭ *SET coil*

"Set" coils enable a **boolean output** of a **connection line** boolean state.
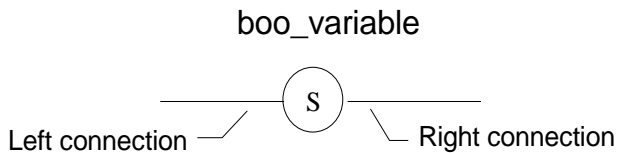
boo_variable

Left connection — — Right connection

The associated variable is **SET TO TRUE** when the boolean **state of the left connection** becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)
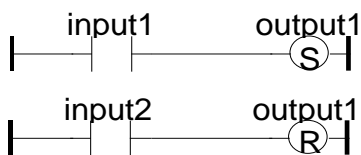
(* ST Equivalence: *)

```
IF input1 THEN
   output1 := TRUE;
END_IF;
IF input2 THEN
   output1 := FALSE;
END_IF;
```

▭ *RESET coil*

"Reset" coils enable **boolean output** of a **connection line** boolean state.

boo_variable

Left connection   Right connection
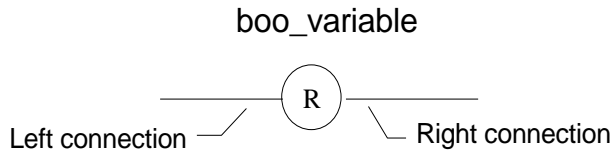
The associated variable is **RESET TO FALSE** when the boolean **state of the left connection**
   becomes **TRUE**. The output variable keeps this value until an inverse order is made by a
   "SET" coil. The state of the left connection is propagated into the right connection. Right
   connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)
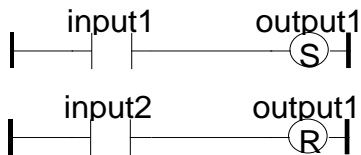

```
(* ST Equivalence: *)
IF input1 THEN
   output1 := TRUE;
END_IF;
IF input2 THEN
   output1 := FALSE;
END_IF;
```

▭ *Coil with rising edge detection*

"Positive" coils enable **boolean output** of a **connection line** boolean state. This type of coils is
   only available using the Quick ladder editor.

boo_variable

Left connection   Right connection
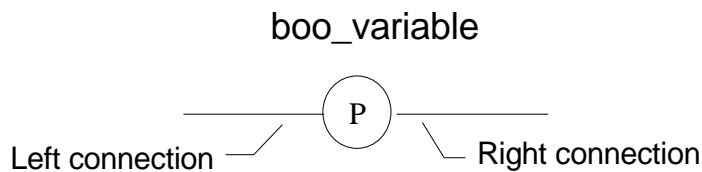
The associated variable is set to **TRUE** when the boolean **state of the left connection** rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)

```
      input1         output1
├─────┤ ├───┤ ├──────(P)─┤
```

(* ST Equivalence: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)

- *Coil with falling edge detection*

"Negative" coils enable **boolean output** of a **connection line** boolean state. This type of coils is only available using the Quick ladder editor.

```
              boo_variable
                  (N)
Left connection        Right connection
```
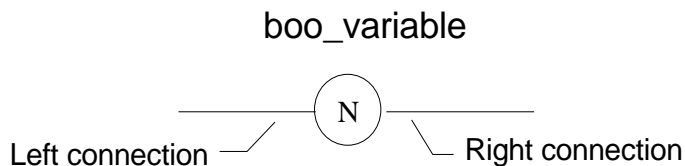
The associated variable is set to **TRUE** when the boolean **state of the left connection** falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)

```
      input1         output1
├─────┤ ├───┤ ├──────(N)─┤
```
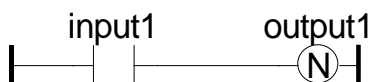
(* ST Equivalence: *)
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)

E.6.4 RETURN statement

The **RETURN** label can be used as an output to represent a conditional end of the program. No connection can be put on the right of a RETURN symbol.

```
───────────⟨ RETURN ⟩
```

If the **left connection** line has the **TRUE** boolean state, the program ends without executing the equations entered on the following lines of the diagram.

Note: When the LD program is a sub-program, its name has to be associated with an output coil to set the return value (returned to the calling program).

(* Example using RETURN symbol *)

```
manual_mode
├──┤/├──────────⟨ RETURN ⟩

   input1   input2      result
├──┤ ├──┬──┤ ├───┤ ├──────( )──┤
   │ input3 │
   └──┤ ├───┘
```
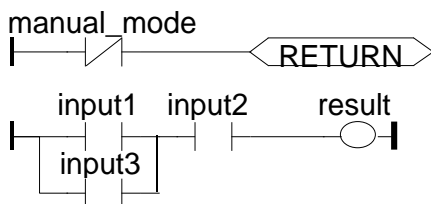
(* ST Equivalence: *)
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;

E.6.5 Jumps and labels

Labels, conditional and unconditional JUMPS symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB   jump to label named "LAB"
LAB: definition of the label named "LAB"

If the **connection on the left** of the jump symbol has the **TRUE** boolean state, the program execution is driven after the label symbol.

(* Example using JUMP and LABEL symbols *)

```
manual_mode
 ├──┤/├──────────────≫OTHER
    input1         result
 ├──┤ ├──────────────( )──┤
                     ≫END
OTHER:
    input2         result
 ├──┤ ├──────────────( )──┤
END:
```
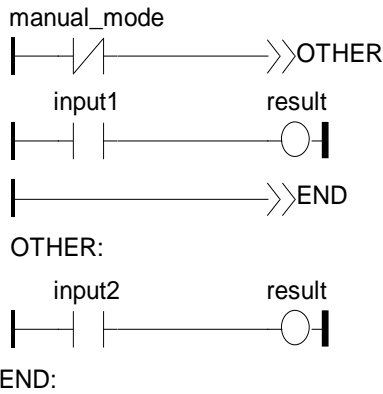
(* IL Equivalence: *)
```
     ldn      manual_mode
     jmpc     other
     ld       input1
     st       result
     jmp      END
OTHER:   ld   input2
     st       result
END:         (* end of program *)
```

E.6.6 Blocks in LD

Using the Quick LD editor, you connect function boxes to boolean lines. A function can actually
be an operator, a function block or a function. As all blocks do not have always a boolean
input and/or a boolean output, inserting blocks in an LD diagram leads to the addition of new
parameters EN, ENO to the block interface. The EN, ENO parameters are not added if you
use the FBD/LD editor as you can connect the variable with the required type.

