
Contents of Appendix

CONTENTS OF APPENDIX.....	1
APPENDIX A: FUNCTIONS & FUNCTION BLOCKS FOR ICP DAS CONTROLLERS	6
APPENDIX A.1: STANDARD ISAGRAF FUNCTION BLOCKS	6
APPENDIX A.2: ADDING NEW FUNCTION BLOCKS TO ISAGRAF.....	8
APPENDIX A.3: I-8xx7 & I-7188EGD/XGD's 7-SEGMENT LED REFERENCE TABLE	10
APPENDIX A.4: FUNCTION BLOCKS FOR THE CONTROLLER	11
A4_20_to	11
ARRAY_R	12
ARRAY_W	12
ARY_F_R	13
ARY_F_W	13
ARY_N_R	14
ARY_N_W	14
ARY_W_R	15
ARY_W_W	15
BCD_V	16
BIN2ENG	16
BIT_WD	16
COMARY_R	17
COMARY_W	17
COMAY_NW	18
COMAY_WW	19
COMCLEAR	20
COMCLOSE	20
COMOPEN	21
COMOPEN2	22
COMREAD	23
COMREADY	24
COMSTR_W	25
COMWRITE	26
CRC_16	27
DI_CNT	28
EBUS_B_R	28
EBUS_B_W	28
EBUS_N_R	29
EBUS_N_W	29
EBUS_STS	29
EEP_B_R	30
EEP_B_W	30
EEP_BY_R	31
EEP_BY_W	31
EEP_EN	32
EEP_N_R	32
EEP_N_W	32

EEP_PR.....	33
EEP_WD_R.....	33
EEP_WD_W.....	33
EMAIL.....	34
F_APPEND.....	34
F_COPY.....	34
F_CREAT.....	35
F_DELETE.....	35
F_DIR.....	36
F_END.....	36
F_EOF.....	37
F_READ_B.....	37
F_READ_F.....	37
F_READ_W.....	38
F_SEEK.....	38
F_WRIT_B.....	38
F_WRIT_F.....	39
F_WRIT_W.....	39
F_WRIT_S.....	39
FBUS_B_R.....	40
FBUS_B_W.....	40
FBUS_N_R.....	41
FBUS_N_W.....	41
FBUS_STS.....	41
GET_SN.....	42
INP10LED.....	43
INP16LED.....	44
INT_REAL.....	45
I_RESET.....	45
I7000_EN.....	45
LONG_WD.....	46
MBUS_B_R.....	46
MBUS_BR1.....	46
Mbus_EN.....	47
MBUS_B_W.....	47
MBUS_N_R.....	48
MBUS_NR1.....	48
MBUS_N_W.....	49
MBUS_R.....	50
MBUS_R1.....	51
MBUS_WB.....	52
MI_BOO.....	53
MI_INP_N.....	53
MI_INP_S.....	54
MI_INT.....	54
MI_REAL.....	55
MI_STR.....	55
MSGARY_R.....	56
MSGARY_W.....	56

PID_AL.....	57
PWM_DIS.....	57
PWM_EN.....	57
PWM_EN2.....	57
PWM_ON.....	57
PWM_OFF.....	57
PWM_SET.....	57
PWM_STS.....	57
PWM_STS2.....	57
RDN_B.....	58
RDN_F.....	58
RDN_N.....	58
RDN_T.....	58
REAL_INT.....	59
REAL_STR.....	59
REA_STR2.....	59
Retain_B.....	60
Retain_F.....	60
Retain_N.....	60
Retain_T.....	61
Retain_X.....	61
R_MB_Adr.....	62
R_MB_Rel.....	62
S_B_R.....	63
S_B_W.....	63
S_BY_R.....	64
S_BY_W.....	64
S_DL_DIS.....	65
S_DL_EN.....	65
S_DL_RST.....	65
S_DL_STS.....	65
SET_LED.....	66
S_FL_AVL.....	67
S_FL_INI.....	68
S_FL_RST.....	68
S_FL_STS.....	69
SMS_GET.....	70
SMS_GETS.....	71
SMS_SEND.....	72
SMS_STS.....	73
SMS_TEST.....	74
S_M_R.....	75
S_M_W.....	75
S_MV.....	76
S_N_R.....	77
S_N_W.....	77
S_R_R.....	78
S_R_W.....	78
S_WD_R.....	79

S_WD_W	79
STR_REAL	80
SYSDAT_R	81
SYSDAT_W	82
SYSTM_R	83
SYSTM_W	84
TIME_STR	85
TWIN_LED	85
To_A4_20	86
To_V0_10	87
UDP_recv	88
UDP_send	88
V0_10_to	89
VAL_HEX	90
VAL10LED	90
VAL16LED	91
V_BCD	91
WD_BIT	92
WD_LONG	92
W_MB_Adr	93
W_MB_Rel	93
APPENDIX B: SETTING THE IP, MASK & GATEWAY IN THE I-8437/8837 & I-7188EG	94
APPENDIX C: UPDATE THE I-8417 / 8817 / 8437 / 8837 CONTROLLER TO NEW HARDWARE DRIVER	96
APPENDIX C.1: SETTING I-8xx7 & I-7188EG's COM1 As NONE-MODBUS-SLAVE PORT	98
APPENDIX D: TABLE OF THE ANALOG IO VALUE	99
I-87013, I-7013, I-7033	99
I-8017H	100
I-87017, I-7017	101
I-87018, I-7011, I-7018	102
I-7021	104
I-7022	104
I-8024	105
I-87024, I-7024	105
APPENDIX E: LANGUAGE REFERENCE	106
ISAGRAF	107
LANGUAGE REFERENCE	107
ALTERSYS INC.	107
E.1 PROJECT ARCHITECTURE	108
E.1.1 Programs	108
E.1.2 Cyclic and sequential operations	108
E.1.3 Child SFC and FC programs	109

E.1.4 Functions and sub-programs	109
E.1.5 Function blocks	110
E.1.6 Description language	111
E.1.7 Execution rules	112
E.2 COMMON OBJECTS	113
E.2.1 Basic types	113
E.2.2 Constant expressions	113
E.2.3 Variables	115
E.2.4 Comments	119
E.2.5 Defined words	119
E.3 SFC LANGUAGE	121
E.3.1 SFC chart main format	121
E.3.2 SFC basic components	121
E.3.3 Divergences and convergences	123
E.3.4 Macro steps	125
E.3.5 Actions within the steps	126
E.3.6 Conditions attached to transitions	132
E.3.7 SFC dynamic rules	134
E.3.8 SFC program hierarchy	134
E.4 FLOW CHART LANGUAGE	136
E.4.1 FC components	136
E.4.2 FC complex structures	139
E.4.3 FC dynamic behaviour	140
E.4.4 FC checking	140
E.5 FBD LANGUAGE	141
E.5.1 FBD diagram main format	141
E.5.2 RETURN statement	142
E.5.3 Jumps and labels	142
E.5.4 Boolean negation	143
E.5.5 Calling function or function blocks from the FBD	143
E.6 LD LANGUAGE	145
E.6.1 Power rails and connection lines	145
E.6.2 Multiple connection	146
E.6.3 Basic LD contacts and coils	147
E.6.4 RETURN statement	153
E.6.5 Jumps and labels	153
E.6.6 Blocks in LD	154
E.7 ST LANGUAGE	156
E.7.1 ST main syntax	156
E.7.1 Expression and parentheses	156
E.7.3 Function or function block calls	157
E.7.4 ST specific boolean operators	158
E.7.5 ST basic statements	160
E.7.6 ST extensions	165
E.8 IL LANGUAGE	171
E.8.1 IL main syntax	171
E.8.2 IL operators	172
APPENDIX F: HOW TO ENABLE/DISABLE W-8X47'S LAN2.....	179

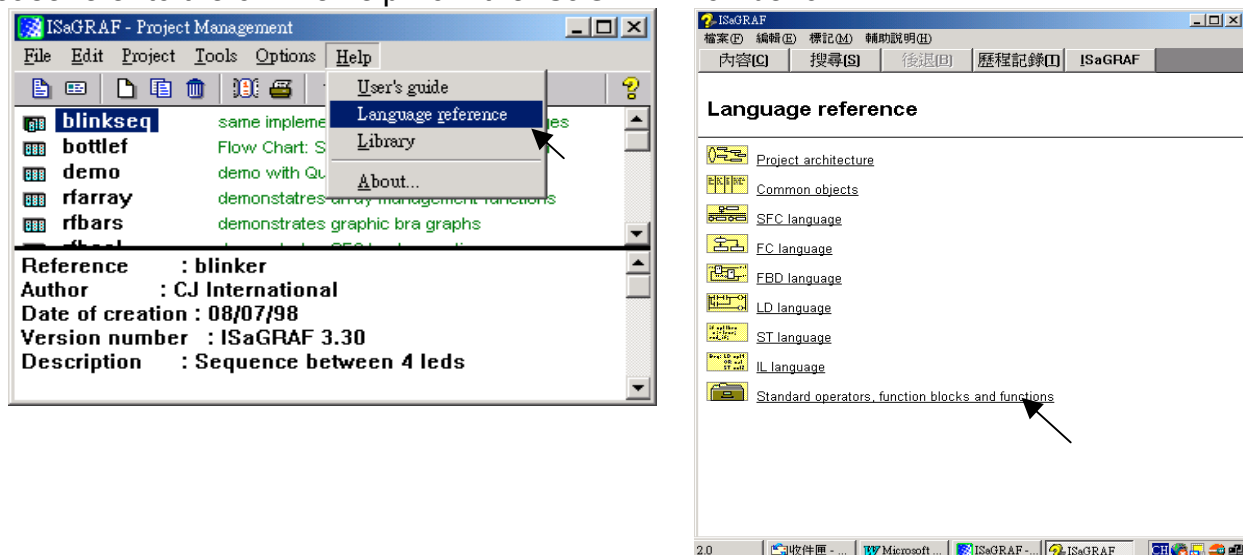
Appendix A: Functions & Function Blocks For ICP DAS Controllers

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 & W-8xx7 controller however labeled with “*” & “#” is not supported by I-8xx7 & I-7188EG/XG, while W-8xx7 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.



The function blocks listed in section A.4 are created by ICP DAS exclusively for the I-8xx7, I-7188EG/XG & W-8xx7 controller system. After installing the "ICP DAS Utilities For ISaGRAF" (please refer to section 1.2), these blocks in section A.4 can be found in the ISaGRAF Workbench program. Please refer to section A.4 for the "List Of Blocks" created for the controller system.

ICP DAS continually strives to improve the functionality of the I-8xx7, I-7188EG/XG & W-8xx7 controller system and the ISaGRAF Workbench program. Please visit the ICP DAS web site at

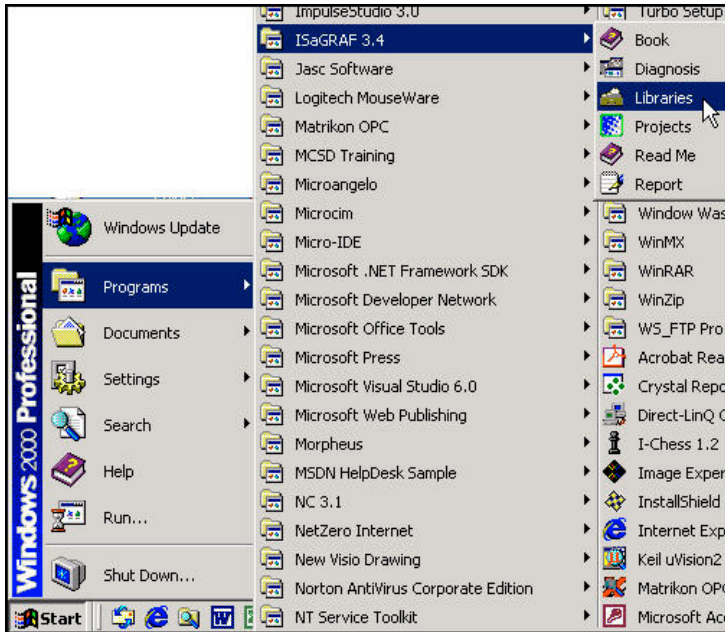
<http://www.icpdas.com/products/PAC/i-8000/isagraf.htm>

for updates and additions of new function blocks and functions created for the controller system.

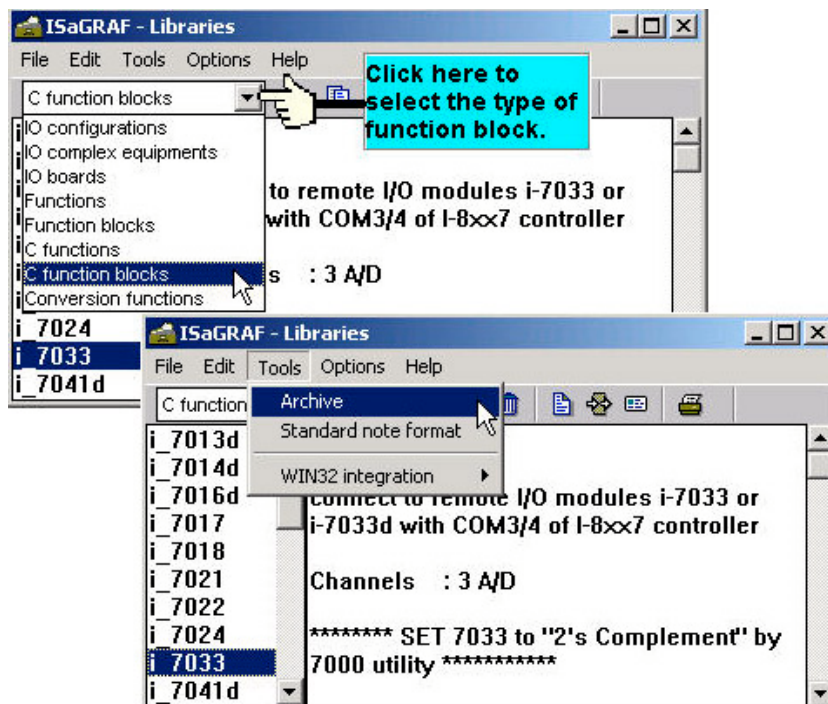
Please refer to section A.2 for more information on how to "Add New Blocks one by one To The ISaGRAF Workbench" program. (Section 1.2 is to install all of them at once)

Appendix A.2: Adding New Function Blocks To ISaGRAF

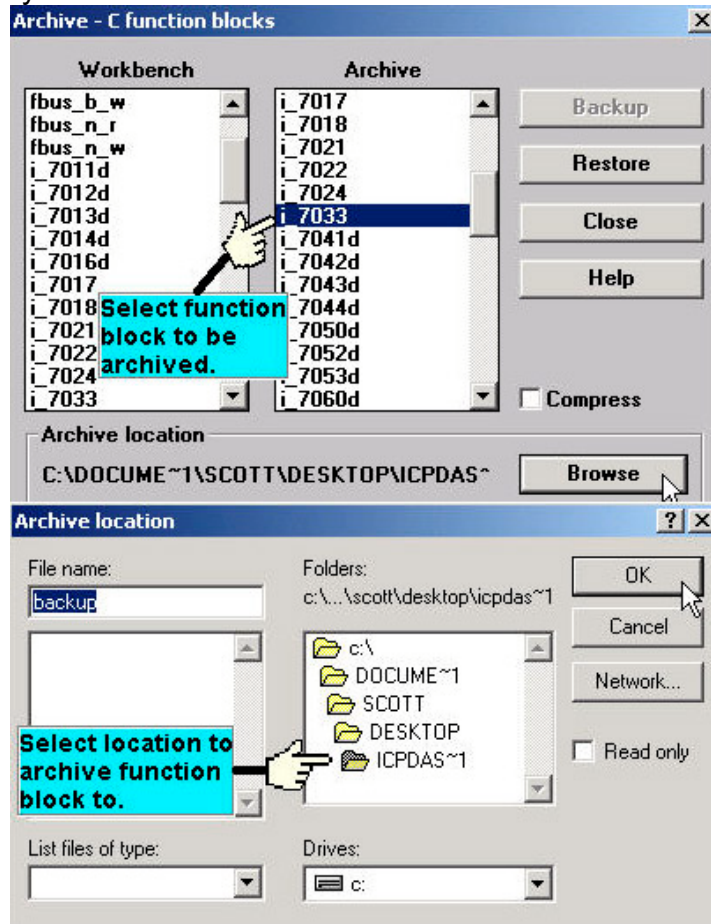
To add or update functions or function blocks one by one for the ISaGRAF Workbench program, click on the Windows "Start" menu, select "Programs", select "ISaGRAF 3.4" or "ISaGRAF 3.5", then click on "Libraries" to begin installing or updating ISaGRAF functions or function blocks.



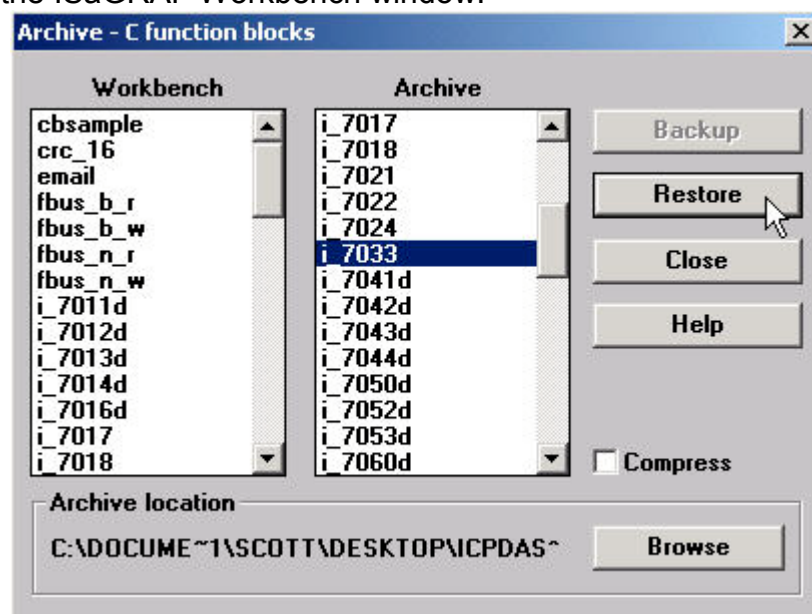
When you click on "Libraries" the "ISaGRAF Libraries" window will open. To add a new function block or function select "Tools" from the menu bar and then click on "Archive".