▭   *The "EN" input*

On some operators, functions or function blocks, the first input does not have boolean data type.
As the first input must always be connected to the rung, another input is automatically
inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE.
Below is the example of a comparison operator, and the equivalent code expressed in ST:

```
                    >
              en         q
  value1
              in1
  value2
              in2
```

```
IF rung_state THEN
  q := (value1 > value 2);
ELSE
  q := FALSE;
END_IF;
(* continue rung with q state *)
```
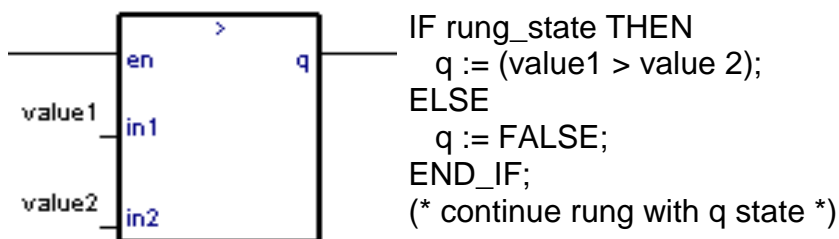
▭   *The "ENO" output*

On some operators, functions or function blocks, the first output does not have boolean data
type. As the first output must always be connected to the rung, another output is
automatically inserted at the first position, called "**ENO**". The **ENO** output always takes the

same state as the first input of the block. Below is an example with AVERAGE function block, and the equivalent code expressed in ST:
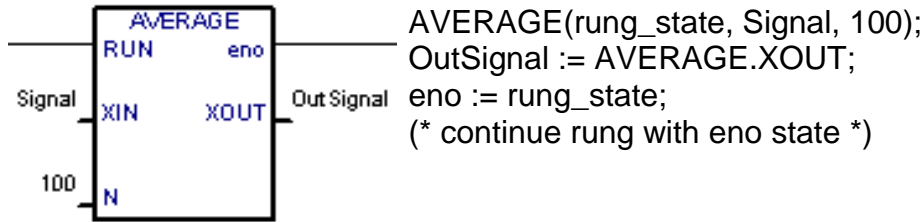
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)

▭ *The "EN" and "ENO" parameters*

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in ST:
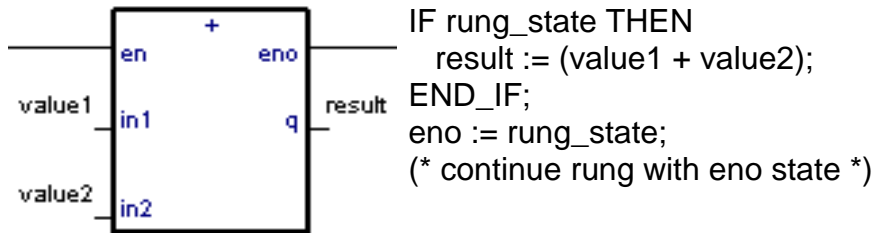
IF rung_state THEN
    result := (value1 + value2);
END_IF;
eno := rung_state;
(* continue rung with eno state *)

E.7 ST language

ST (**Structured Text**) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST is the default language for the description of the actions within the steps and conditions attached to the transitions of the **SFC** language.

E.7.1 ST main syntax

An ST program is a list of ST **statements**. Each statement ends with a semi-colon (**";"**) separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with **inactive separators** (space character, end of line or tab stops) or by **active separators**, which have a well defined significance (for example, the **">"** separator indicates a "greater than" comparison. Comments may be freely inserted into the text. A comment must begin with **"(*"** and ends with **"*)"**. Each statement terminates with a semi-colon (**";"**) separator. These are basic types of ST statements:

- **assignment** statement (variable := expression;)
- sub-program or function call
- function block call
- **selection** statements (IF, THEN, ELSE, CASE...)
- **iteration** statements (FOR, WHILE, REPEAT...)
- **control** statements (RETURN, EXIT...)
- special statements for links with other languages such as **SFC**

Inactive separators may be freely entered between active separators, constant expressions and identifiers. ST inactive separators are: **Space** (blank) character, **Tabs** and **End of line** character. Unlike line-formatted languages such as IL, end of lines may be entered anywhere in the program. The rules shown below should be followed when using inactive separators to increase ST program readability:

- Do not write more than one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase readability of lines or paragraphs

E.7.2 Expression and parentheses

ST expressions combine ST **operators** and variable or constant **operands**. For each single expression (combining operands with one ST operator), the **type** of the operands must be the same. This single expression has the same type as its operands, and can be used in a more complex expression. For example :

(boo_var1 AND boo_var2)	has BOO type
not (boo_var1)	has BOO type
(sin (3.14) + 0.72)	has REAL ANALOG type
(t#1s23 + 1.78)	is an invalid expression

Parentheses are used to isolate sub parts of the expression, and to explicitly order the priority of the operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default **priority** between ST operators. For example:

2 + 3 * 6   equals 2+18=20   because multiplication operator has a higher priority

(2+3) * 6   equals 5*6=30   priority is given by parenthesis

Warning: A maximum number of **8** levels of parentheses can be nested within an expression.

E.7.3 Function or function block calls

Standard ST function calls may be used for each of following objects:
- Sub-programs
- Library functions and function blocks written in IEC languages
- "C" functions and function blocks
- Type conversion functions

▭ *Calling sub-programs or functions*

Name: name of the called sub-program
                 or library function written in IEC language or in "C"
Meaning: calls a ST, IL, LD or FBD sub-program or function or a "C" function
                 and gets its return value
Syntax: <variable> := <subprog> (<par1>, ... <parN>);
Operands: The type of return value and calling parameters must follow
                 the interface defined for the sub-program.
Return value: value returned by the sub-program

Sub-program calls may be used in any expression. They also may be used in an SFC transition.

Example1: Sub-program call

(* Main ST program *)
(* gets an analog value and converts it into a limited time value *)
ana_timeprog := SPlimit (tprog_cmd);
appl_timer := tmr (ana_timeprog * 100);

(* Called FBD program named 'SPlimit' *)
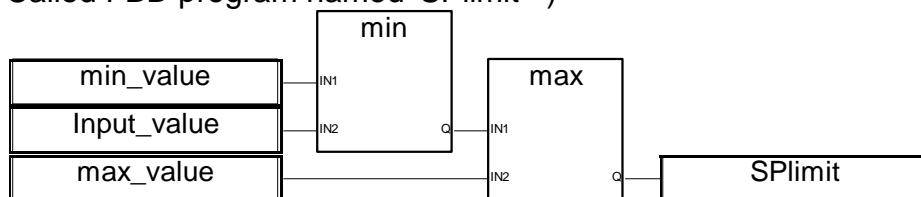
Example2: Function call

(* functions used in complex expressions: min, max, right, mlen and left are standard "C" functions *)

limited_value := min (16, max (0, input_value));
rol_msg := right (message, mlen (message) - 1) + left (message, 1);

□   *Calling function blocks*

Name:	name of the function block instance
Meaning:	calls a function block from the ISaGRAF library or from the user's library and accesses its return parameters
Syntax:	(* call of the function block *)
	<blockname> (<p1>, <p2> ...);
	(gets its return parameters *)
	<result> := <blockname>. <ret_param1>;
	...
	<result> := <blockname>. <ret_paramN>;
Operands:	parameters are expressions which match the type of the parameters specified for that function block
Return value:	See Syntax to get the return parameters.

Consult the ISaGRAF library to find the meaning and type of each function block parameter. The function block instance (name of the copy) must be declared in the dictionary

Example :

(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

E.7.4 ST specific boolean operators

The following boolean operators are specific to the ST language:
- REDGE   rising edge detection
- FEDGE   falling edge detection

Other standard boolean operators such as:
- NOT   boolean negation
- AND (&)   logical AND
- OR   logical OR
- XOR   logical exclusive OR
can be used. Their description is to be found in the section 'Standard operators, function blocks and functions'.

□   *"REDGE" operator*

Name:	REDGE
Meaning:	evaluates the rising edge of a complete boolean expression

Syntax:	<edge> := REDGE (<boo_expression>,<memo_variable>);
Operands:	first operand is any boolean variable or complex expression
	second operand is an internal boolean variable used to store the last state
	of the expression
Return value:	TRUE when the expression changes from FALSE to TRUE
	FALSE for all other cases

The rising edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. This operator can be used to describe the condition attached to an SFC transition.

<u>Warning</u>: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the REDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other REDGE evaluations.

Example:

(* ST program using REDGE operator *)

(* this program counts the rising edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If REDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of R_TRIG standard block. It has been kept for compatibility reasons.

- *"FEDGE" operator*

Name:	FEDGE
Meaning:	evaluates the falling edge of a boolean expression
Syntax:	<edge> := FEDGE (<boo_expression>, <memo_variable>);
Operands:	first operand is any boolean variable or complex expression
	second operand is an internal boolean variable used to store
	the last state of the expression
Return value:	TRUE when the expression changes from TRUE to FALSE
	FALSE for all other cases

The falling edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. The operator can be used to describe the condition attached to an SFC transition.

<u>Warning</u>: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the FEDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other FEDGE evaluations.

Example:

(* ST program using FEDGE operator *)

(* this program counts the falling edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If FEDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of F_TRIG standard block. It has been kept for compatibility reasons.

E.7.5 ST basic statements

The basic statements of the ST language are:
- Assignment
- RETURN statement
- IF-THEN-ELSIF-ELSE structure
- CASE statement
- WHILE iteration statement
- REPEAT iteration statement
- FOR iteration statement
- EXIT statement

▱ *Assignment*

Name: :=
Meaning: assigns a variable to an expression
Syntax: \<variable\> := \<any_expression\> ;
Operands: variable must be internal or output
              variable and expression must have the same type

The expression can be a call to a sub-program or a function from the ISaGRAF library

Example:

(* ST program with assignments *)

```
(* variable <<= variable *)
bo23 := bo10;

(* variable <<= expression *)
bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;

(* assignment with sub-program return value *)
rc := PSelect ( );

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

RETURN statement

Name: RETURN
Meaning: terminates the execution of the current program
Syntax: RETURN ;
Operands: (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

Example:

(* FBD specification of the program: programmable counter *)
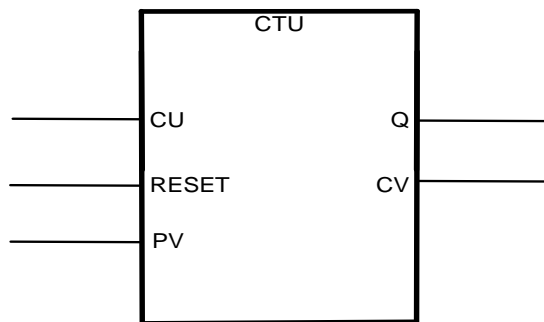
(* ST implementation of the program, using RETURN statement *)

```
If not (CU) then
    Q := false;
    CV := 0;
    RETURN;   (* terminates the program *)
end_if;

if R then
    CV := 0;
else
    if (CV < PV) then
     CV := CV + 1;
```

```
        end_if;
end_if;
Q := (CV >= PV);
```

- **IF-THEN-ELSIF-ELSE statement**

Name: IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF
Meaning: executes one of two lists of ST statements
            selection is made according to the value
            of a boolean expression
Syntax: IF <boolean_expression> THEN
               <statement> ;
               <statement> ;

               ...
            ELSIF <boolean_expression> THEN
               <statement> ;
               <statement> ;

               ...
            ELSE
               <statement> ;
               <statement> ;

               ...
            END_IF;

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no
    instruction is executed when the condition is FALSE.

Example:

(* ST program using IF statement *)

IF manual AND not (alarm) THEN
    level := manual_level;
    bx126 := bi12 OR bi45;
ELSIF over_mode THEN
    level := max_level;
ELSE
    level := (lv16 * 100) / scale;
END_IF;

(* IF structure without ELSE *)
If overflow THEN
alarm_level := true;
END_IF;

- *CASE statement*

Name: CASE ... OF ... ELSE ... END_CASE
Meaning: executes one of several lists of ST statements
          selection is made according to an integer expression
Syntax: CASE <integer_expression> OF
              <value> : <statements> ;
              <value> , <value> : <statements> ;

              ...
           ELSE
              <statements> ;
           END_CASE;

Case values must be integer constant expressions. Several values, separated by comas, can lead to the same list of statements. The ELSE statement is optional.

Example:

(* ST program using CASE statement *)

CASE error_code OF
    255:      err_msg := 'Division by zero';
     fatal_error := TRUE;
    1:        err_msg := 'Overflow';
    2, 3:   err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
END_CASE;

- *WHILE statement*

Name: WHILE ... DO ... END_WHILE
Meaning: iteration structure for a group of ST statements
          the "continue" condition is evaluated BEFORE any iteration
Syntax: WHILE <boolean_expression> DO
              <statement> ;
              <statement> ;

              ...
           END_WHILE ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

Example:

(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)

(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END_WHILE;

□   *REPEAT statement*

Name: REPEAT ... UNTIL ... END_REPEAT
Meaning: iteration structure for a group of ST statements
            the "continue" condition is evaluated AFTER any iteration
Syntax: REPEAT
                <statement> ;
                <statement> ;

                ...
            UNTIL <boolean_condition>
            END_REPEAT ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed
    during REPEAT iterations. The change of state of an input variable cannot be used to
    describe the ending condition of a REPEAT statement.

Example:

(* ST program using REPEAT statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
    REPEAT
     string := string + ComGetChar ( );
     nbchar := nbchar + 1;
    UNTIL ((nbchar >= 16) OR NOT (ComIsReady ( )))
    END_REPEAT;
END_IF;

□   *FOR statement*

Name: FOR ... TO ... BY ... DO ... END_FOR
Meaning: executes a limited number of iterations,
            using an integer analog index variable

Syntax: FOR <index> := <mini> TO <maxi> BY <step> DO
                 <statement> ;
                 <statement> ;
              END_FOR;

Operands: index: internal analog variable increased at any loop
            mini: initial value for index (before first loop)
            maxi: maximum allowed value for index
            step: index increment at each loop

The [BY step] statement is optional. If not specified, the increment step is 1

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed
            during FOR iterations.

This is the "while" equivalent of a FOR statement:

    index := mini;
    while (index <= maxi) do
      <statement> ;
      <statement> ;
      index := index + step;
    end_while;

Example:

(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
      code := ascii (message, index);
      IF (code >= 48) & (code <= 57) THEN
            target := target + char (code);
      END_IF;
END_FOR;

◘   *EXIT statement*

Name: EXIT
Meaning: exit from a FOR, WHILE or REPEAT iteration statement
Syntax: EXIT;
Operands: (none)

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

Example:

(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