Click on the file name you want to "Archive" and then click "Browse" button to select the sub-directory to where (CD_ROM: \Napdos\ISaGRAF\ARK\) you want to archive the function block library to.

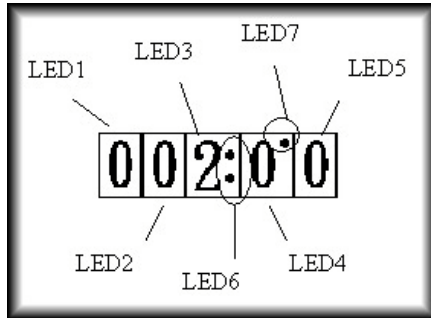


Select the new function block in the "Archive" window that you want to add, and then click on the "Restore" button. When you click on the "Restore" button the function block will be added to the ISaGRAF Workbench window.



Appendix A.3: I-8xx7 & I-7188EGD/XGD's 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 controller 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

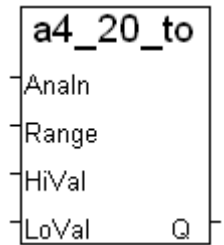
Displayed Char.	Given Value	Displayed Char.	Given Value	Displayed Char.	Given Value
0	0	4.	20	r	40
1	1	5.	21	L	41
2	2	6.	22	n	42
3	3	7.	23	y	43
4	4	8.	24	U	44
5	5	9.	25	P	45
6	6	A.	26	o	46
7	7	b.	27	r.	47
8	8	C.	28	n.	48
9	9	d.	29	y.	49
A	10	E.	30	h.	50
b	11	F.	31	L.	51
C	12	~	32	U.	52
d	13	-	33	P.	53
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 & W-8xx7 controller system.

A4_20_to

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Convert Analog Input from 4 - 20 mA to User's Engineering Value ("Real" format)

Arguments:

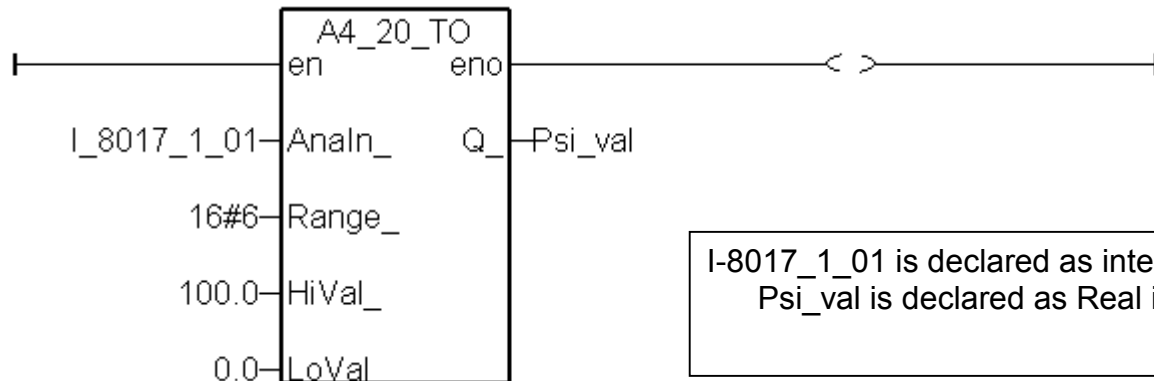
Analn_ Integer	the integer variable related to the Analog input board or module. The variable value is usually from -32768 to +32767 depends on the range setting of the IO board.
Range_ Integer	Range setting of the Analog input board or module. 16#6 : -20 to +20 mA 16#D : -20 to +20 mA
HiVal_ Real	User's 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 For example, 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 (depends 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



I-8017_1_01 is declared as integer input,
Psi_val is declared as Real internal.

Note:

Please refer to similar functions: to_A4_20 , to_V0_10 , A4_20_to , V0_10_to

ARRAY_R

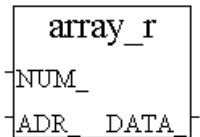
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

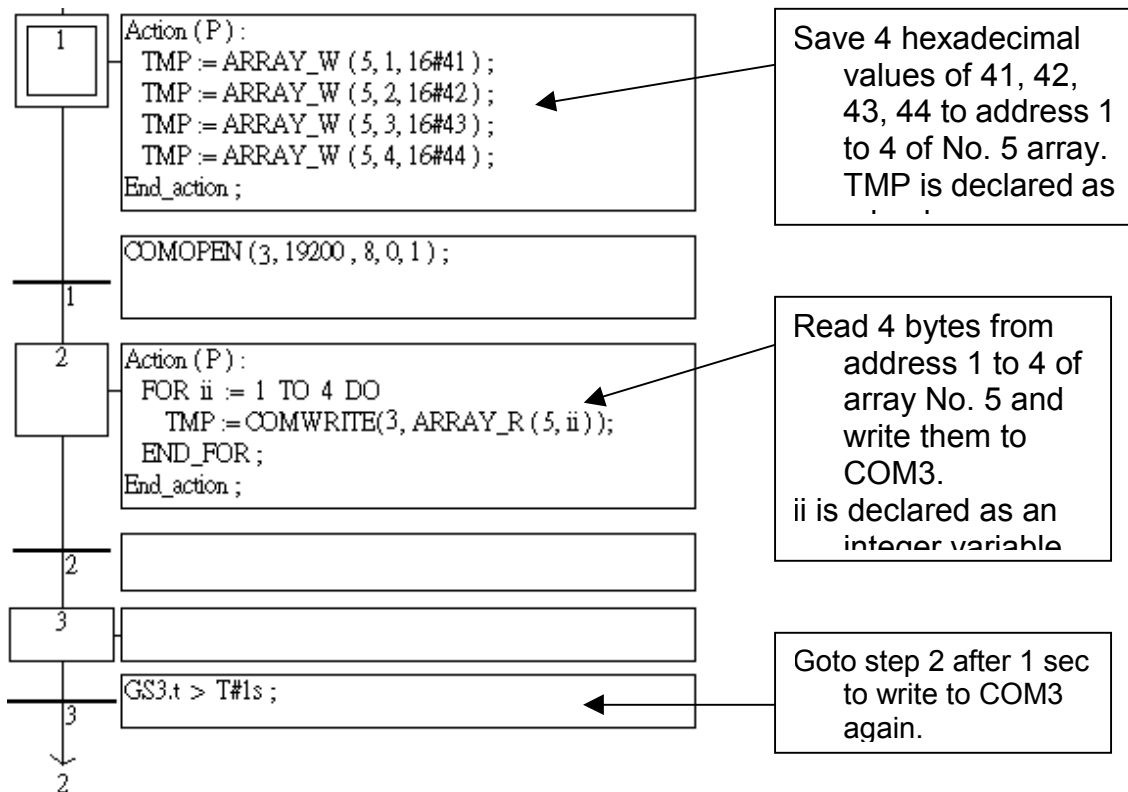
Function Read one byte from a byte array

Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & 7188EG/XG is from 1 to 24. For W-8xx7 is 1 to 48.
ADR_	integer	address in the array where the byte is to be stored, for the I-8xx7 & 7188EG/XG is from 1 to 256. For W-8xx7 is 1 to 512.
DATA_	integer	the byte value returned



Example:



ARRAY_W

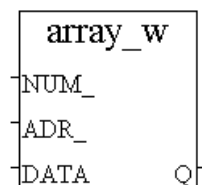
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Save one byte to a byte array

Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & 7188EG/XG is from 1 to 24. For W-8xx7 is 1 to 48.
ADR_	integer	address in the array where the byte is to be stored, for the I-8xx7 & 7188EG/XG is from 1 to 256. For W-8xx7 is 1 to 512.
DATA_	integer	the byte value to be saved to, valid range values from 0 to 255.
Q_	boolean	if OK. return TRUE, else return FALSE

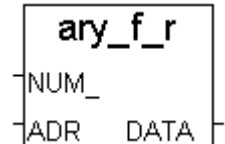


ARY_F_R

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Read one float value (32-bit format) from an float array**



Arguments:

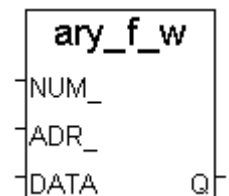
NUM_	integer	array ID to be operated, valid range values is from 1 to 18.
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	real	the float value returned

ARY_F_W

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Save one float value (32-bit format) to an float array**



Arguments:

NUM_	integer	array ID to be operated, valid range values is from 1 to 18
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	real	the float value to be saved to.
Q_	boolean	if OK. return TRUE, else return FALSE

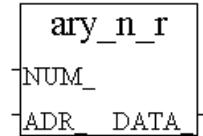
Note: The datas stored in array are cleared after power off

ARY_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Read one integer (signed 32-bit) from an integer array**



Arguments:

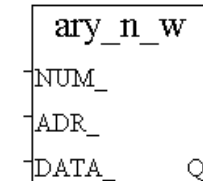
NUM_ integer array ID to be operated, valid range values for the I-8xx7 & I-7188EG/XG is from 1 to 6. For W-8xx7 is 1 to 18.
ADR_ integer address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_ integer the integer value returned

ARY_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Save one integer to an integer array**



Arguments:

NUM_ integer array ID to be operated, valid range values for the I-8xx7 & I-7188EG/XG is from 1 to 6. For W-8xx7 is 1 to 18
ADR_ integer address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_ integer the integer value to be saved to.
Q_ boolean if OK. return TRUE, else return FALSE

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. (Please refer to Section 4.5)

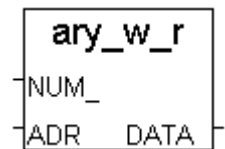
Word array (ID, ADR)	Integer array (ID, ADR)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
...	...
...	
(12,255)	(6,256)
(12,256)	
...	...
...	

2. Data stored in array is cleared after power off

Example: Refer to the “ARRAY_R” example.

ARY_W_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function read short integer (signed 16-bit) from array

Arguments:

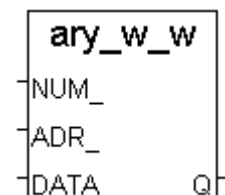
NUM_ integer array ID to be operated, for the I-8xx7 & I-7188EG/XG is from 1 to 12. For W-8xx7 is 1 to 36

ADR_ integer address in the array where the integer is to be stored, valid range values from 1 to 256

DATA_ integer the integer value returned, ranging from -32768 ~ +32767

ARY_W_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function write 1 short integer (signed 16-bit) to array of I-8xx7 controller

Arguments:

NUM_ integer array ID to be operated, for the I-8xx7 & I-7188EG/XG is from 1 to 12. For W-8xx7 is 1 to 36

ADR_ integer address in the array where the integer is to be stored, valid Range values from 1 to 256

DATA_ integer the integer value to be saved to. (-32768~+32767)

Q_ boolean if OK. return TRUE, else return FALSE

Note:

1. The long integer array use the same memory as short integer array. Be careful if use both of them at the same time.

Word array (ID, ADR)	Integer array (ID, ADR)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
...	...
...	
(12,255)	(6,256)
(12,256)	
...	...
...	

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

Example: Refer to the “ARRAY_R” example.

BCD_V

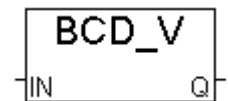
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Convert BCD value to decimal value

Arguments:

IN_	integer	the BCD value to be converted
Q_	integer	the returned value, For ex. 16#12345 → 12345 16#3490 → 3490 18 → 12



BIN2ENG

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Transfer 2's complement value to Engineering format value

Arguments:

IN_	integer	2's complement value to be converted
HI_2s_	integer	upper limit of 2's complement, -32768 to +32767
LO_2s_	integer	lower limit of 2's complement, -32768 to +32767
HI_EN_	integer	upper limit of engineering format, -32768 to +32767
LO_EN_	integer	lower limit of engineering format, -32768 to +32767
OUT_	integer	the returned engineering format value, for ex.

HI_2s_ = 32767 , LO_2s_ = -32768, HI_EN_ = 1000, LO_EN_ = -1000
IN_ = 16383 → OUT_ = 500
IN_ = -12345 → OUT_ = -377



BIT_WD

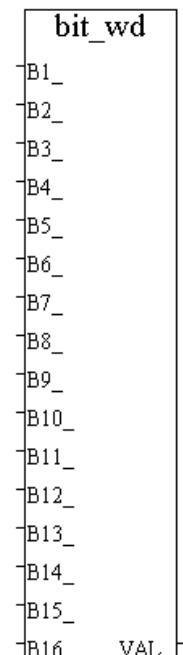
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Convert 16 boolean values to a word value

Arguments:

B1_ ~ B16_	boolean	the 16 boolean values to be converted
VAL_	integer	the word value after the conversion For ex. If B1_ and B2_ are TRUE and others are all FALSE, VAL_ will be 3. If only B4_ is TRUE and others are all FALSE, VAL_ will be 8

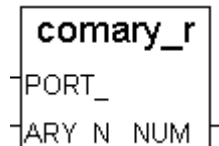


COMARY_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Read all of the ready data of a COM PORT to a byte array**



Argument:

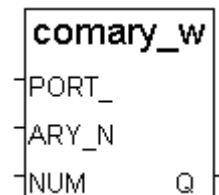
PORT_ integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
ARY_NO_ integer	Byte array ID (1-24 for I-8xx7 & I-7188EG/XG), (1-48 for W-8xx7), which is used to store the read bytes
NUM_ integer	return the number of bytes been read

COMARY_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Write a byte array to a COM PORT**



Argument:

PORT_ integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
ARY_NO_ integer	Byte array ID (1-24 for I-8xx7 & I-7188EG/XG), (1-48 for W-8xx7), which is used to store the read bytes
NUM_ integer	the number of bytes starting from the first address in the byte array to write
Q_ boolean	OK. return TRUE

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

Example:

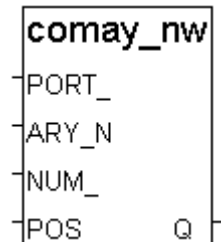
Refer to Chapter 11 - Demo_21, 22 & 23.
Refer to function "ARRAY_R" & "ARRAY_W"

COMAY_NW

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Write one long integer 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 formate.

[lowest byte] [] [] [highest byte]

For ex., if there is 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]

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
ARY_NO_	integer	array ID (1-6 for I-8xx7 & I-7188EG/XG), (1-18 for W-8xx7), which is to write
NUM_	integer	the number of long integers starting from the POS_ address in the array to write
POS_	Integer	start position inside the array to write (1-256) if POS_ + NUM_ > 257, only (257-POS_) integer will be written for ex. if POS_=255, NUM_=3, only 2 integers written. They are Pos. 255 & Pos. 256.
Q_	boolean	OK. return TRUE

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

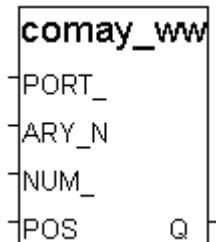
The long int array use the same memory as short interger array. Be careful if use both of them at the same time (please refer to Ary_n_r, Ary_n_w, Ary_w_r, Ary_w_w)

COMAY_WW

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Write one short Integer (Word) array to COM PORT



Each short integer is composed of 2 bytes. And the format is a signed short int (-32768 ~ +32767).

Each short integer written is composed of 2 bytes in the below INTEL formate.

[low byte] [high byte]

For ex., if there is 3 short integers to write, the first one is 16#0403 (1,027), the second 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]

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
ARY_NO_	integer	array ID (1-12 for I-8xx7 & I-7188EG/XG), (1-36 for W-8xx7), which is to write
NUM_	integer	the number of short integers starting from the POS_ address in the array to write
POS_	Integer	start position inside the array to write (1-256) if POS_ + NUM_ > 257, only (257-POS_) integer will be written for ex. if POS_=255, NUM_=3, only 2 integers written. They are Pos. 255 & Pos. 256.
Q_	boolean	OK. return TRUE

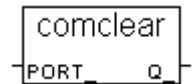
Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

The long int array use the same memory as short interger array. Be careful if use both of them at the same time.

COMCLEAR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

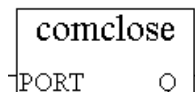
Function Clear receiving buffer of a COM PORT

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
Q_	boolean	OK. return TRUE

COMCLOSE

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Close COM PORT

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
Q_	boolean	OK. return TRUE

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

Example:

Refer to the "COMOPEN" example.