```
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
        code := ascii (message, index);
        IF (code  = searched_char) THEN
                found := YES;
                EXIT;
        END_IF;
END_FOR;
```

E.7.6 ST extensions

The following functions are extensions of the ST language:
- TSTART - TSTOP: timer control

The following statements and functions are available to control the execution of the SFC child programs. They may be used inside ACTION(): ... END_ACTION; blocks in SFC steps.

- GSTART   starts an SFC program
- GKILL   kills an SFC program
- GFREEZE freezes an SFC program
- GRST   restarts a frozen SFC program
- GSTATUS gets current status of an SFC program

Warning: These functions are not in the IEC 1131-3 norm.

Easy equivalent can be found for GSTART and GKILL using the following syntax in the SFC step:
    child_name(S); (* equivalent to GSTART(child_name); *)
    child_name(R); (* equivalent to GKILL(child_name); *)

The following fields can be used to access the status of an SFC step:
GSnnn.x   boolean value that represents the activity of the step
GSnnn.t   time elapsed since the last activation of the step
    ("**nnn**" is the reference number of the SFC step)

It is also possible to test the activity of a step declared in another SFC program, by using the following syntax:

GSnnn(progname).x

Warning: referencing a step of an other program, using this syntax is not in the IEC 1131-3 norm. An easy way to do the same respecting IEC rules, is to declare a global boolean variable in the dictionary which will represent the step activity to be tested (for example ref_step_X). Then you insert in the step, the variable with the N qualifier (ref_step_X(N);). Then in the program which wants to test the activity of the step, you use the variable.

Prog program   the other program which needs step activity of Prog program.
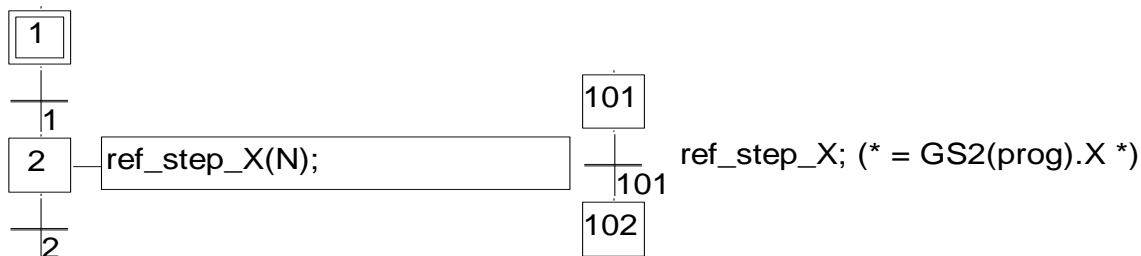
◦ *TSTART statement*

Name: TSTART
Meaning: starts the counting of a timer variable
                  timer value is not modified by the TSTART command, i.e. the counting starts
                  from the current value of the timer.
Syntax: TSTART (<timer_variable>);
Operands: any inactive timer variable
Return value: (none)

Example:

(* SFC program using TSTART and TSTOP statements *)

```
10 ─┐ ACTION(P):
    │   bo100 := TRUE; (* boolean output *)
    │   tm_ctrl := t#0s;
    │   TSTART(tm_ctrl);
    │ END_ACTION;

──┬── bi100 OR (tm_ctrl > time_out);

11 ─┐ ACTION(P):
    │   TSTOP(tm_ctrl);
    │   alarm := not(bi100);
    │ END_ACTION;
```
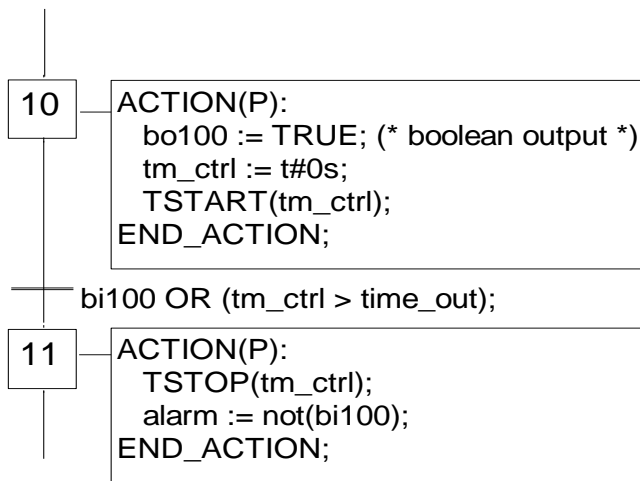
Time diagram if bi100 is always FALSE:
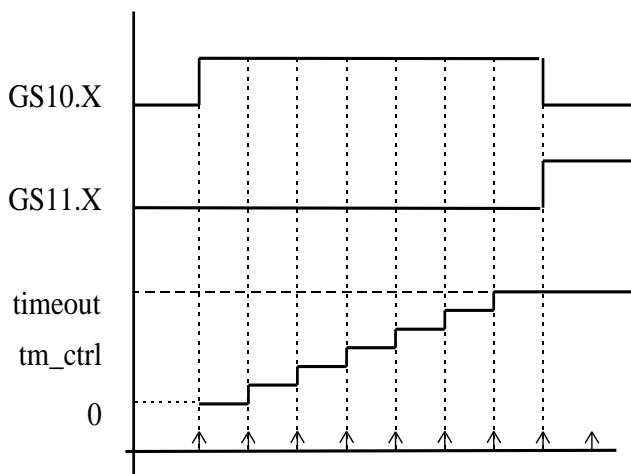
The timer keeps the same value during one cycle.

◓ *TSTOP statement*

Name: TSTOP
Meaning: stops updating a timer variable
              timer value is not modified by the TSTOP command
Syntax: TSTOP (<timer_variable>);
Operands: any active timer variable
Return value: (none)

Example: See TSTART (the function is described above)

◓ *GSTART statement*

Name: GSTART
Meaning: starts a child SFC program by putting a token
                  into each of its initial steps
Syntax: GSTART (<child_program>);
Operands: the specified SFC program must be a child of the one
                  in which the statement is written
Return value: (none)

Children of the child program are not automatically started by the GSTART statement.
Note: As GSTART is not in the IEC 1131-3 norm, prefer the use of the S qualifier, with the
    following syntax to start a child SFC:
    Child_name(S);

Example: Use of GSTART and GKILL
(* Sequence 'Sfather' *)   (* Sequence 'Schild' *)

◳ **GKILL statement**

Name: GKILL
Meaning: kills a child SFC program by removing the tokens
                  currently existing in its steps
Syntax: GKILL (<child_program>);
Operands: the specified SFC program must be a child of the one
                  in which the statement is written
Return value: (none)

Children of the child program are automatically killed with the specified program.
Note: As GKILL is not in the IEC 1131-3 norm, prefer the use of the R qualifier, with the
    following syntax to kill a child SFC:
    Child_name(R);

Example: See GSTART (function described above)

◳ **GFREEZE statement**

Name: GFREEZE
Meaning: Suspends the execution of a child SFC program.
                  Frozen program can be restarted by the GRST statement.
Syntax: GFREEZE (<child_program>);
Operands: the specified SFC program must be a child of the one
                  in which the statement is written
Return value: (none)

Children of the child program are automatically frozen along with the specified program.
Note: GFREEZE is not in the IEC 1131-3 norm.

Example:

```
        Suspend_cmd;
   ┌─┴─
     │1
 ┌───┴─┐
 │  2  │─┐ ACTION(P):
 └───┬─┘    GFREEZE(Schild);
     │    END_ACTION;

        NOT (Suspend_cmd);
   ┌─┴─
     │2
 ┌───┴─┐
 │  3  │─┐ ACTION(P):
 └───┬─┘    GRST(Schild);
     │    END_ACTION;
     │
```
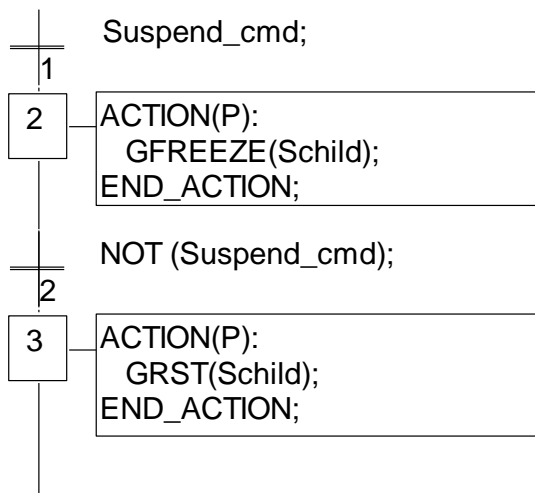
▫ *GRST statement*

Name: GRST
Meaning: Restarts a child SFC program frozen by the GFREEZE   statement.
Syntax: GRST (<child_program>);
Operands: the specified SFC program must be a child of the one
in which the statement is written
Return value: (none)
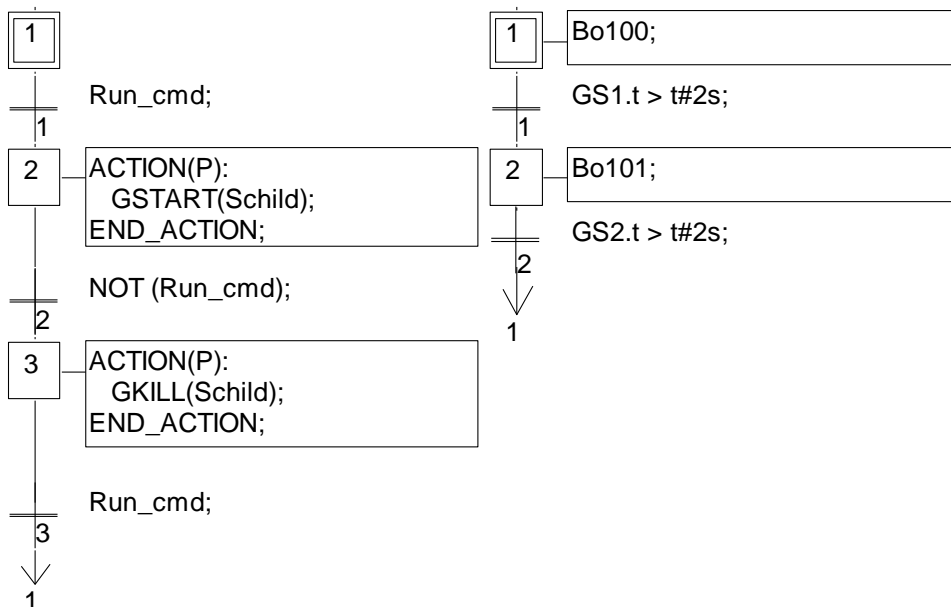

Children of the child program are automatically restarted by the GRST statement
Note: GRST is not in the IEC 1131-3 norm.

Example: See GFREEZE (function described above)

▫ *GSTATUS statement*

Name: GSTATUS
Meaning: returns the current status of an SFC program
Syntax: <ana_var> := GSTATUS (<child_program>);
Operands: the specified SFC program must be a child of the one
in which the statement is written
Return value: 0 = program is inactive (killed)
1 = program is active (started)
2 = program is frozen

Note: GSTATUS is not in the IEC 1131-3 norm.

Example:


```
1

    ──── Run_cmd;
    1

  ┌─┐  ┌──────────────────┐                    ┌───┐  ┌──────────────────────────┐
  │2│──│ ACTION(P):        │                    │201│──│ ACTION(N):                │
  └─┘  │   GSTART(Schild); │                    └───┘  │   if GSTATUS(Schild) = 0 then │
       │ END_ACTION;       │                           │     Mstat := 'Stopped';   │
       └──────────────────┘                            │   else                    │
                                                        │     Mstat := 'Running';   │
    ──── NOT(Run_cmd);                                  │   end_if;                 │
    2                                                   │ END_ACTION;               │
  ┌─┐  ┌──────────────────┐                             └──────────────────────────┘
  │3│──│ ACTION(P):        │
  └─┘  │   GKILL(Schild);  │
       │ END_ACTION;       │
       └──────────────────┘

    ──── Run_cmd;
    3
    ↓
    2
```
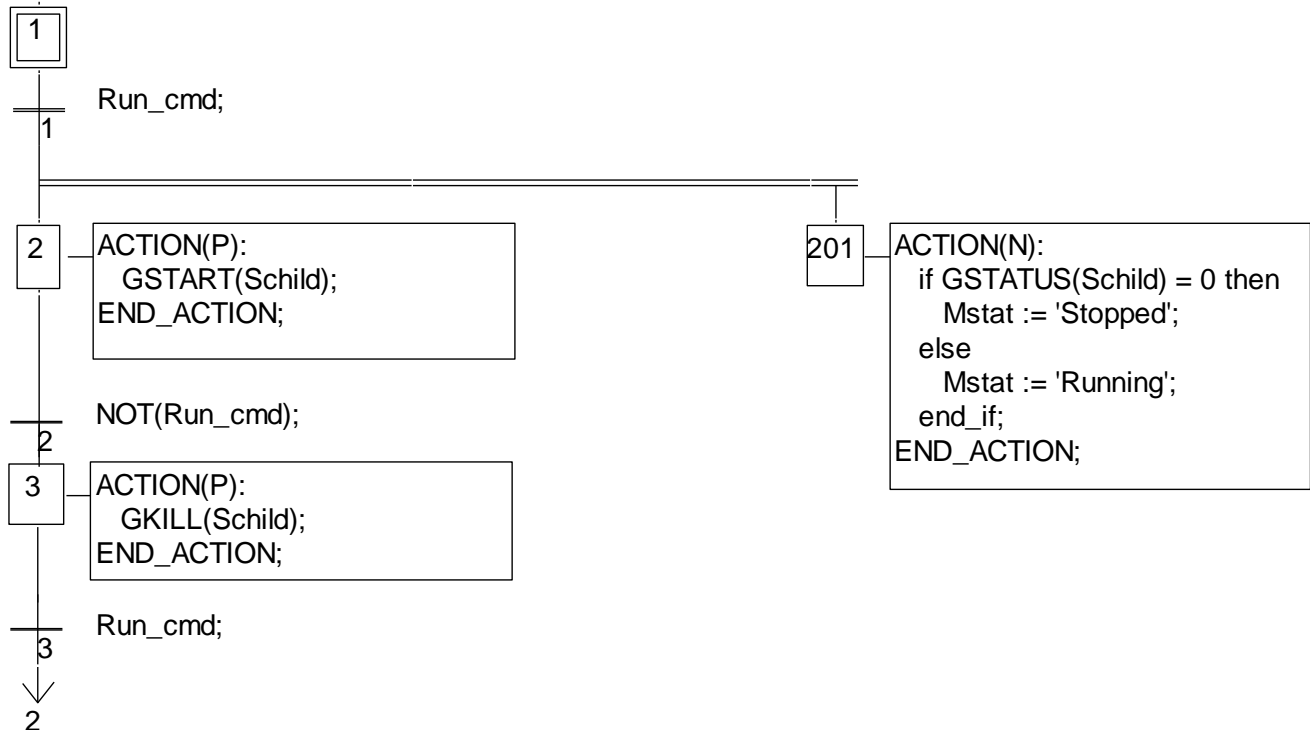
E.8 IL language

Instruction List, or **IL** is a low level language. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

E.8.1 IL main syntax

An IL program is a list of **instructions**. Each instruction must begin on a new line, and must contain an **operator**, completed with optional **modifiers** and, if necessary, for the specific operation, one or more **operands**, separated with commas (**','**). A **label** followed by a colon (**':'**) may precede the instruction. If a **comment** is attached to the instruction, it must be the last component of the line. Comments always begin with **'(*'** and ends with **'*)'**. Empty lines may be entered between instructions. Comments may be put on empty lines. Below are examples of instruction lines:

Label	Operator	Operand	Comments
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

◻ *Labels*

A **label** followed by a colon (**':'**) may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Naming labels must conform to the following rules:
- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters**, **digits** or **'_'** character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a variable.

◻ *Operator modifiers*

The available operator modifiers are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

N boolean negation of the operand
(delayed operation
C conditional operation

The **'N'** modifier indicates a boolean negation of the operand. For example, the instruction **ORN IX12** is interpreted as: **result := result OR NOT (IX12)**.

The parenthesis **'('** modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis **')'** operator is encountered.

The **'C'** modifier indicates that the attached instruction must be executed only if the current result has the boolean value TRUE (different than 0 for non-boolean values). The **'C'** modifier can be combined with the **'N'** modifier to indicate that the instruction must be executed only if the current result has the boolean value FALSE (or 0 for non-boolean values).

◛ *Delayed operations*

Because there is only one IL register (current result), some operations may have to be delayed, so that the execution order or the instructions can be changed. Parentheses are used to indicate delayed operations:

'('   is a modifier   indicates the operation to be
                                 delayed
')'   is an   executes the delayed
       operator   operation

The opening parenthesis **'('** modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis **')'** operator is encountered. For example, following sequence:

```
AND(   IX12
OR     IX35
)
```

is interpreted as:

```
result := result AND ( IX12 OR IX35 )
```

E.8.2 IL operators

The following table summarizes the standard operators of the IL language:

Operator	Modifiers	Operand	Description
LD	N	Variable, constant	Loads operand
ST	N	Variable	Stores current result
S		BOO variable	Sets to TRUE
R		BOO variable	Resets to FALSE
AND	N (	BOO	boolean AND
&	N (	BOO	boolean AND
OR	N (	BOO	boolean OR
XOR	N (	BOO	exclusive OR
ADD	(	variable, constant	Addition
SUB	(	variable, constant	Subtraction
MUL	(	variable, constant	Multiplication
DIV	(	variable, constant	Division

GT	(	variable, constant	Test: >
GE	(	variable, constant	Test: >=
EQ	(	variable, constant	Test: =
LE	(	variable, constant	Test <=
LT	(	variable, constant	Test <
NE	(	variable, constant	Test <>
CAL	C N	Function block	Calls a function block
JMP	C N	instance name	Jumps to label
RET	C N	Label	Returns from sub-program
)			Executes delayed operation

In the next section, only operators which are specific to the IL language are described, other standard operators can be found in the section "standard operators, function blocks and functions".

LD operator

Operation   loads a value in the current result
Allowed modifiers   N
Operand   constant expression
                   internal, input or output variable

Example:

```
    (* EXAMPLES OF LD OPERATIONS *)
LDex:   LD      false               (* result := FALSE boolean constant *)
        LD      true                (* result := TRUE boolean constant *)
        LD      123                 (* result := integer constant *)
        LD      123.1               (* result := real constant *)
        LD      t#3ms               (* result := time constant *)
        LD      boo_var1            (* result := boolean variable *)
        LD      ana_var1            (* result := analog variable *)
        LD      tmr_var1            (* result := timer variable *)
        LDN     boo_var2            (* result := NOT ( boolean variable ) *)
```

ST operator

Operation   stores the current result in a variable
                 the current result is not modified by this operation
Allowed modifiers   N
Operand   internal or output variable

Example:

```
        (* EXAMPLES OF ST OPERATIONS *)
STboo:  LD      false
        ST      boo_var1            (* boo_var1 := FALSE *)
        STN     boo_var2            (* boo_var2 := TRUE *)
STana:  LD      123
```

```
              ST        ana_var1            (* ana_var1 := 123 *)
STtmr:        LD        t#12s
              ST        tmr_var1            (* tmr_var1 := t#12s *)
```

▭ *S operator*

Operation: stores the boolean value TRUE in a boolean variable, if the current result has
                   the boolean value TRUE. No operation is processed if current result is FALSE.
                   The current result is not modified by this operation

Allowed modifiers: (none)
Operand: output or internal boolean variable

Example:

```
          (* EXAMPLES OF S OPERATIONS *)
SETex:    LD      true            (* current result := TRUE *)
          S       boo_var1        (* boo_var1 := TRUE *)
                                  (* current result is not modified *)
          LD      false           (* current result := FALSE *)
          S       boo_var1        (* nothing done - boo_var1 unchanged *)
```

▭ *R operator*

Operation   stores the boolean value FALSE in a boolean variable, if the current result has
                   the boolean value TRUE. No operation is processed if current result is FALSE.
                   The current result is not modified by this operation

Allowed modifiers   (none)
Operand   output or internal boolean variable

Example:

```
    (* EXAMPLES OF R OPERATIONS *)
RESETex:  LD    true            (* current result := TRUE *)
          R     boo_var1        (* boo_var1 := FALSE *)
                                (* current result is not modified *)
          ST    boo_var2        (* boo_var2 := TRUE *)
          LD    false           (* current result := FALSE *)
          R     boo_var1        (* nothing done - boo_var1 unchanged *)
```

▭ *JMP operator*

Operation   jumps to the specified label
Allowed modifiers   C   N
Operand   label defined in the same IL program

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)  *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with        *)
(* the JMPC operator                            *)

JMPex:    LD        selector            (* selector is 0 or 1 or 2 *)
          BOO                           (* conversion to boolean *)
```

```
            JMPC      test1               (* if selector = 0 then *)
            LD        true
            ST        bo0                 (* bo0 := true *)
            JMP       JMPend              (* end of the program *)
test1:      LD        selector
            SUB       1                   (* decrease selector:  is now 0 or 1 *)
            BOO                           (* conversion to boolean *)
            JMPC      test2               (* if selector = 0 then *)
            LD        true
            ST        bo1                 (* bo1 := true *)
            JMP       JMPend              (* end of the program *)
test2:      LD        true                (* last possibility *)
            ST        bo2                 (* bo2 := true *)
JMPend:                                   (* end of the IL program *)
```

▬ *RET operator*

Operation   ends the current instruction list. If the IL sequence is a sub-program, the current result is returned to the calling program

Allowed modifiers   C   N
Operand   (none)

Example:

(* the following example tests the value of an analog selector (0 or 1 or 2)   *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with   *)
(* the JMPC operator   *)

```
JMPex:      LD        selector            (* selector is 0 or 1 or 2 *)
            BOO                           (* conversion to boolean *)
            JMPC      test1               (* if selector = 0 then *)
            LD        true
            ST        bo0                 (* bo0 := true *)
            RET                           (* end - return 0 *)
                                          (* decrease selector *)
test1:      LD        selector
            SUB       1                   (* selector is now 0 or 1 *)
            BOO                           (* conversion to boolean *)
            JMPC      test2               (* if selector = 0 then *)
            LD        true
            ST        bo1                 (* bo1 := true *)
            LD        1                   (* load real selector value *)
            RET                           (* end - return 1 *)
                                          (* last possibility *)
test2:      RETNC                         (* returns if the selector has *)
                                          (* an invalid value *)
            LD        true
            ST        bo2                 (* bo2 := true *)
            LD        2                   (* load real selector value *)
                                          (* end - return 2 *)
```

- *")" operator*

Operation　　　　executes a delayed operation. The delayed operation was notified by '('
Allowed modifiers　　(none)
Operand　　　　　(none)

Example:

(* The following program interleaves delayed operations: *)
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)

Delayed:	LD	a1	(* result := a1; *)
	ADD(	a2	(* delayed ADD - result := a2; *)
	MUL(	a3	(* delayed MUL - result := a3; *)
	SUB	a4	(* result := a3 - a4; *)
	)		(* execute delayed MUL - result := a2 * (a3-a4); *)
	MUL	a5	(* result := a2 * (a3 - a4) * a5; *)
	)		(* execute delayed ADD *)
			(* result := a1 + (a2 * (a3 - a4) * a5); *)
	ADD	a6	(* result := a1 + (a2 * (a3 - a4) * a5) + a6; *)
	ST	res	(* store current result in variable res *)

- *Calling sub-programs or functions*

A sub-program or a function (written in any of the IL, ST, LD, FBD or "C" language) is called
　　from the IL language, using its name as an operator.

Operation　　　　executes a sub-program or a function - the value returned by the
　　　　　　　　　　sub-program or function is stored into the IL current result
Allowed modifiers　　(none)
Operand　　　　　The first calling parameter must be stored in the current result before the
　　　　　　　　　　call. The following ones are expressed in the operand field, separated by
　　　　　　　　　　comas.