COMOPEN

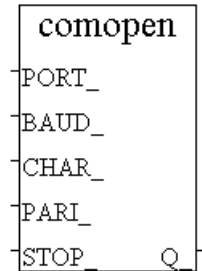
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Open COM port**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or 5 to 14...
BAUD_	integer	baud rate, can be 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	integer	character size, can be 7 or 8
PARI_	integer	parity, can be 0: none, 1: even, 2: odd, 3: mark, 4: space 3 & 4 is only for I-8xx7: COM3 ~ 20, I-7188EG/XG: COM3 ~ 8. While COM2, or ... for Wincon-8xx7
STOP_	integer	stop bit, can be 1 or 2
Q_	boolean	OK. return TRUE

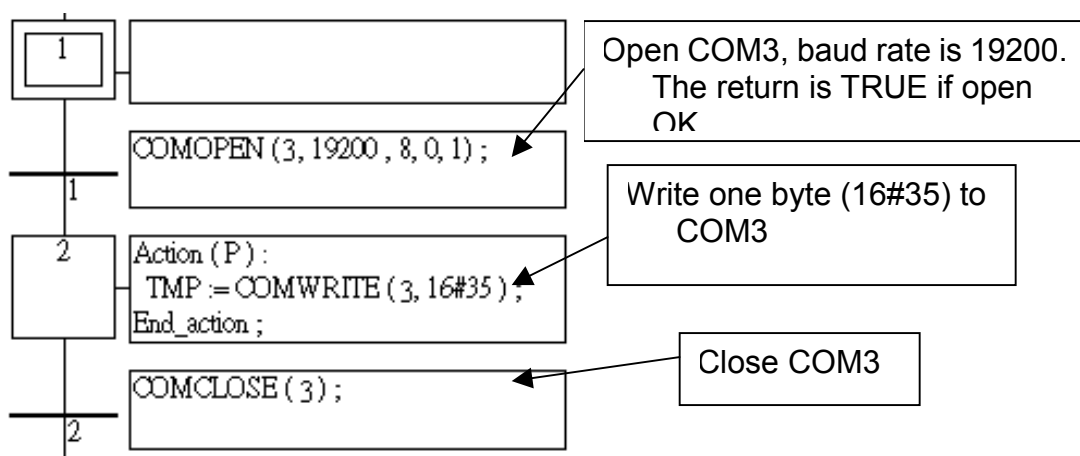


Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

Example:

Refer to Chapter 11 - Demo_21, 22 & 23.



COMOPEN2

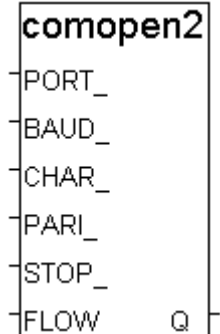
□ I-8417/8817 □ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Open COM port with flow control, for RS232 port only**

Argument:

PORT_	integer	I-7188EG/XG:3~8, W-8xx7:2, or 5 to 14
BAUD_	integer	baud rate, can be 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	integer	character size, can be 7 or 8
PARI_	integer	parity, can be 0: none, 1: even, 2: odd, 3: mark, 4: space 3 & 4 is only for I-8xx7: COM3 ~ 20, I-7188EG/XG: COM3 ~ 8. While COM2, or ... for Wincon-8xx7
STOP_	integer	stop bit, can be 1 or 2
FLOW_	boolean	True: flow control by hardware(CTS / RTS) (7188EG/XG 3 ~ 5), False: by software (XON / XOF) (7188EG/XG 3 ~ 8)
Q_	boolean	OK. return TRUE



* If Target is W-8xx7, please make sure its COM2 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)

COMREAD

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

comread
PORT_DATA

Description:

Function **Read one byte from a COM port**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
Q_	integer	the data returned

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available
- * **Call COMREADY to test data coming or not . If there is data, COMREAD & COMARY_R can be used to read the data. If no data coming, do not call COMREAD & COMARY_R, or COM port will block.**

Example:

Refer to "COMREADY" example.

COMREADY

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Test a COM port for data**

Argument:

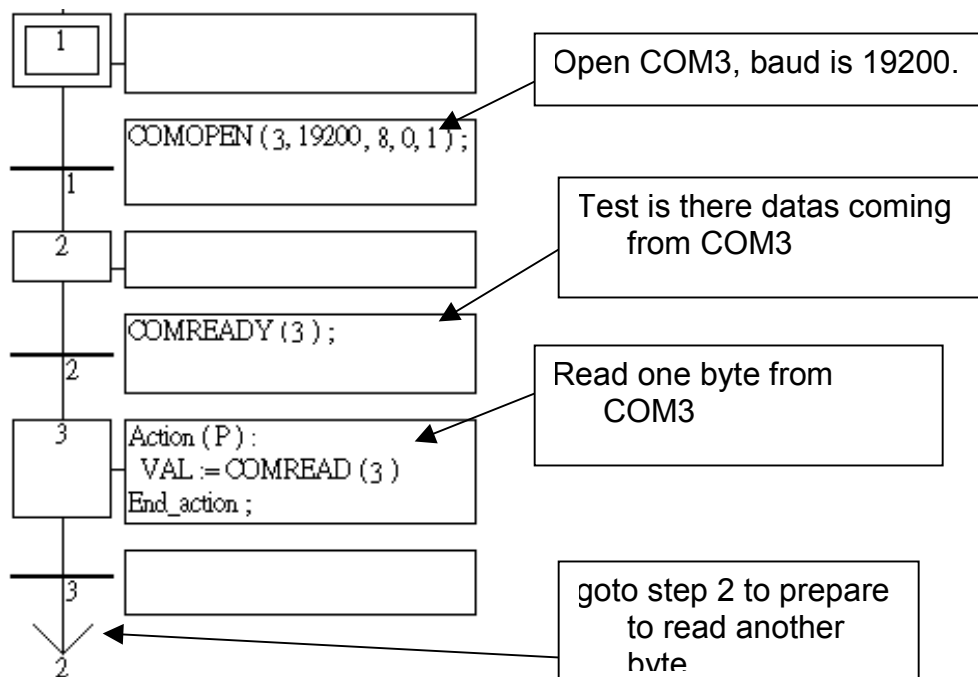
PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
Q_	boolean	If there is data coming, return TRUE. Else, return FALSE.

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available
- * **Call COMREADY to test data coming or not . If there is data, COMREAD & COMARY_R can be used to read the data. If no data coming, do not call COMREAD & COMARY_R, or COM port will block.**

Example:

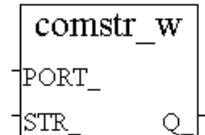
Refer to Chapter 11 - Demo_21, 22 & 23.



COMSTR_W

Argument:

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function

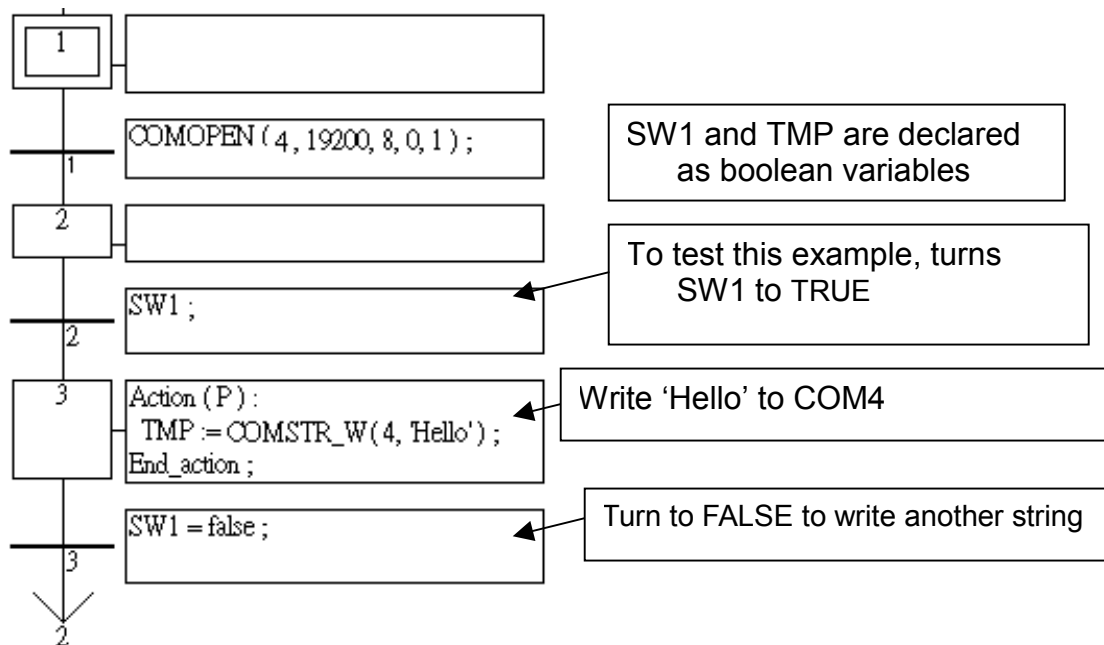
Write one string to a COM port

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
STR_	Message	the string to be written (max length is 255).
Q_	boolean	Ok. return TRUE, else return FALSE.

Note:

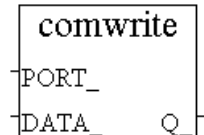
- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For I-8xx7:
 - 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 is not available

Example:



COMWRITE

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Write one byte to a COM port

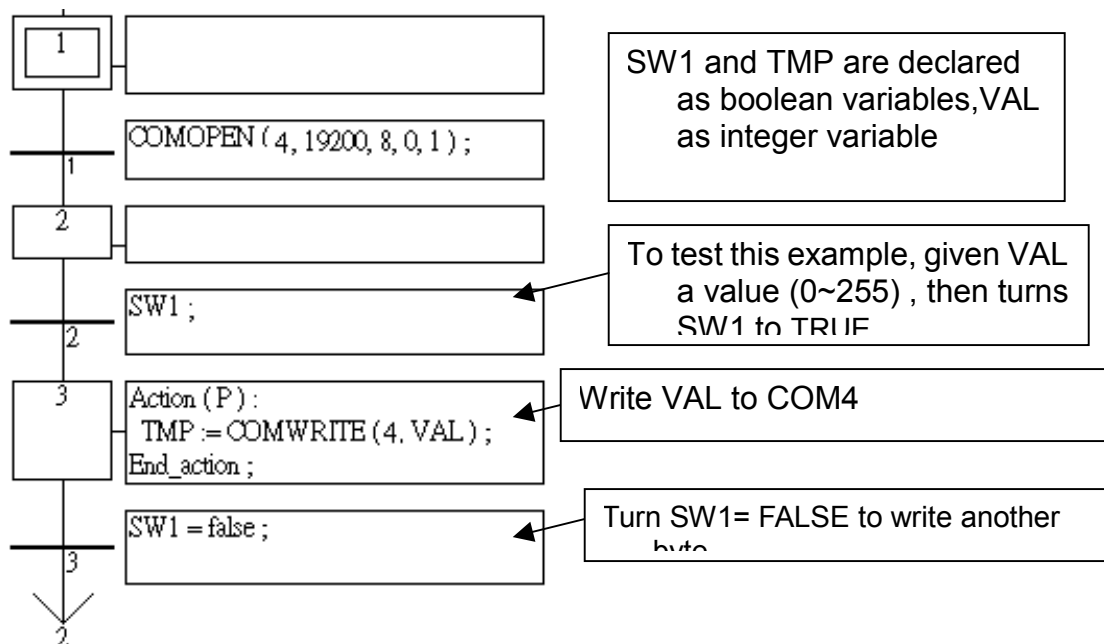
Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or ...
DATA_	integer	the byte to be written, valid range values from 0 ~ 255.
Q_	boolean	Ok. return TRUE, else return FALSE.

Note:

- * If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
- * If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
- * For 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 is not available

Example:

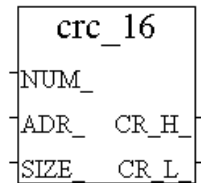


CRC_16

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:
Function Block

Calculate checksum - CRC-16

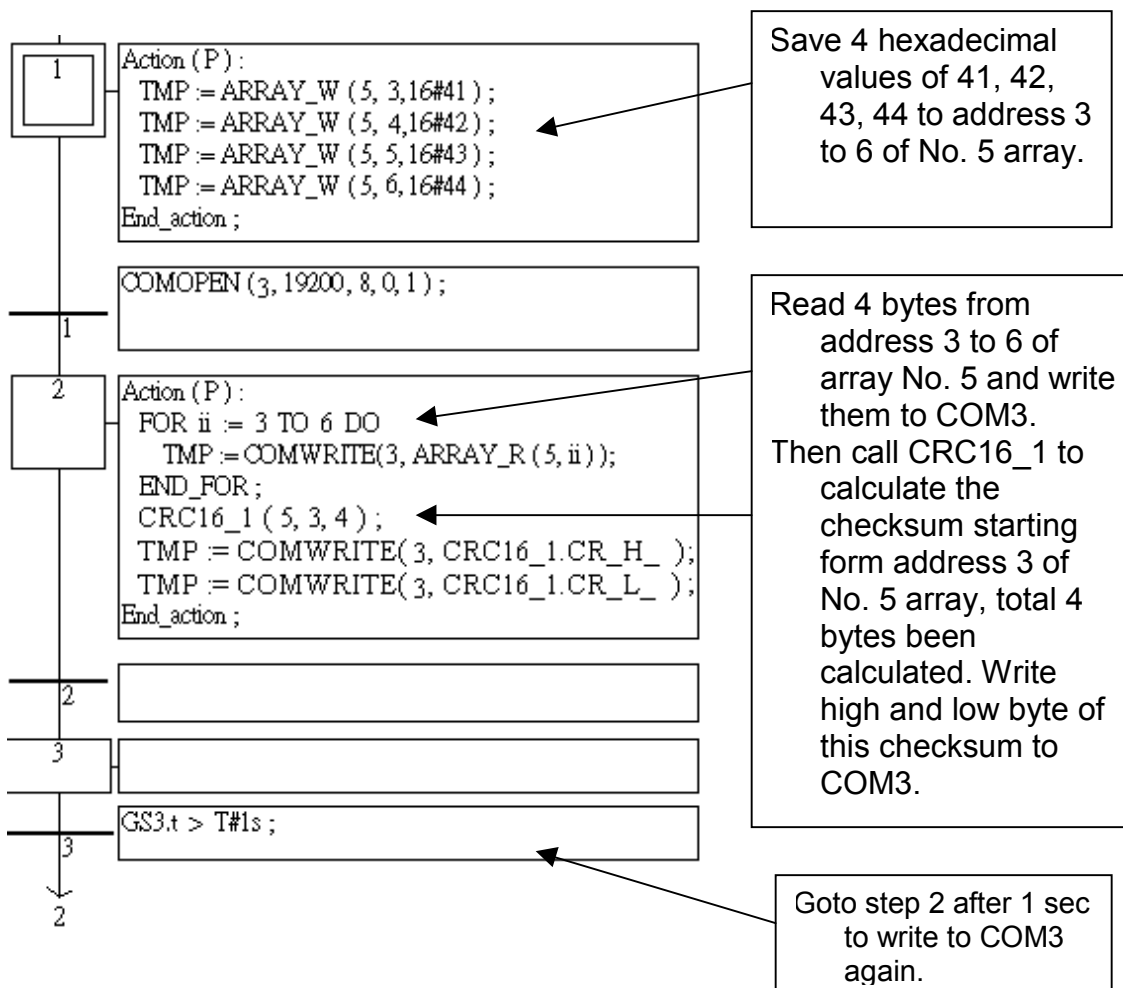


Argument:

NUM_	integer	byte array ID to be operated, Valid range for I-8xx7 & I-7188EG/XG is 1 to 24, for W-8xx7 is 1 to 48
ADR_	integer	starting address in the array which is to be calculated
SIZE_	integer	the number of bytes to be calculated
CR_H_	integer	the returned high byte of the CRC-16 after calculation.
CR_L_	integer	the returned low byte of the CRC-16 after calculation.

Example:

TMP is declared as a boolean. ii, CR_H_ and CR_L_ as integers, CRC16_1 is declared as FB instance of type – CRC_16.



DI_CNT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Get parallel D/I counter at I-8xx7 & I-7188EG/XG's slot 0 , W-8xx7 slot 1

Please refer to Section 3.8

EBUS_B_R

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

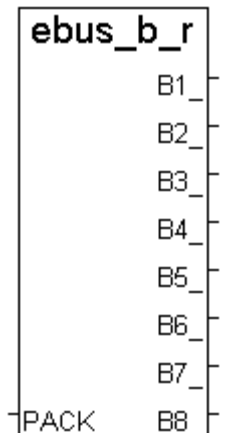
Function block Read a boolean package from the Ebus device

Arguments:

PACK_ integer which package No. to read (1 - 128)
B1_ ~ B8_ boolean the 8 boolean values contained in the package

Example:

Refer to Section 7.5



EBUS_B_W

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

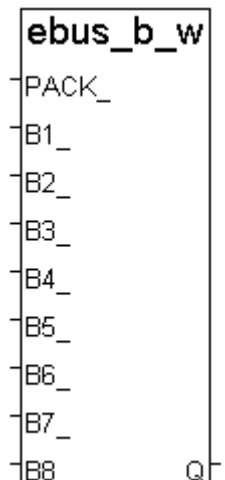
Function block Write a boolean package to the Ebus device

Arguments:

PACK_ integer write to which package No. (1-128)
B1_ ~ B8_ boolean the 8 boolean values contained in the package
Q boolean always return TRUE.

Example:

Refer to Section 7.5



EBUS_N_R

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

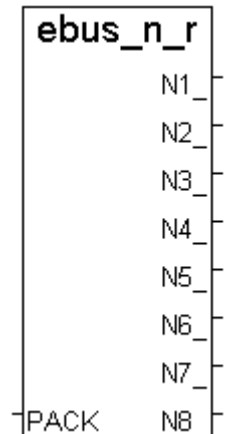
Function block **Read a integer package from the Ebus device**

Arguments:

PACK_ integer which package No. to read. (1-128)
N1_ ~ N8_ integer the 8 integer values contained in the package

Example:

Refer to Section 7.5



EBUS_N_W

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

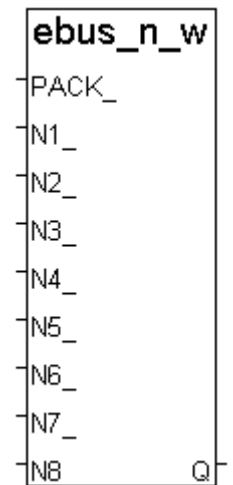
Function block **Write a integer package to the Ebus device**

Arguments:

PACK_ integer write to which package No. (1-128)
N1_ ~ N8_ boolean the 8 integer values contained in the package
Q boolean always return TRUE.

Example:

Refer to Section 7.5



EBUS_STS

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Get Package Status of Ebus**

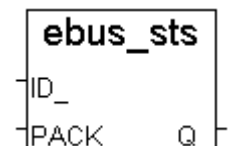
Arguments:

ID_ **Integer** to get what ? 0: Boolean package , 1: Integer package
PACK_ **Integer** get which package No. (1-128)

return:

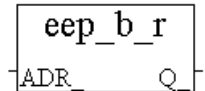
Q_ **boolean** TRUE: package is alive, FALSE: dead (communication break)

Example: Please refer to demo_49a & demo_49b



EEP_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **read a boolean value from the EEPROM**

Argument:

ADR_	integer	address in the EEPROM where the boolean value is stored, I-8xx7, 7188EG/XG: 1 ~ 256 , W-8xx7: 1 ~ 1024
Q_	boolean	the boolean value returned

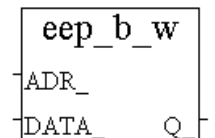
* Read operation of the EEPROM can be used freely without to remove the protection.

* Be careful to use EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W, the EEPROM can only to be written up to 100,000 times.

Example: refer to demo_17

EEP_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **write a boolean value to the EEPROM**

Arguments:

ADRES_	integer	address in the EEPROM where the boolean value is to be written to. I-8xx7, 7188EG/XG: 1 ~ 256 , W-8xx7: 1 ~ 1024
DATA_	Boolean	the boolean value to be written to
Q_	Boolean	Ok. return TRUE.

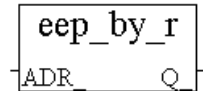
* To write to the EEPROM, the protection must be removed in advance

* Be careful to use EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W, EEPROM can only to be written up to 100,000 times.

Example: refer to demo_17

EEP_BY_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

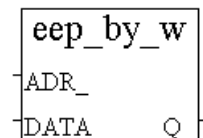
Function read a byte (8-bit integer) value from the EEPROM

Argument:

ADR_	integer	address in the EEPROM where the byte value is stored. I-8xx7, 7188EG/XG:1 ~ 1512 , W-8xx7: 1 ~ 14272
Q_	integer	the byte value returned (0~255)

EEP_BY_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function write a byte (8-bit integer) value to the EEPROM

Arguments:

ADR_	integer	address in the EEPROM where the byte value is to be written to. I-8xx7, 7188EG/XG:1 ~ 1512 , W-8xx7: 1 ~ 14272
DATA_	integer	the byte value to be written to, valid range values from 0 to 255.
Q_	Boolean	Ok. return TRUE.

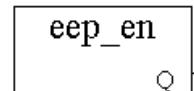
Note:

- * If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_N_R, and EEP_N_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.
- * Read operation of the EEPROM will work without removing the EEPROM protection.
- * 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.

Example: refer to demo_17

EEP_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Remove the EEPROM write protection

Argument:

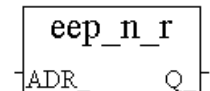
Q_ Boolean Ok: return TRUE, Fail: return FALSE

* BEFORE writing to the EEPROM, the EEPROM write protection must be turned off.

* 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.

EEP_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function read an signed 32-bit integer value from the EEPROM

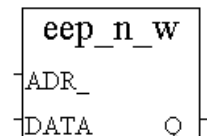
Argument:

ADR_ integer address in the EEPROM where the 32-bit integer value is stored.
I-8xx7, 7188EG/XG: 1 ~ 378 , W-8xx7: 1 ~ 3568

Q_ integer the signed 32-bit integer value returned

EEP_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function write a signed 32-bit integer value to the EEPROM

Arguments:

ADR_ integer address in the EEPROM where the 32-bit integer value is to be written to , I-8xx7, 7188EG/XG: 1 ~ 378 , W-8xx7: 1 ~ 3568

DATA_ integer the 32-bit integer value to be written to

Q_ Boolean Ok. return TRUE.

Note:

* 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.

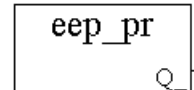
* Read operation of the EEPROM will work without removing the EEPROM protection.

* 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.

Example: refer to demo_17

EEP_PR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Set the EEPROM write protection**

Argument:

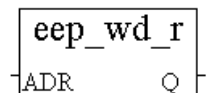
Q_ Boolean Ok: return TRUE, Fail: return FALSE

* After writing to an EEPROM, it is better to turned off the write protection.

* 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.

EEP_WD_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **read a word (signed 16-bit integer) value from the EEPROM**

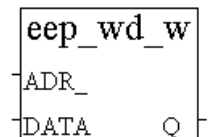
Argument:

ADR_ integer address in the EEPROM where the word value is stored.
I-8xx7,7188EG/XG: 1 ~ 756 , W-8xx7: 1 ~ 7136

Q_ integer the word value returned (-32768 ~ +32767)

EEP_WD_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **write a word (signed 16-bit integer) value to the EEPROM**

Arguments:

ADR_ integer address in the EEPROM where the word value is to be written to.
I-8xx7,7188EG/XG: 1 ~ 756 , W-8xx7: 1 ~ 7136

DATA integer the word value to be written to, range from -32768 to +32767.

Q_ Boolean Ok. return TRUE.

Note:

* If you are using this function with the EEP_N_R, EEP_N_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.

* Read operation of the EEPROM will work without removing the EEPROM protection.

* 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.

Example: refer to demo_17

EMAIL

■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8XX7/W-8XX6

Description:

Function Block **Send an email**

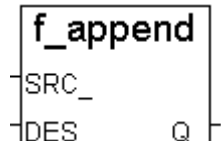
Please refer to Chapter 12 – “Sending Emails” .

F_APPEND

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Append one file to the other file**



Argument:

SRC_	message	File path of source file. For ex '\CompactFlash\data.txt'
DES_	message	File path of destination file. For ex '\CompactFlash\data1.txt'
Q_	boolean	True: Ok, False: error

Note:

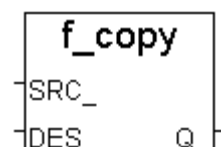
1. If one of thses two files is not found, return FALSE.
2. Source and Destination file should be close, not open.
3. Please refer to ISaGRAF standard function F_wopen , F_ropen , F_close , F_eof , Fa_read , Fa_write (F_wopen: Open existing file for Read & Write)
4. Please refer to F_creat , F_copy , F_append , F_dir , F_end , F_seek , F_writ_b , F_writ_f , F_writ_s , F_writ_w

F_COPY

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Copy one file to another one**



Argument:

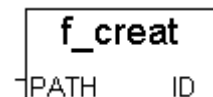
SRC_	message	File path of source file. For ex '\data.txt'
DES_	message	File path of destination file. For ex '\CompactFlash\data1.dat'
Q_	boolean	True: Ok, False: error

Note:

1. Copy SRC_ to DES_
2. SRC must exist.
3. If there is an old DES_ file existing, it will be replaced by new contents.
4. SRC_ & DES_ file should be closed.

F_CREAT

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Creat an empty file for Reaing & Writing**

Argument:

PATH_	message	File path and name. For ex 'CompactFlash\data.txt'
ID_	Integer	file ID returned, if error happens, it returns 0

Note:

1. If file is already exist, it will be destroyed.
2. 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)
3. 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

Example: Wdemo_11 & Wdemo_12 in W-8x37 CD-ROM:\napdos\isagraf\wincon\demo\

F_DELETE

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

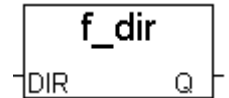
Function **Delete a file**

Argument:

NAME_	message	File name, for ex, 'CompactFlash\Temp\HTTP\Data\alarm.txt'
Q_	boolean	True: Ok , False: fail or file doesn't exists

F_DIR

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Create a directory**

Argument:

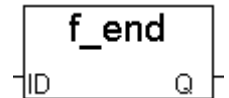
DIR_	message	directory name, For example '\DATA111'
Q_	boolean	TRUE: ok , FALSE: error happens (For ex. path is not correct or the directory already exists)

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('\DATA111') ;
end_if;
```

F_END

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Move file position to End-Of-File**

Argument:

ID_	integer	File ID No. returned by F_ROPEN() , F_CREAT() or F_WOPEN()
Q_	boolean	True: Ok , False: fail

Note: Please use F_SEEK to move to a specified file position

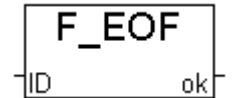
Example: Wdemo_11 & Wdemo_12 in W-8x37 CD-ROM:\napdos\isagraf\wincon\demo\

Note:

1. 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)
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_writ_s

F_EOF

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Tests if End Of File has been reached (ISaGRAF Standard Function)

Argument:

ID_	integer	File ID No. returned by F_ROPEN(), F_CREAT() or F_WOPEN()
OK	boolean	True: reach End Of File , False: not yet

F_READ_B

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

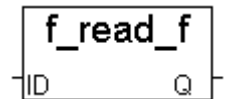
Function Read one byte from current position of an open file

Argument:

ID_	integer	File ID No. returned by F_ROPEN(), F_CREAT() or F_WOPEN()
Q_	integer	the returned byte (0 - 255)

F_READ_F

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Read one float value from current position of an open file

Argument:

ID_	integer	File ID No. returned by F_ROPEN(), F_CREAT() or F_WOPEN()
Q_	real	the returned float value (32-bit float format)

Note:

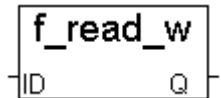
1. This function may cause Local Controller fault if the REAL value hasn't been well set yet.
2. Please refer to Section 10.6 for "Controller Fault Detection"

Note:

1. 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)
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_writ_s

F_READ_W

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

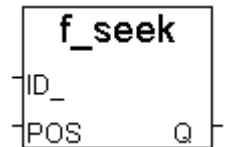
Function **Read one word (signed 16-bit integer) from current position of an open file.**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
Q_	integer	the returned word (-32768 ~ +32767)

F_SEEK

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

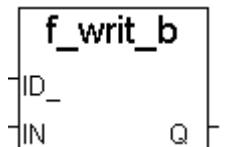
Function **Move file position to ...**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
POS_	integer	position, unit is byte (1 to ...)
Q_	boolean	True: Ok , False: fail

F_WRIT_B

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Write one byte to current position of an open file**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	integer	The byte value to write, 0 ~ 255. if value > 255 or <0, the lowest byte is written
Q_	boolean	True: Ok , False: fail

Note:

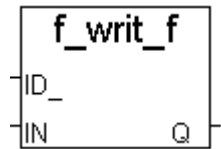
1. Using ISaGRAF Standard Function –“**FA_READ**” & “**FA_WRITE**” to R/W **long** integer
2. Using ISaGRAF Standard Function –“**FM_READ**” , “**F_WRIT_S**” & “**FM_WRITE**” to R/W message (string)
3. The “**FM_WRITE**” writes <CR><LF> at the end of the message, while “**F_WRIT_S**” doesn’t.

Example:

Refer to Wincon CD:\napdos\isagraf\wincon\demo\ “wdemo_01 & wdemo_02”

F_WRIT_F

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

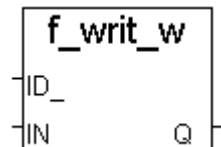
Function Write one float value (32-bit format) to current position of an open file

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	real	The float value to write, (32-bit format)
Q_	boolean	True: Ok , False: fail

F_WRIT_W

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

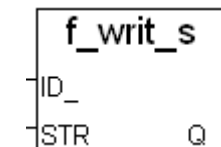
Function Write one word value (signed 16-bit integer) to current position of an open file

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	integer	The word value to write, (-32768 ~ +32767)
Q_	boolean	True: Ok , False: fail

F_WRIT_S

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Write one string to current position of an file without <CR> <LF> at the end

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
STR_	message	Message (String) to write.
Q_	boolean	True: Ok , False: fail

Note:

1. Using ISaGRAF Standard Function –“**FA_READ**” & “**FA_WRITE**” to R/W **long** integer
2. Using ISaGRAF Standard Function –“**FM_READ**” , “**F_WRIT_S**” & “**FM_WRITE**” to R/W message (string)
3. The “**FM_WRITE**” writes <CR><LF> at the end of the message, while “**F_WRIT_S**” doesn't.

Example:

Refer to Wincon CD:\napdos\isagraf\wincon\demo\ “wdemo_01 & wdemo_02”

FBUS_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

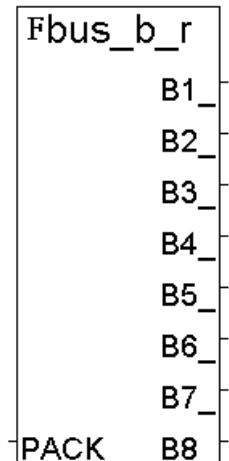
Function block **Read a boolean package from the Fbus device**

Arguments:

PACK_ integer which package No. to read. (1-128)
B1_ ~ B8_ boolean the 8 boolean values contained in the package

Example:

Refer to Chapter 7 or demo_11a.



FBUS_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

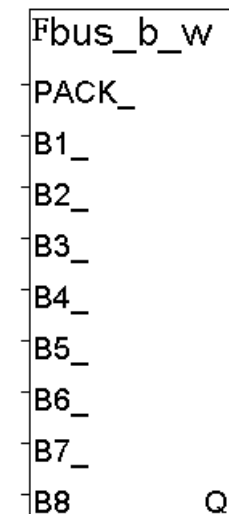
Function block **Write a boolean package to the Fbus device**

Arguments:

PACK_ integer write to which package No. (1-128)
B1_ ~ B8_ boolean the 8 boolean values contained in the package
Q boolean always TRUE.

Example:

Refer to Chapter 7 or demo_11b.



FBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

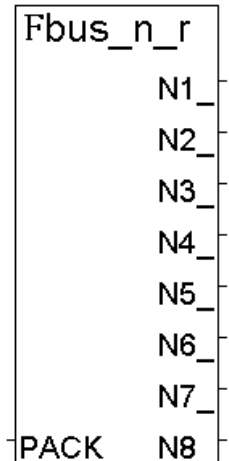
Function block **Read an integer package (signed 32-bit) from the Fbus device**

Arguments:

PACK_ integer which package No. to read. (1-128)
N1_ ~ N8_ integer the 8 integer values contained in the package

Example:

Refer to Chapter 7 or demo_11b.



FBUS_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

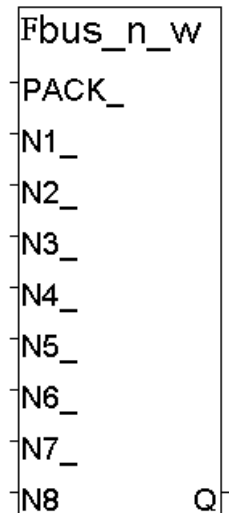
Function block **Write an integer package (signed 32-bit) to the Fbus device**

Arguments:

PACK_ integer write to which package No. (1-128)
N1_ ~ N8_ boolean the 8 integer values contained in the package
Q boolean always TRUE.

Example:

Refer to Chapter 7 or demo_11a.



FBUS_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function **Get Package Status of Fbus**

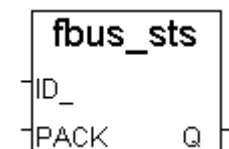
Arguments:

ID_ **Integer** to get what ? 0: Boolean package , 1: Integer package
PACK_ **Integer** get which package No. (1-128)

return:

Q_ **boolean** TRUE: package is alive, FALSE: dead (communication break)

Example: Please refer to demo_49a & demo_49b



GET_SN

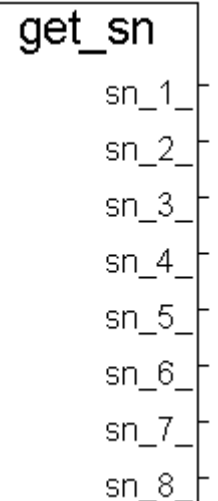
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block **get hardware unique serial No.**

Arguments:

Sn_1_ ~ 8 Integer the returned serial No. 8 bytes. (-128 to +127)



INP10LED

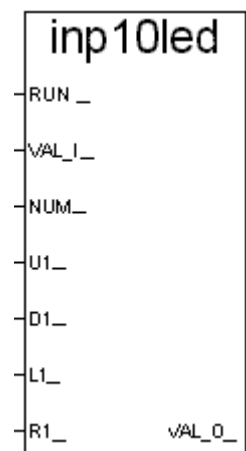
■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8XX7/W-8XX6

Description:

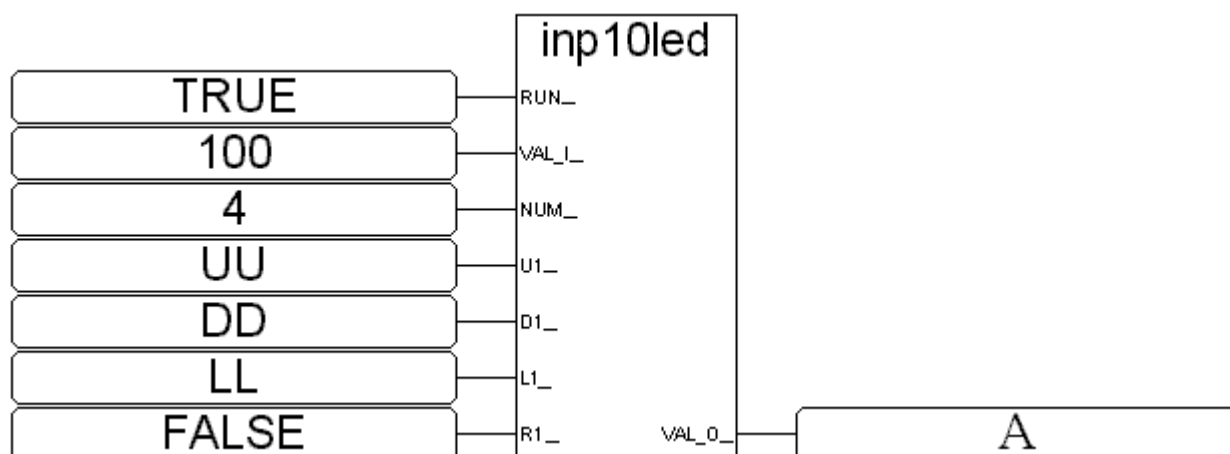
Function **input an decimal integer from the S_MMI**

Arguments:

RUN_	Boolean	When "TRUE", Process & Display Value To SMMI
VAL_I_	Integer	Initial Value Displayed On S-MMI, Minimum Value Is "0", maximum is 99999
NUM_	Integer	Number Of Digits To Display, Valid Range From 1 To 5
U1_	Boolean	When Rising From "FALSE" To "TRUE", Add 1 To The Currently Displayed Digit
D1_	Boolean	When Rising From "FALSE" To "TRUE", Subtract 1 From The Currently Displayed Digit
L1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Left 1 Position From Currently Displayed Digit
R1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Right 1 Position From Currently Displayed Digit
VAL_O_	integer	The Displayed Integer Value After Operation



Example: refer to demo_08, demo_11a.



ST equivalence:

```
A := INP10LED(TRUE,100,4,UU,DD,LL,FALSE);
```

(* A is declared as an integer variable *)

(* UU,DD,LL are declared as boolean variables, can be linked to "push4key" board *)

INP16LED

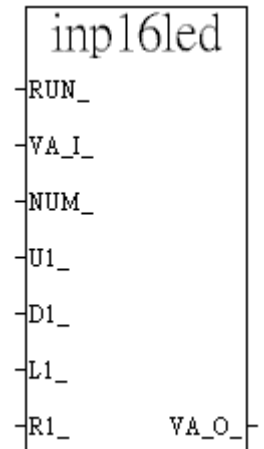
■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8XX7/W-8XX6

Description:

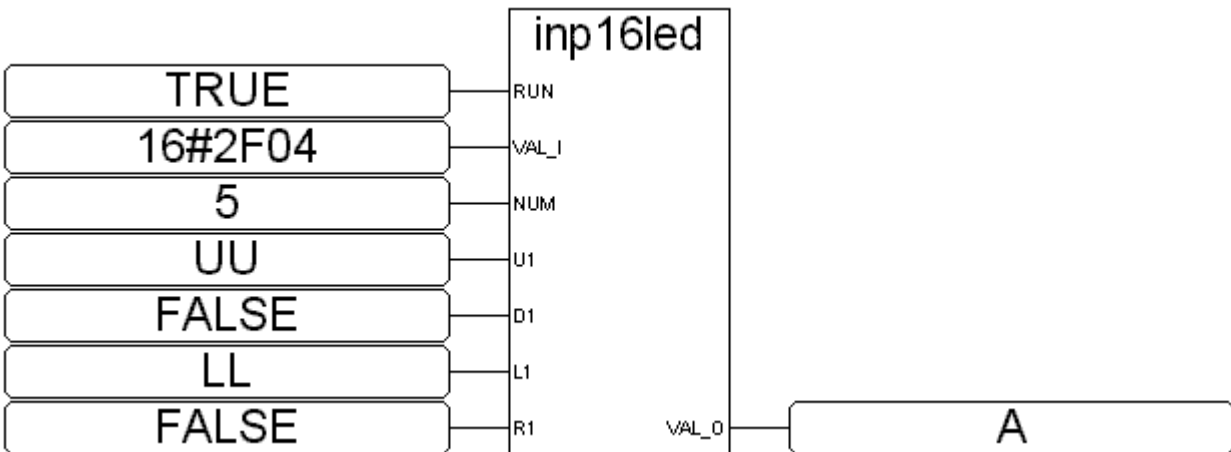
Function input an hexadecimal integer from the S_MMI

Arguments:

RUN_	Boolean	When "TRUE", Process & Display Value To S-MMI
VAL_I_	Integer	Initial Value Displayed On S-MMI, Minimum Value Is "0", maximum is 16#FFFF
NUM_	Integer	Number Of Digits To Display, Valid Range From 1 To 5
U1_	Boolean	When Rising From "FALSE" To "TRUE", Add 1 To The Currently Displayed Digit
D1_	Boolean	When Rising From "FALSE" To "TRUE", Subtract 1 From The Currently Displayed Digit
L1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Left 1 Position From Currently Displayed Digit
R1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Right 1 Position From Currently Displayed Digit
VAL_O_	integer	The Displayed Integer Value After Operation



Example:



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

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Map a long integer to a Real value.**

The algorithm in C language is `Real_ = *((float *)&Long_);`

Arguments:

Long_	integer	the 32-bit integer
Real_	real	the real value after mapping

Note: "Real_Int" can be used to map a Real value to a long integer.

If you just want to convert one integer value to real value, please use "REAL()" function.

Warning:

The "int_real(L1)" may cause controller un-stable if the parameter "L1" is not generated by a previous "L1 := real_int(R1)" command (**section 10.6**). If this situation happens (for ex, controller may automatically reset), please delete the ISaGRAF program inside controller.

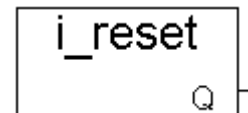
isa7188e *d= (for 7188EG, at INIT mode)

isa7188 *d= (for 7188XG, at INIT mode)

(please refer to 1.3.7 for I-8417/8817/8437/8837)

I_RESET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

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. If the controller is always reset, please refer to section 1.3.7 to delete the project inside the I-8xx7 controller.

Example:

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

I7000_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Enable/Disable Bus7000 communication**

Please refer to Section 6.4

LONG_WD

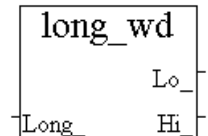
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block Convert one integer to two words

Arguments:

LONG_	integer	the 32-bit integer to be converted
LO_	integer	the low word value after the conversion, from -32768 to +32767
HI_	integer	the high word value after conversion, from -32768 to +32767



MBUS_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

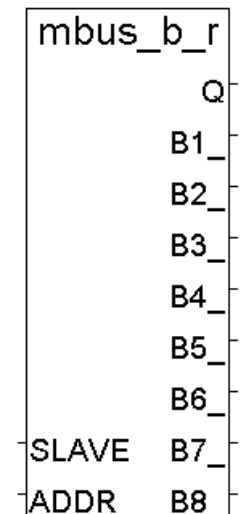
Description:

Function block Read 8 bits (booleans) from the Mdobus device

Use Modbus function code ---- 1

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255, should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
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” that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256

MBUS_BR1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

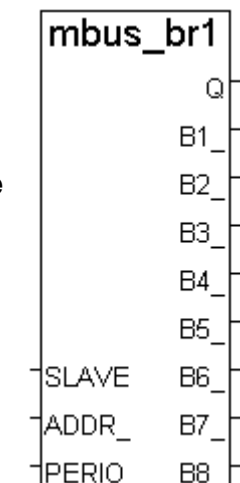
Description:

Function block Read 8 bits (booleans) from the Mdobus device with period time

Use Modbus function code ---- 1

Arguments:

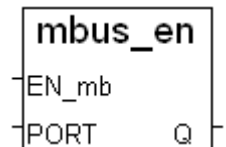
SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)
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_BR1” + “MBUS_B_R” that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256

Mbus_EN

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function This function can Enable\Disable "Mbus" or "Mbus_asc"

Arguments:

EN_mbus_	boolean	TRUE: Enable, FALSE: Disable
PORT_	integer	2 (COM2:), 3 (COM3:), 5 (COM5:) to 9 (COM9) or 10 (MSP1:), 11 (MSP2:), 12 (MSP3:), 13 (MSP4:), 14 (MSP5:)
Q_	boolean	False: PORT_ Number error , TRUE: Ok

Note:

1. Default status is Enable.
2. Only work when I/O complex equipment - "mbus" or "mbus_asc" is connected.

MBUS_B_W

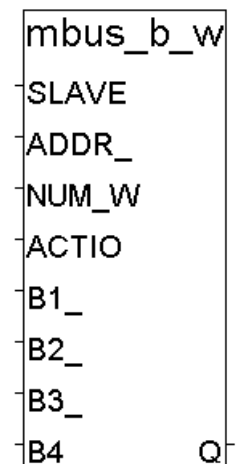
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block write 1 to 4 bits (booleans) to the Mdobus device

Use Modbus function code 5 when NUM_W = 1

Use Modbus function code 15 when NUM_W = 2 to 4



Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of bits to write, valid range from 1 to 4. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
B1_ ~ B4_	boolean	bits to write
Q_	boolean	Ok. return TRUE, else return FALSE

Note: The total number of "MBUS_B_W" + "MBUS_WB" blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256

Example:

Refer to Chapter 8 or demo_16.

MBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

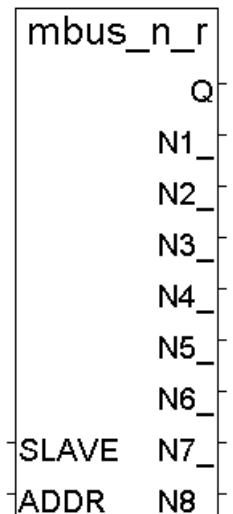
Description:

Function block **Read 8 words (16-bit integer) from the Mdbus device**

Use Modbus function code ---- 3

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N8_	integer	the 8 word values that have been read, valid range values from -32768 to +32767



Note: The total number of “MBUS_N_R” + “MBUS_R” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256.

Example:

Refer to Chapter 8 or demo_15a.

MBUS_NR1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

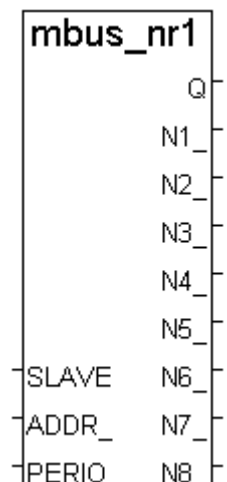
Description:

Function block **Read 8 words (16-bit integer) from the Mdbus device with period time**

Use Modbus function code ---- 3

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N8_	integer	the 8 word values that have been read, valid range values from -32768 to +32767



Note: The total number of “MBUS_N_R” + “MBUS_R” + “MBUS_NR1” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256.

MBUS_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

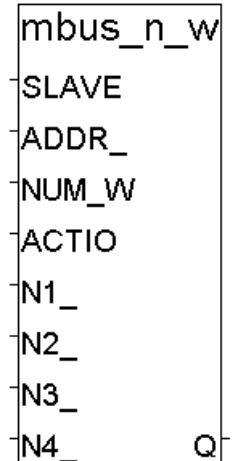
Function block **write 1 to 4 words (booleans) to the Mdobus device**

Use Modbus function code 6 when NUM_W = 1

Use Modbus function code 16 when NUM_W = 2 to 4

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of words to write, valid range values from 1 to 4. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
N1_ ~ N4_	integer	words to write (-32768 ~ 32767)
Q_	boolean	Ok. return TRUE, else return FALSE



Note: The total number of “MBUS_N_W” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256.

Example:

Refer to Chapter 8.

MBUS_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

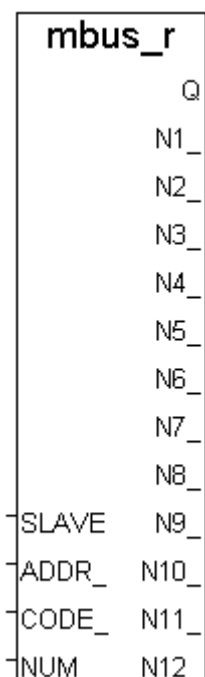
Description:

Function block **Read Modbus code 1-4 from the Modbus device**

- * ISaGRAF controller is the Master, remote equipment is Slave
- * adapt Modbus function code 1 or 2 or 3 or 4
- * please make sure the remote device support the associated Modbus function code

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
CODE_	integer	Request which Modbus function codes, 1-4. , should be constant value not variable.
NUM_	integer	Request how many bits? 1-192 for code 1 & 2 or How many words? 1-12 for code 3 & 4. , should be constant value not variable.
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N12_	integer	The bits or words received. If CODE_ is 1 & 2, N1_ returns bit 1 to 16, N2_ returns bit 17 to 32, ... N12_ returns bit 177 to 192. If CODE_ is 3 & 4, N1_ to N12_ returns the associated words (-32768 to 32767). N1_ to N12_ is absolutely correct Only when Q return TRUE (comm. ok)



Note: The total number of “MBUS_N_R” + “MBUS_R” + “MBUS_R1” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256.

MBUS_R1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

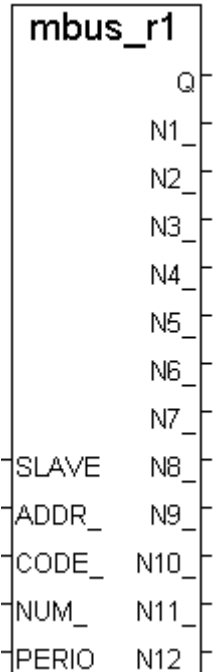
Function block **Read Modbus code 1-4 from the Modbus device with period time**

- * ISaGRAF controller is the Master, remote equipment is Slave
- * adapt Modbus function code 1 or 2 or 3 or 4
- * please make sure the remote device support the associated Modbus function code

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.	SLAVE	N8_
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.	ADDR_	N9_
CODE_	integer	Request which Modbus function codes, 1-4. , should be constant value not variable.	CODE_	N10_
NUM_	integer	Request how many bits? 1-192 for code 1 & 2 or How many words? 1-12 for code 3 & 4 . , should be constant value not variable.	NUM_	N11_
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)	PERIO	N12_
Q_	boolean	Ok. return TRUE, else return FALSE		
N1_ ~ N12_	integer	The bits or words received. If CODE_ is 1 & 2, N1_ returns bit 1 to 16, N2_ returns bit 17 to 32, ... N12_ returns bit 177 to 192. If CODE_ is 3 & 4, N1_ to N12_ returns the associated words (-32768 to 32767). N1_ to N12_ is absolutely correct Only when Q return TRUE (comm. ok)		

Note: The total number of “MBUS_N_R” + “MBUS_R” + “MBUS_R1” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256



MBUS_WB

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

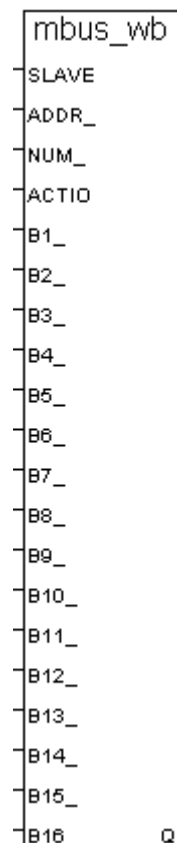
Function block write 1 to 16 bits (booleans) to the Mdobus device

Use Modbus function code 15

Arguments:

SLAVE_	integer	slave No. of the Modbus device, range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of bits to write, valid range from 1 to 16. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
B1_ ~ B16_	boolean	bits to write
Q_	boolean	Ok. return TRUE, else return FALSE

Note: The total number of “MBUS_B_W” + “MBUS_WB” blocks that can be used in one ISaGRAF project is up to I-8xx7 & I-7188EG/XG:64 , W-8xx7: 256.



MI_BOO

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Display a boolean value on MMICON**



Arguments:

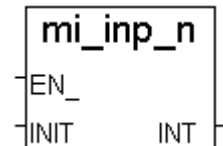
X_	integer	X position, 1-30
Y_	integer	Y position, 1-8
BOO_	boolean	boolean value to display. TRUE display "ON", FALSE display "OFF"
Q_	boolean	Ok. return TRUE, else return FALSE

MI_INP_N

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Input an integer value from MMICON**



Arguments:

EN_	boolean	TRUE: enable
INIT_	integer	Initial value to input
INT_	integer	The integer value been input. 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 fail.

Demo:

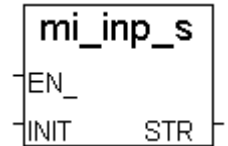
Please refer to Chapter 16 & demo_38, demo_39

MI_INP_S

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Input an string from MMICON**



Arguments:

EN_	boolean	TRUE: enable
INIT_	message	Initial string value to input
STR_	message	The string been input. 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 fail.

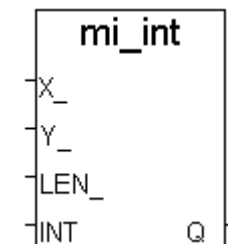
To input a real value, please use MI_INP_S, STR_REAL & REAL_STR and refer to demo_39.

MI_INT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Display an Integer value on MMICON**



Arguments:

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.
Q_	boolean	Ok. return TRUE, else return FALSE

Demo:

Please refer to Chapter 16 & demo_38, demo_39

MI_REAL

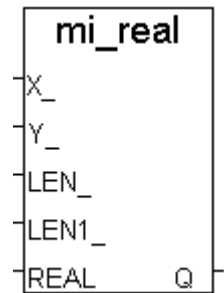
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Display a real value on MMICON**

Arguments:

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_. For ex. if LEN_=7, LEN1_=2, "123.4567" will be displayed as " 123.45"
REAL_	real	real value to display. If the number of digits exceeds LEN_, '*****' will be displayed
Q_	boolean	Ok. return TRUE, else return FALSE



Note:

If abs. of the real value $\geq 1,000,000$ or (> 0 & < 0.0001), please give LEN_ as 13 to display for 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

MI_STR

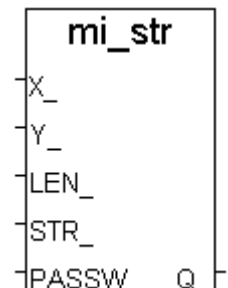
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Display a string on MMICON**

Arguments:

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.
Q_	boolean	Ok. return TRUE, else return FALSE



Demo:

Please refer to Chapter 16 & demo_38, demo_39

MSGARY_R

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Read a string from message array**

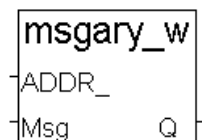
Arguments:

ADDR_	integer	which address, 1 - 1024
Msg_	message	the string returned (Len is between 0 to 255)

Example: Please refer to Chapter 11 - Wincon-8xx7's demo "wdemo_06"

MSGARY_W

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Write a string to message array**

Arguments:

ADDR_	integer	which address, 1 - 1024
Msg_	message	the string to write (Len is between 0 to 255)
Q_	boolean	True: Ok, False: Fail

Example: Please refer to Chapter 11 - Wincon-8xx7's demo "wdemo_06"

PID_AL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Example:

Please refer to Chapter 11 - Demo_18, and ICP DAS CD-ROM :

\\Napdos\\ISaGRAF\\8000\\English_Manual\\PID_AL.Complex PID algorithm implementation.htm

PWM_DIS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Disable PWM output**

PWM_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Enable PWM output.**

PWM_EN2

Description:

Function **Enable PWM output to output some pulse.**

Please refer to Section 3.7 .

PWM_ON

Description:

Function **Set parallel D/O to TRUE immediatelly**

PWM_OFF

Description:

Function **Set parallel D/O to FALSE immediatelly**

PWM_SET

Description:

Function **Dynamically change the ON_, OFF_ & NUM_ setting**

PWM_STS

Description:

Function **Get PWM status**

PWM_STS2

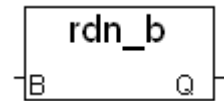
Description:

Function **Get the pulse number which has been output by “pwm_en2” or “pwm_en”**

Please refer to Section 3.7.

RDN_B

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Set a Boolean variable as Redundant synchronous data.**

RDN_F

Description:

Function **Set a REAL variable as Redundant synchronous data.**

RDN_N

Description:

Function **Set an Integer variable as Redundant synchronous data.**

RDN_T

Description:

Function **Set a Timer variable as Redundant synchronous data.**

Notes:

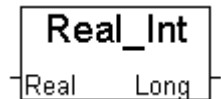
1. The ISaGRAF program for redundant Master & Slave controllers are the same.
2. All I-7K & I-87K function blocks should be located on the top of the ISaGRAF project. Please refer to Example program: Wdmo_18
3. Before using RDN_B, RDN_N, RDN_F & RDN_T functions, please make sure the below "IO complex equipment" is well connected in the ISaGRAF IO connection window.
"rdn" : set this controller as Redundant controller.
4. RDN_B, RDN_N, RDN_F & RDN_T must be called only in the 1st PLC scan.
5. RDN_B : Set a Boolean variable to be a Redundant data
6. RDN_N : Set an Integer variable to be a Redundant data
7. RDN_F : Set a REAL variable to be a Redundant data
8. RDN_T : Set a Timer variable to be a Redundant data
9. Max number of synchronous bytes is 6000. Boolean: 1 byte, Integer & Real & Timer is 4 byte. (sum of RDN_B, RDN_N, RDN_F, RDN_T)
10. **Please refer to Chapter 20 for more information.**

Example: Wdmo_18 located at

1. W-8x37/8x36/8x47/8x46 CD-ROM:\napdos\isagraf\wincon\demo\
2. ftp.icpdas.com.tw/pub/cd/winconcd/napdos/isagraf/wincon/demo/

REAL_INT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Map a Real value to a long integer.**

The algorithm in C language is `Long_ = *((long *)&Real_);`

Arguments:

Real_	real	the real value to map
Long_	integer	the 32-bit integer after mapping

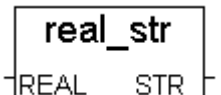
Note:

"Int_Real" can be used to map a long integer to a Real value.

If you just want to convert one real value to integer value, please use "ANA()" function.

REAL_STR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Convert a Real value to a string.**

Arguments:

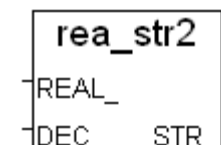
REAL_	real	the real value to convert
STR_	message	the string returned (Max length is 13), For ex. 1.234 ---> '1.234' 123456789.0 ---> '1.23457E+008' 0.00001234 ---> '1.234E-005'

Note:

"STR_REAL" can be used to convert a string to a Real value.

REA_STR2

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Convert a Real value to a string with fixed digit number**

Arguments:

REAL_	real	the real value to convert
DEC_	Integer	The digit number after the dot "." symbol (0 - 5)
STR_	message	the string returned (Max length is 13), For ex.

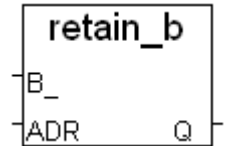
For ex. if DEC_ = 2
1.236 ---> '1.24'
123456.0 ---> '123456.00'
0.00001234 ---> '0.00'

Retain_B

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Set a Boolean variable to be as "retained variable"



Arguments:

B_	Boolean	B_ should be a boolean variable name, not a constant value.
ADR_	Integer	Set retained address for this boolean variable. I-8xx7 & I-7188EG/XG: 1 to 256 , W-8xx7: 1 to 1024

return:

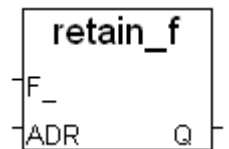
Q_	Boolean	Ok return True, Error return False
-----------	---------	------------------------------------

Retain_F

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Set a Real variable to be as "retained variable"



Arguments:

F_	Real	F_ should be a Real variable name, not a constant value.
ADR_	Integer	Set retained address for this Real variable. I-8xx7 & I-7188EG/XG: 1 to 1024 , W-8xx7: 1 to 4096

return:

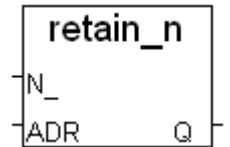
Q_	Boolean	Ok return True, Error return False
-----------	---------	------------------------------------

Retain_N

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Set an Integer variable to be as "retained variable"



Arguments:

N_	Integer	N_ should be a Integer variable name, not a constant value.
ADR_	Integer	Set retained address for this Integer variable. I-8xx7 & I-7188EG/XG: 1 to 1024 , W-8xx7: 1 to 4096

return:

Q_	Boolean	Ok return True, Error return False
-----------	---------	------------------------------------

Note:

1. Befor using Retain_X, Retain_N , Retain_B , Retain_F & Retain_T functions, please make sure the below "IO complex equipment" is well connected in the ISaGRAF IO connection window.

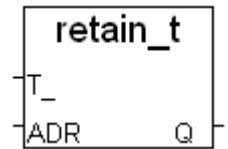
I-7188EG/XG : "X607_608"

I-8417/8817/8437/8837 & W-8XX7/W-8XX6 : "S256_512"

2. Please refer to Chapter 10 for detail information

Retain_T

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Set a Timer variable to be as "retained variable"

Arguments:

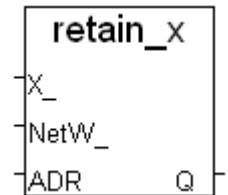
T_	Timer	T_ should be a Timer variable name, not a constant value.
ADR_	Integer	Set retained address for this Integer variable. I-8xx7 & I-7188EG/XG: 1 to 256 , W-8xx7: 1 to 1024

return:

Q_	Boolean	Ok return True, Error return False
-----------	---------	------------------------------------

Retain_X

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Set a variable to be as "retained variable" by using its Network address No

Arguments:

X_	Message	'B' : boolean , 'N' : Integer , 'F' : Real , 'T' : Timer
NetW_	Integer	Network address No. for the related variable. I-8xx7 & I-7188EG/XG : 1 to 4095 W-8xx7 : 1 to 8191
ADR_	Integer	Set retained address for this Integer variable. I-8xx7 & I-7188EG/XG: 1 to 1024 , W-8xx7: 1 to 4096

return:

Q_	Boolean	Ok return True, Error return False
-----------	---------	------------------------------------

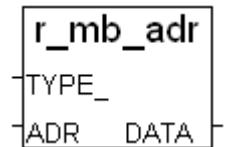
Note:

1. Before using Retain_X, Retain_N , Retain_B , Retain_F & Retain_T functions, please make sure the below "IO complex equipment" is well connected in the ISaGRAF IO connection window.
I-7188EG/XG : "X607_608"
I-8417/8817/8437/8837 & W-8XX7/W-8XX6 : "S256_512"

2. Please refer to Chapter 10 for detail information

R_MB_Adr

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Read boolean or integer variable by using Network address No

Arguments:

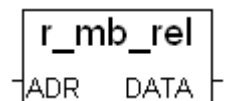
TYPE_ Integer 0: boolean variable , 1: integer variable
ADR_ Integer read which Network address
I-8xx7 & I-7188EG/XG: 1 to 4095 , W-8xx7: 1 to 8191

return:

DATA_ Integer the read value (if TYPE is boolean, 1 means True, 0 means False)

R_MB_Rel

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Read REAL variable by using Network address No

Arguments:

ADR_ Integer read which Network address
I-8xx7 & I-7188EG/XG: 1 to 4095 , W-8xx7: 1 to 8191

return:

DATA_ Real the real value been read

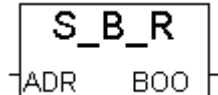
Note: Please refer to Section 11.3.5 for a demo description

1. Please use R_MB_REL function to read "REAL" variable. Please also refer to W_MB_Adr & W_MB_Rel
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 No. Please refer to section 4.2 of "User's Manual Of The ISaGRAF Embedded Controllers".

Example: Wincon:Wdemo_24 & I-8xx7:demo_70

S_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one boolean from the volatile SRAM

Arguments:

ADR_ Integer read which address, 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)

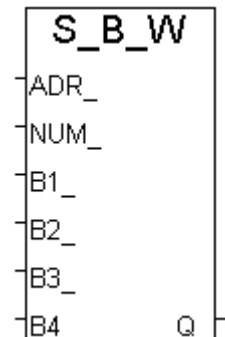
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

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

S_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write up to 4 boolean to the volatile SRAM

Arguments:

ADR_ Integer start from which address, 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)

NUM_ Integer how many booleans to write, 0 ~ 4

B1_~B4_ Boolean the boolean value to write

return:

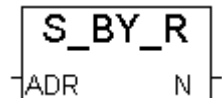
Q_ Boolean Ok: TRUE, Fail: FALSE

The boolean value will be stored is FALSE=0, TRUE=1

Please refer to section 10.3

S_BY_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one byte from the volatile SRAM

Arguments:

ADR_ Integer read which address, 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)

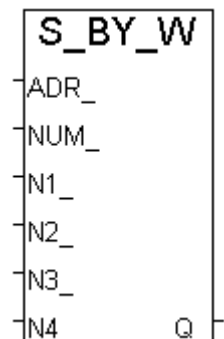
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

N_ Integer The byte value been read, 0 ~ 255

S_BY_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write up to 4 bytes to the volatile SRAM

Arguments:

ADR_ Integer start from which address, 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)

NUM_ Integer how many bytes to write, 0 ~ 4

N1_~N4_ Boolean the byte value (0-255) to write

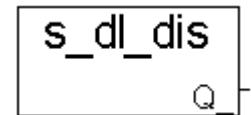
return:

Q_ Boolean Ok: TRUE, Fail: FALSE

Please refer to section 10.3

S_DL_DIS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

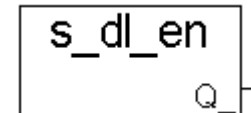
Function Disable the download permission, so that PC can not download data to the SRAM

return:

Q_ Boolean TRUE: ok, FALSE: fail

S_DL_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Enable the download permission for PC to download data to the volatile SRAM

return:

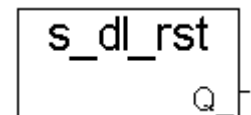
Q_ Boolean TRUE: ok, FALSE: fail

Note:

The default setting is "Disable". S_DL_EN should be called, then PC download data to the volatile SRAM is possible.

S_DL_RST

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

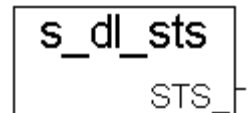
Function Reset the Download Status to "-1:No action" for the volatile SRAM

return:

Q_ Boolean TRUE: ok, FALSE: fail

S_DL_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Get PC's Download Status for the volatile SRAM

return:

STS_ Integer
-1: No action,
1: PC is Downloading data to the SRAM now
2: download accomplishment

Note:

S_DL_RST can be called to reset the status to -1 (reset to "No action")

Please refer to section 10.3

SET_LED

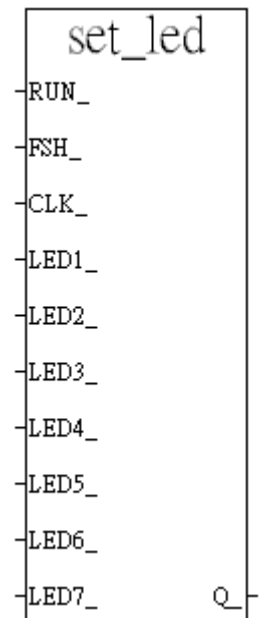
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function **Displays A Message To The S-MMI**

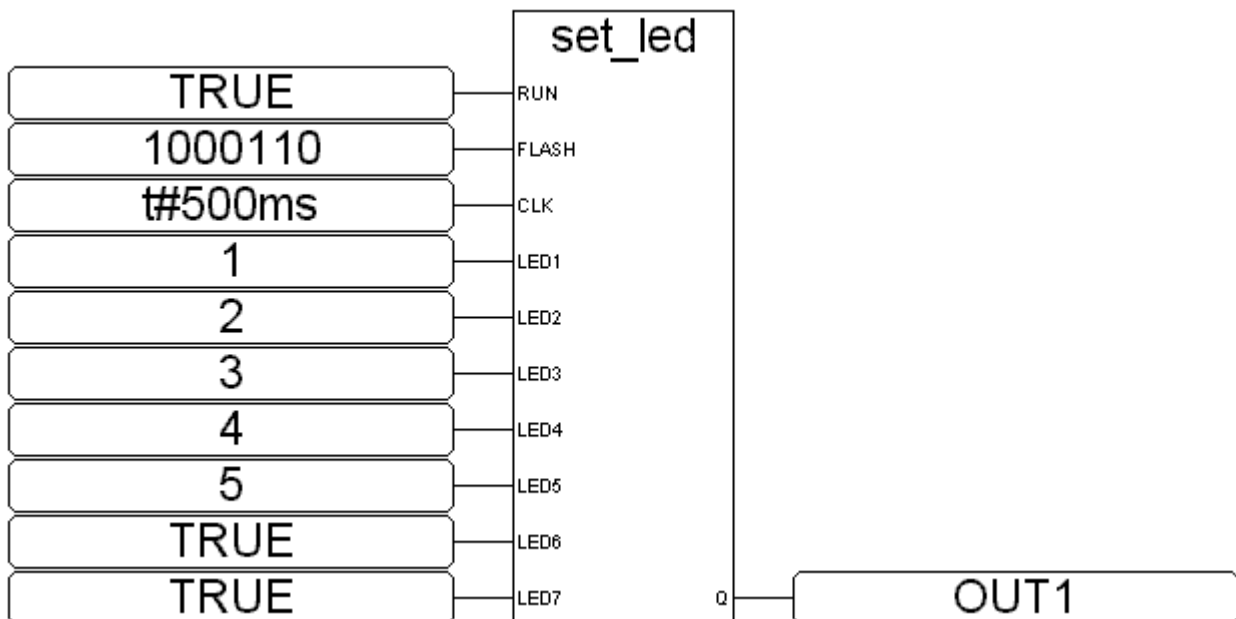
Arguments:

RUN_	Boolean	Set To "TRUE" To Display Message
FLASH_	Integer	Set each digit To "1" To blink each Message. Example: Set To 11 (0000011) Means The 6 th & 7 th Display Positions Will Blink. Set To 100001 (0100001) Means The 2 nd & 7 th Display Positions Will blink
CLK_	Timer	Amount Of Time For Display To blink
LED1_	Integer	Value Of Position Display #1
LED2_	Integer	Value Of Position Display #2
LED3_	Integer	Value Of Position Display #3
LED4_	Integer	Value Of Position Display #4
LED5_	Integer	Value Of Position Display #5
LED6_	Boolean	Value Of Position Display #6
LED7_	Boolean	Value Of Position Display #7



* Refer to section A.3 to see the display char. of LED1 ~ LED5, LED6, LED7.

Example:



ST equivalence:

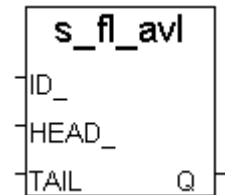
```
OUT1 := SET_LED(TRUE,1000110,t#500ms,1,2,3,4,5,TRUE,TRUE);
(* OUT1 is declared as a boolean variable *)
```

S_FL_AVL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function Set one file's current available byte No. for the volatile SRAM



Arguments:

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 reside inside the area of the associate file (Please refer to "S_FL_INI"), or Q_ will return FALSE

1 or	
S256:	1 ~ 249,856
S512:	1 ~ 512,000
X607:	1 ~ 118,784
X608:	1 ~ 512,000

For ex.,

A file of ID_ = 1 resides at byte No. of 1 ~ 20000 , it can store up to 20000 bytes.

1. if setting one of HEAD_ and TAIL_ to -1, no data of the file is available. It means 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, no data of the file is available. It means 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

S_FL_INI

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function Init one file's name & location for the volatile SRAM

Arguments:

ID_	Integer	File identifier No. (1 ~ 8)
NAME_	Message	File name, up to 8 char. for the name & up to 3 char. for the extension. For ex., "data1.txt", "A1234567.bin". Valid char. are A ~ Z , a ~ z , _ , 0 ~ 9, and the 1st should be A ~ Z or a ~ z
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

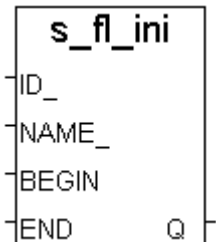
X608: 1 ~ 512,000

For 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-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function Reset file's Status to "Not been load by PC yet" for the volatile SRAM

Arguments:

ID_	Integer	File identifier No. (1 ~ 8)
------------	---------	-----------------------------

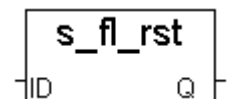
return:

Q_ Boolean TRUE: ok, FALSE: fail

Note:

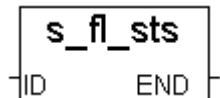
1. S_FL_INI should be called first.
2. S_FL_STS can be called to get file's status

Please refer to section 10.3



S_FL_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Get file's Status, end byte No. that has been load by PC for the volatile SRAM

Arguments:

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 of ID_ = 1 is located at byte No. of 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 return END_ is -1, it means PC hasn't load it yet.
2. If return END_ is 10000 (Normally the value is equal to the current available ending byte No.), it means PC has load it from 1001 ~ 10000
3. If return END_ is 8000, it means PC has load it from 1001 ~ 8000

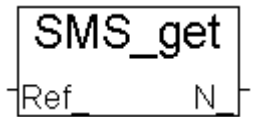
Note:

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")

Please refer to section 10.3

SMS_GET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Get message date and time from controller's date & time

Arguments:

REF_ **Integer** to get what ? , 1 ~ 7

- 1: get year, (N_ = 2000 ~ 2099)
- 2: get month, (N_ = 1 ~ 12)
- 3: get date, (N_ = 1 ~ 31)
- 4: get week date,(N_ = 1 ~ 7, 7 means Sunday)
- 5: get hour, (N_ = 0 ~ 23)
- 6: get minute, (N_ = 0 ~ 59)
- 7: get second, (N_ = 0 ~ 59)

others: return N_=-1 : error

return:

N_ **Integer** Return associated with Ref_. If return -1, it may be "No message" or Ref_ out of range of 1 ~ 7

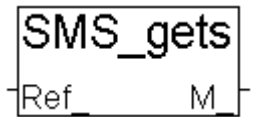
Note:

- 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 orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example: demo_43

SMS_GETS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Get message data and other information

Arguments:

REF_ Integer to get what ? , 1 ~ 3

- 1: get message data
- 2: get phone No. of sender
- 3: get date & time in string format

others: return M_ = 'error'

return:

M_ **Message** Return associated 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
2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No message". So if the orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

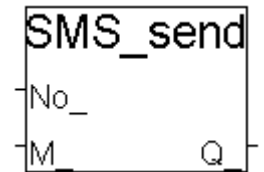
Example: demo_43

SMS_SEND

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Trigger the controller to send a new message



Arguments:

No_ message to which phone No. , fro ex. '+886920119135', max len is 31 digits

M_ message the message to send

return:

Q_ Boolean True: ok. ,
False: wrong phone No or "message sending status" is not 0 or 21

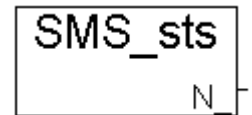
Note:

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

SMS_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Get Message Sending status**

return:

N_ **Integer**

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.)

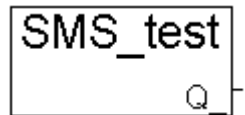
Note:

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

SMS_TEST

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Test if message coming or not

return:

Q_ Boolean TRUE: A message is coming, FALSE: No message

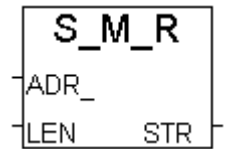
Note:

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 orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example: demo_43

S_M_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one string from the volatile SRAM

Arguments:

ADR_ Integer read which address.

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 value been read

For ex., data in memory is 16#31, 16#32, 16#33, 16#34, 16#35, 0, 16#37, 16#38, ...

LEN_=0 ----> STR_= " (empty string)

LEN_=3 ----> STR_= '123'

LEN_=5 ----> STR_= '12345'

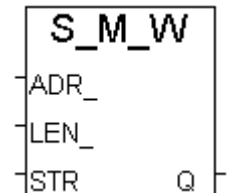
LEN_=6 ----> STR_= '12345'

LEN_=7 ----> STR_= '12345'

LEN_=100 ----> STR_= '12345'

S_M_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write one string to the volatile SRAM

Arguments:

ADR_ Integer write to which address.

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 write, 0 ~ 255.

STR_ Message the string value.

For ex.

LEN_=0 , STR_='12345' ----> no data written

LEN_=1 , STR_='12345' ----> 16#31 (1 byte written)

LEN_=3 , STR_='12345' ----> 16#31, 16#32, 16#33 (3 bytes written)

LEN_=7 , STR_='12345' ----> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0 (7 bytes written)

LEN_=100 , STR_='12345' --> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0, 0, ... (100 bytes written)

Return:

Q_ boolean Ok: TRUE, Fail: FALSE

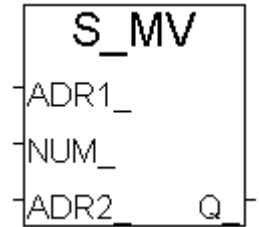
S_MV

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function Move some bytes inside the volatile SRAM

Arguments:



ADR1_ **Integer** destination start position

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 move, 0 - 512,000

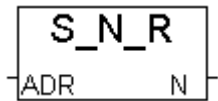
ADR2_ **Integer** Move from which starting position

return:

Q_ **boolean** Ok: TRUE, Fail: FALSE

S_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one integer from the volatile SRAM

Arguments:

ADR_ Integer read which address, 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)

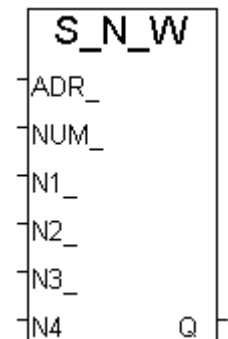
return:

N_ Integer The integer value been read, 32-bit, signed

The integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte], for ex. a integer of 16#01020304, it will be saved in the SRAM as [04] [03] [02] [01]

S_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write up to 4 integers to the volatile SRAM

Arguments:

ADR_ Integer start from which address, one Integer occupy 4 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 integers to write, 0 ~ 4

N1_~N4_ Integer the integer value (32-bit, signed) to write

The integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte], for ex. a integer of 16#01020304, it will be saved in the SRAM as [04] [03] [02] [01]

return:

Q_ Boolean Ok: TRUE, Fail: FALSE

S_R_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one real value from the volatile SRAM

Arguments:

ADR_ Integer read which address, 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)

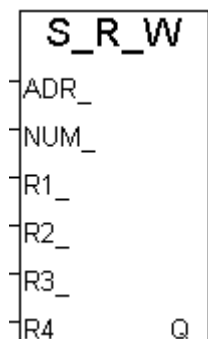
return:

R_ Real The real value been read, 32-bit float

The real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte]. For ex. Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F

S_R_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write up to 4 real values to the volatile SRAM

Arguments:

ADR_ Integer start from which address, one Real occupy 4 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 real values to write, 0 ~ 4

R1_~R4_ Real the real value (32-bit float) to write

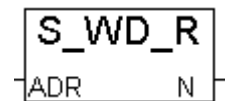
The real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte]. For ex. Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F

return:

Q_ Boolean Ok: TRUE, Fail: FALSE

S_WD_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Read one word from the volatile SRAM

Arguments:

ADR_ Integer read which address, one Word 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)

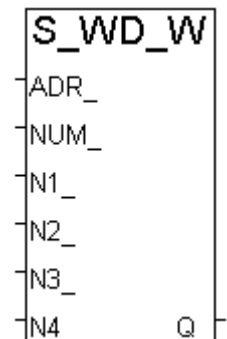
return:

N_ Integer The word value been read, -32768 ~ +32767

The word written in the SRAM is [Low byte] [High byte], for ex. a integer of 16#0102, it will be saved in the SRAM as [02] [01]

S_WD_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6



Description:

Function Write up to 4 words to the volatile SRAM

Arguments:

ADR_ Integer start from which address, one Word 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 write, 0 ~ 4

N1_~N4_ Integer the word value (-32768 ~ 32767) to write

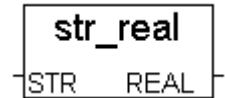
The word written in the SRAM is [Low byte] [High byte], for ex. a integer of 16#0102, it will be saved in the SRAM as [02] [01]

return:

Q_ Boolean Ok: TRUE, Fail: FALSE

STR_REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Convert a string to Real value

Arguments:

STR_	message	For ex, '123.456' , '-0.2345' , ' +2.13E10' , ' 15.2345E-2'
REAL_	real	The real value returned. If REAL_ is 1.23E-20 , it means STR_ is a wrong setting. For ex, if STR_=' 123.AB' or '23-45.17' or '1.2.345', REAL_ will return 1.23E-20

Note:

"REAL_STR" can be used to convert real value to a string

SYSDAT_R

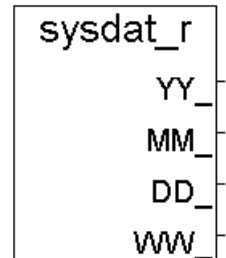
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block Read system year, month, day and date.

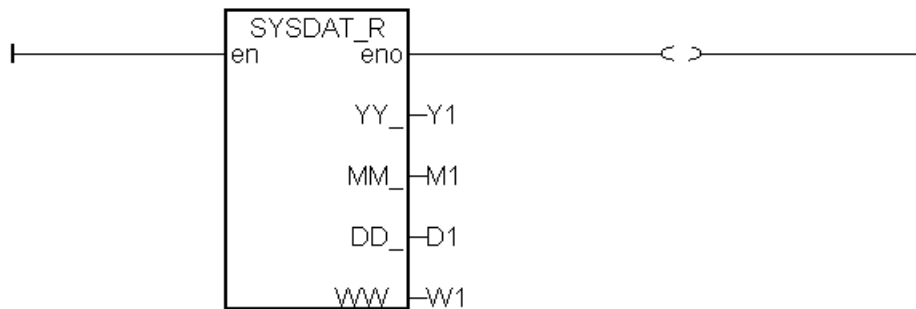
Arguments:

YY_	Integer	Year Returned (Example: 2002, 2003, 2010, Etc.)
MM_	Integer	Month Returned (1 = Jan., 3 = March, 10 = October, Etc.)
DD_	Integer	Day Returned, Valid Range From 1 To 31
WW_	Integer	Date Returned (1 = Monday, 4 = Thursday, 7 = Sunday, Etc.)



Example: refer to demo_03.

Y1, M1, D1 and W1 are declared as integer variables.



ST equivalence:

```
DAT_R1( ); (* call DAT_R1 *)
Y1 := DAT_R1.YY_ ; (* get year *)
M1 := DAT_R1.MM_ ; (* get month *)
D1 := DAT_R1.DD_ ; (* get day *)
W1 := DAT_R1.WW_ ; (* get date *)
(* DAT_R1 is declared as FB instance with typed - SYSDAT_R *)
```

SYSDAT_W

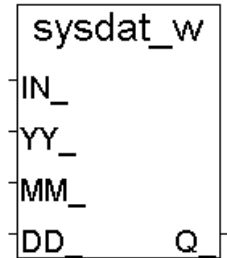
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block Set system year, month and day.

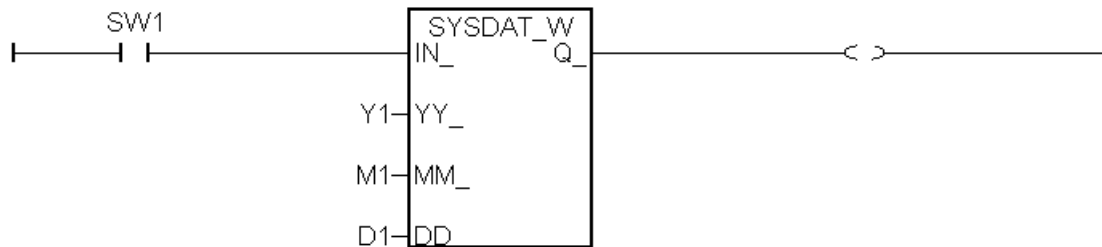
Arguments:

IN_	Boolean	Set System Date When Rising From "FALSE" To "TRUE"
YY_	Integer	Year To Write (Example: 2002, 2003, 2010, Etc.)
MM_	Integer	Month To Write (1 = Jan., 3 = March, 10 =October, Etc.)
DD_	Integer	Day Returned, Valid Range From 1 To 31
Q_	Boolean	If "OK", Returns "TRUE"



Example: refer to demo_03.

SW1 is declared as a boolean variable. Y1, M1, D1 are declared as integer variables.



St equivalence:

```

DAT_W1( SW1, Y1, M1, D1);      (* call DAT_W1 *)
OUT1 := DAT_W1.Q_ ;            (* get return value *)
(* DAT_W1 is declared as a FB instance with type - SYSDAT_W *)
(* OUT1 as a boolean variable *)

```

SYSTIM_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

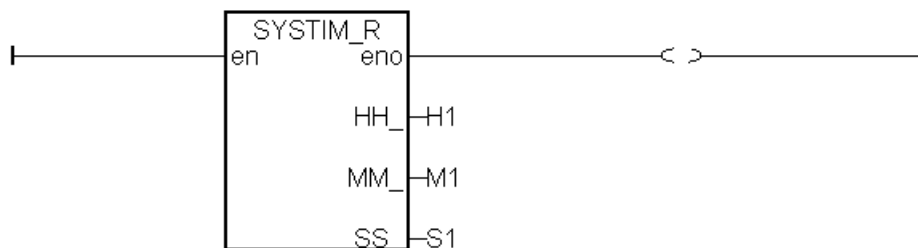
Function block Read system hour, minute and second.

Arguments:

HH_	Integer	Hour Returned (Valid Range From 0 To 23)
MM_	Integer	Minute Returned (Valid Range From 0 To 59)
SS_	Integer	Second Returned (Valid Range From 0 To 59)

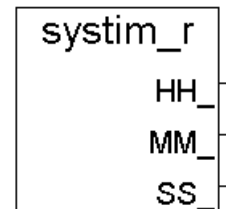
Example: refer to demo_03, demo_15b.

H1, M1 and S1 are declared as integer variables.



ST equivalence:

```
(* TIM_R1 is declared as FB instance with type - 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

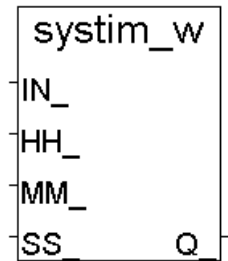
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function block Set system hour, minute and second.

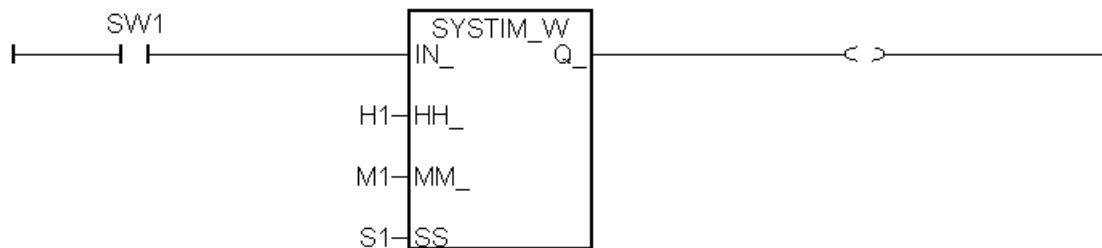
Arguments:

IN_	Boolean	Set System Date When Rising From "FALSE" To "TRUE"
HH_	Integer	Hour To Write, 0 - 23
MM_	Integer	Minute to Write, 0 - 59
SS_	Integer	Second to write, 0 - 59
Q_	Boolean	If "OK", Returns "TRUE"



Example: refer to demo_03.

SW1 is declared as a boolean variable. H1, M1, S1 are declared as integer variables.