Example:

(* Calling program : converts an analog value into a time value *)

Main:	LD	bi0	
	SUBPRO	bi1,bi2	(* call sub-program to get analog value *)
	ST	result	(* result := value returned by sub-program *)
	GT	vmax	(* test value overflow *)
	RETC		(* return if overflow *)
	LD	result	
	MUL	1000	(* converts seconds in milliseconds *)
	TMR		(* converts to a timer *)
	ST	tmval	(* stores converted value in a timer *)

(* Called sub-program named 'SUBPRO' : evaluates the analog value *)
(* given as a binary value on three boolean inputs: in0, in1, in2 are the three boolean input
　　parameters of the sub-program *)

```
        LD        in2
        ANA                                (* result = ana (in2); *)
        MUL       2                        (* result := 2*ana (in2); *)
        ST        temporary                (* temporary := result *)
        LD        in1
        ANA
        ADD       temporary                (* result := 2*ana (in2) + ana (in1); *)
        MUL       2                        (* result := 4*ana (in2) + 2*ana (in1); *)
        ST        temporary                (* temporary := result *)
        LD        in0
        ANA
        ADD       temporary                (* result := 4*ana (in2) + 2*ana (in1)+ana (in0); *)
        ST        SUBPRO                   (* return current result to calling program *)
```

Calling function blocks: CAL operator

Operation calls a function block
Allowed modifiers   C   N
Operand   Name of the function block instance.
     The input parameters of the blocks must be assigned before the call using LD/ST
            operations sequence.
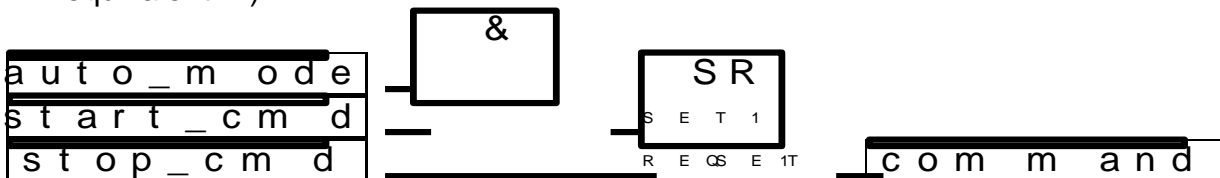   Output parameters are known if used.

Example1:

(* Calling function block SR : SR1 is an instance of SR *)
```
    LD      auto_mode
    AND     start_cmd
    ST      SR1.set1
    LD      stop_cmd
    ST      SR1.reset
    CAL     SR1
    LD      SR1.Q1
    ST      command
```

(* FBD equivalent : *)

Example 2
(*We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU
    block*)
```
    LD      command
    ST      R_TRIG1.clk
    CAL     R_TRIG1
    LD      R_TRIG1.Q
```
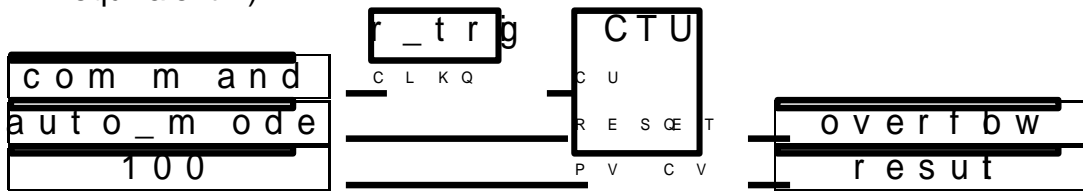
```
ST     CTU1.cu
LDN    auto_mode
ST     CTU1.reset
LD     100
ST     CTU1.pv
CAL    CTU1
LD     CTU1.Q
ST     overflow
LD     CTU1.cv
ST     result
```

(* FBD equivalent: *)

Appendix F : How to Enable/Disable W-8x47's LAN2

Important:
1. Please always set LAN2 as disabled if not using it.
2. Please always set a fixed IP to LAN1 (or LAN2 if it is enabled).

The default setting of LAN2 port of W-8047/8347/8747 & W-8046/8346/8746 is disabled. User must enable it before using LAN2 port.

Please open "Start" – "Setting" - "Control Panel" and then click on "Network and Dual-up Connections" to set as LAN2: DM9CE1 Enable or Disable
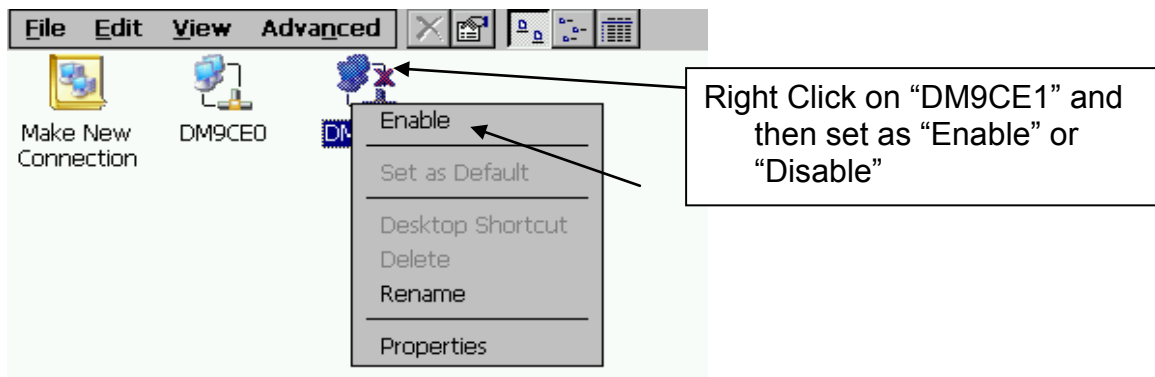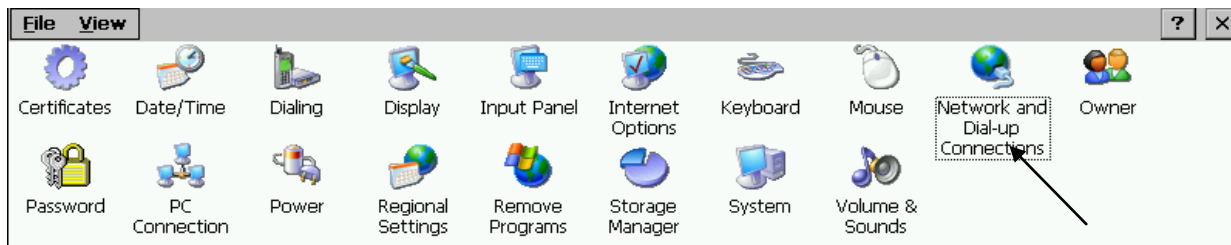
Then run "Start" – "Programs" – "Wincon Utility", click "Save and Reboot" to save the setting.