St equivalence:

```
TIM_W1( Sw1,H1,M1,S1);          (* call TIM_W1 *)
OUT1 := TIM_W1.Q_ ;              (* get return value *)
(* TIM_W1 is declared as a FB instance with type - SYSTIM_W *)
(* OUT1 as a boolean variable *)
```

TIME_STR

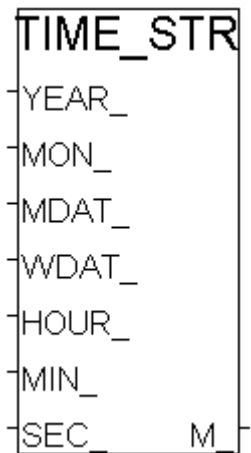
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Convert date & time to string format

Arguments:

YEAR_ integer year, 2000 ~
MON_ integer month, 1 ~ 12 (January ~ December)
MDAY_ integer mday, 1 ~ 31
WDAY_ integer wday, 1 ~ 7 (Monday ~ Sunday)
HOUR_ integer hour, 0 ~ 23
MIN_ integer minute, 0 ~ 59
SEC_ integer second, 0 ~ 59



If given wrong input parameters will return M_ = " (empty string). For. ex. give MON_=14

return:

M_ message length is 24 characters. For ex. 'Feb/18/2003,13:25:45,Tue'

Note: Please use sysdat_r & systim_r to get system date & time

TWIN_LED

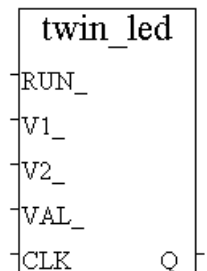
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

Function show a 2 screen values to the S-MMI

Arguments:

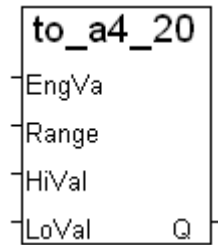
RUN_ boolean to show if TRUE
V1_ integer value displayed on the 2 digits on left of 1st screen, 0 ~ 99
V2_ integer value displayed on the 2 digits on right of 1st screen, 0 ~ 99
VAL_ integer value displayed on the 2nd screen, -99999 ~ 99999
CLK_ timer the blinking period of these 2 screens
Q_ boolean always TRUE



Example: refer to demo_10.

To_A4_20

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Convert User's Engineering Value ("Real" format) to Variable's AO value (Integer format) (Analog output signal is 4 to 20mA)

Arguments:

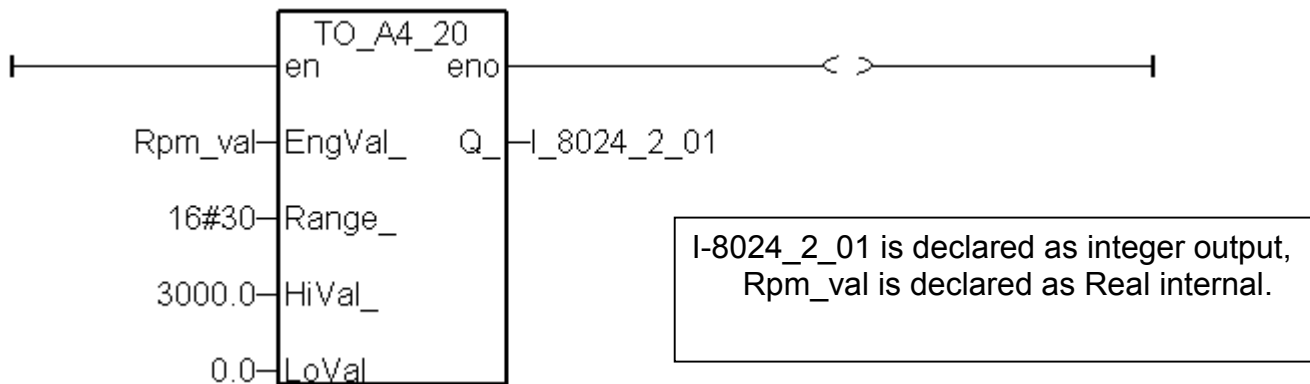
EngVal_	Real	the Engineering value to be converted.
Range_	Integer	Range setting of the Analog output board or module. 16#0 : 0 to 20 mA , 16#1 : 4 to 20 mA 16#30 : 0 to 20 mA , 16#31 : 4 to 20 mA
HiVal_	Real	User's related High Eng. value when analog output signal is 20 mA
LoVal_	Real	User's related Low Eng. value when analog output signal is 4 mA For example, Convert 0 - 100 psi to become I-8024's AO value, please set HiVal_ = 100.0 , LoVal_ = 0.0 and Range_ = 16#30 (dependeds 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
-----------	---------	---

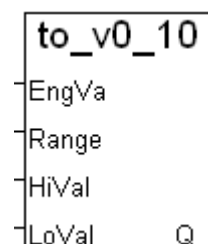
Example:

Scale (0 to 3000 rpm) to I-8024's current output with range setting of 30: (0 to 20 mA). 0 rpm should output 4 mA , 3000 rpm outputs as 20 mA.



To_V0_10

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Convert User's Engineering Value ("Real" format) to Variable's AO value (Integer format) (Analog output signal is 0 to 10V)

Arguments:

EngVal_	Real	the Engineering value to be converted.
Range_	Integer	Range setting of the Analog output board or module. 16#2 : 0 to 10 V , 16#32 : 0 to 10 V , 16#33 : -10 to 10 V 16#34 : 0 to 5 V , 16#35 : -5 to +5 V
HiVal_	Real	User's related High Eng. value when analog ouput signal is 10 V
LoVal_	Real	User's related Low Eng. value when analog output signal is 0 V For example, Convert 0 - 100 psi to become I-8024's AO value, please set HiVal_ = 100.0 , LoVal_ = 0.0 and Range_=16#33 (depeneds on the range setting of the related IO board)

return:

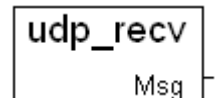
Q_	Integer	The AO value after conversion (value is usually in (0 to 32767) depends on which Range setting of the IO board). If given incorrect Range_ or (HiVal_ = LoVal_) , returns -1
-----------	---------	--

Example:

Please refer to example of "To_A4_20"

UDP_recv

□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Receive message from remote UDP/IP connection (via ethernet)**

Returns:

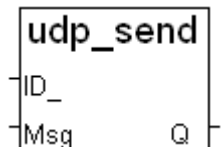
Msg_ Message the received message. If Msg_ = '' (empty message), it means no message coming

Note:

1. Please connect "udp_ip" in the IO connection windows before using UDP_recv & UDP_send
2. The receiving buffer size is Wincon:8096 bytes, I-7188EG & I-8x37:2048 bytes
3. I-7188EG & I-8437/8837 can only activate one of "udp_ip" or "ebus_m" or "ebus_s" or "ebus_s2". They can not be active at the same time in I-7188EG & I-8437/8837.
4. Please refer to Section 19.2 for more information.

UDP_send

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function **Send message to remote UDP/IP connection (via ethernet)**

Parameters:

ID_ Integer send to which connection, can be 1 to 4. The related "ip address" and "port No." is defined in "udp_ip"

Msg_ Message The message to send

return:

Q_ Boolean True: send OK , False: sending buffer is full or parameter error(For example, setting ID_ as 8).

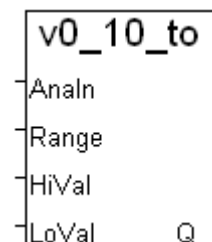
Note:

1. Please connect "udp_ip" in the IO connection windows before using "udp_send" & "udp_recv" functions
2. The sending buffer for Wincon is 2048 bytes. That means max. 2048 bytes in one PLC scan can be send to remote IP
3. I-7188EG & I-8437/8837 can only activate one of "udp_ip" or "ebus_m" or "ebus_s" or "ebus_s2". They can not be active at the same time in 7188EG & I-8437/8837.
4. 7188EG & I-8x37 doesn't support "udp_send"
5. The controller driver will send only one message out each PLC scan when there is message in the sending buffer. For example, if there is 100 messages in the sending buffer, the controller will send over these 100 message in 100 PLC scan cycle.
6. Please refer to Section 19.2 for more information.

Example: Wdemo_19 & Wdemo_20 (Wincon CD_ROM:\napdos\isagraf\wincon\demo\)

V0_10_to

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6



Description:

Function Convert Analog Input from 0 - 10 V to User's Engineering Value ("Real" format)

Arguments:

Analn_ Integer	the integer variable related to the Analog input board or module. The variable value is usually from -32768 to +32767 depends on the range setting of the IO board.
Range_ Integer	Range setting of the Analog input board or module. 16#0 : -15mV to +15 mV 16#1 : -50mV to +50 mV 16#2 : -100mV to +100 mV 16#3 : -500mV to +500 mV 16#4 : -1 to +1 V 16#5 : -2.5 to +2.5 V 16#7 : -1.25 to +1.25 V 16#8 : -10 to +10 V 16#9 : -5 to +5 V 16#A : -1 to +1 V 16#B : -500mV to +500 mV 16#C : -150mV to +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 For example, Convert I-8017H 's input signal from 0 - 10 V to become 0 - 100 psi, please set HiVal_ = 100.0 , LoVal_ = 0.0 and 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 Engineering value after conversion. if given incorrect Range_ , returns 1.23E-20

Example:

Please refer to example of "A4_10_to"

VAL_HEX

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function Convert an integer to a fixed-length hexa-message

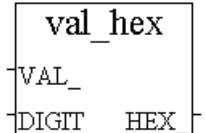
Arguments:

VAL_	integer	the value to be converted
DIGIT_	integer	number of digits of HEX_ , valid values are 1 ~ 8. Given others will do no conversion and force HEX_ to ' ' (empty message)
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' truncated)
val_hex(4589,9) --->  ' ' (DIGIT_ > 8, output ' ')
val_hex(-2,8)   --->  'FFFFFFFE'
  
```



VAL10LED

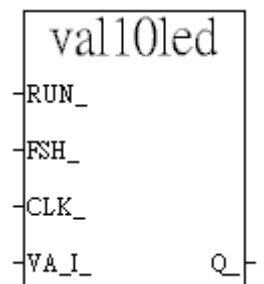
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

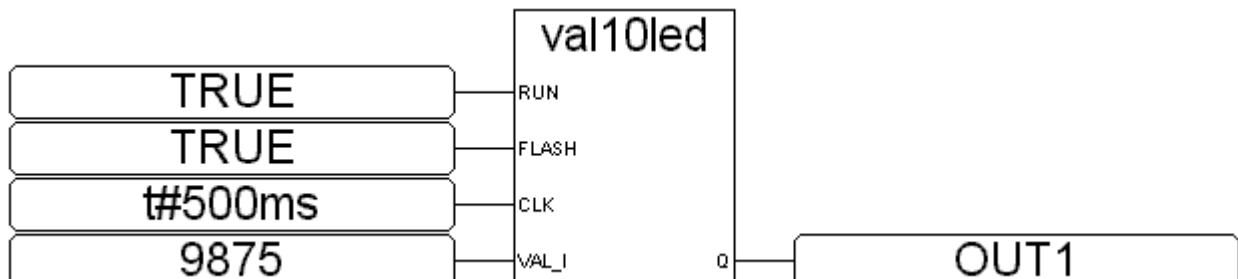
Function display an decimal integer on the S-MMI

Arguments:

RUN_	Boolean	if TRUE, display it
FLASH_	Boolean	if TRUE, blink it
CLK_	Timer	the blinking period
VAL_I_	Integer	the integer to be displayed Range from -9999 to +99999
Q_	Boolean	always returns TRUE °



Example: refer to demo_07, demo_11b.



ST equivalence:

```

out1 := VAL10LED(TRUE,TRUE,t#500ms,9875);
(* out1 is declared as a boolean variable *)
  
```

VAL16LED

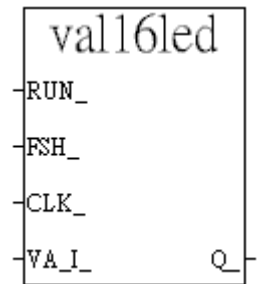
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8XX7/W-8XX6

Description:

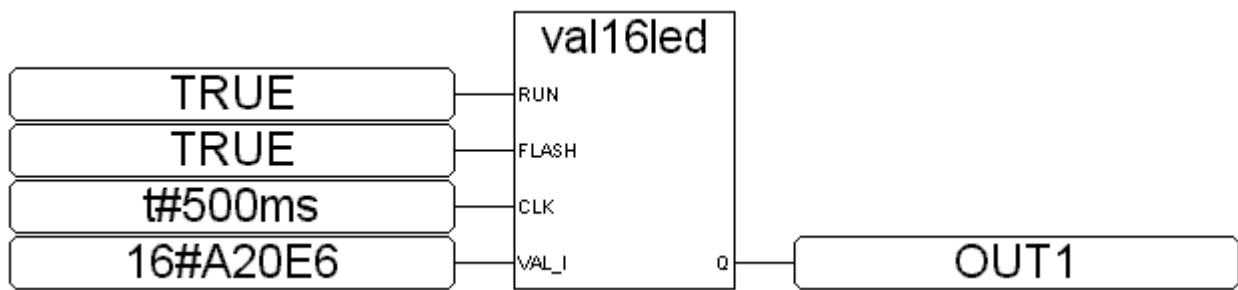
Function **display an hexadecimal integer on S-MMI**

Arguments:

RUN_	Boolean	if TRUE, display it
FLASH_	Boolean	if TRUE, blink it
CLK_	Timer	the blinking period
VAL_I_	integer	the value to be displayed Valid range from 16#0 to 16#FFFFFF
Q_	Boolean	always return TRUE



Example:



ST equivalence:

```
OUT1 := VAL10LED(TRUE,FALSE,t#500ms,16#A20E6);
(* OUT1 is declared as a boolean variable *)
```

V_BCD

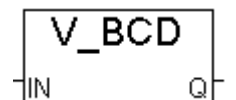
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function **Convert value to BCD value**

Arguments:

IN_	integer	the value to be converted
Q_	integer	the returned BCD value, For ex.
		12345 → 16#12345
		16 → 22 (16#16)



WD_BIT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

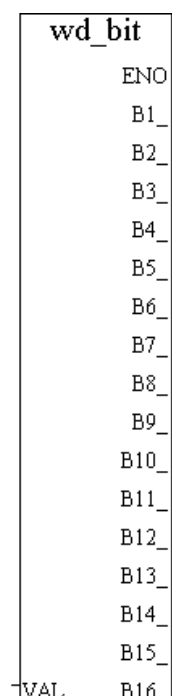
Description:

Function block Convert a word value to 16 boolean values

Arguments:

VAL_	integer	the word to be converted.
ENO	boolean	no usage, don't care about it.
B1_ ~ B16_	boolean	the 16 boolean values after converted

For ex. If VAL_ is 4, B3_ will be TRUE and others will be FALSE.
If VAL_ is 3, B1_ and B2_ will be TRUE and others will be FALSE.



WD_LONG

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

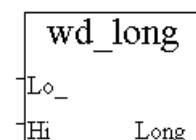
Function Convert two words to one long integer

Arguments:

Lo_	integer	Low word (only the lowest 16-bit is used)
Hi_	integer	High word (only the lowest 16-bit is used)
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)



W_MB_Adr

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function write boolean or integer variable by using its Network address No



Arguments:

TYPE_	Integer	0: boolean variable , 1: integer variable
ADR_	Integer	write to which Network address I-8xx7 & I-7188EG/XG: 1 to 4095 , W-8xx7: 1 to 8191
DATA_	Integer	the integer (or boolean 0:False , 1:True) to be written

return:

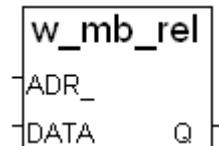
Q_	boolean	TRUE means Ok. , FALSE means fail
-----------	---------	-----------------------------------

W_MB_Rel

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8XX7/W-8XX6

Description:

Function write REAL variable by using its Network address No



Arguments:

ADR_	Integer	write to which Network address I-8xx7 & I-7188EG/XG: 1 to 4095 , W-8xx7: 1 to 8191
DATA_	Real	the Real value to be written

return:

Q_	boolean	TRUE means Ok. , FALSE means fail
-----------	---------	-----------------------------------

Note: Please refer to Section 11.3.5 for a demo description

1. Please use W_MB_REL function to write "REAL" variable. Please also refer to R_MB_Adr & R_MB_Rel
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 No. Please refer to section 4.2 of "User's Manual Of The ISaGRAF Embedded Controllers".

Example: Wincon: Wdemo_24 & I-8xx7:demo_70

Appendix B: Setting The IP, Mask & Gateway In The I-8437/8837 & I-7188EG

This document describe the proper way to set the IP address, address mask and gateway address of the I-8437/8837 & I-7188EG controllers.

EACH I-8437/ 8837 or I-7188EG USES TCP/IP PORT NO. 502 TO TALK TO THE HMI AND ISAGRAF WORKBENCH. A MAX. NUMBER OF 4 PCS CAN TALK TO THE I-8437/8837 or I-7188EG THROUGH MODBUS TCP/IP PROTOCOL.

1. Create a file folder named "8000" in your hard drive.
For example, "c:\8000".
2. Copy \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 your PC to COM1 of the I-8437/8837 (or I-7188EG) controller by a RS232 cable.
5. Power off the I-8437/8837 (or I-7188EG) controller, connect pin "INIT" to "INIT COM" (GND for I-7188EG), and then power it up.
6. If the connection is Ok, messages will appear on the 7188x screen.

**** 7188x Ver. 1.01.0 02/23/2000 ****

*** Press F1 for help. ***

ICP_DAS MiniOS7 for 8000-485 Ver. 1.03 build 014,May 09 2001 14:30:36

SRAM:512K, FLASH MEMORY:512K

Serial number= 5A 5A 5A 5A 5A 5A 5A 5A

8000>

7. Type "ip" to see the current IP address of the I-8437/8837 (or I-7188EG).

8000> ip

IP=192.168.255.255

8000>

8. Type "setip xxx.xxx.xxx.xxx" to set to a new IP address.

8000> setip 192.168.1.200

```
Set IP=192.168.1.200
[ReadBack]IP=192.168.1.200
8000>
```

9. Type "mask" to see the current address mask of the I-8437/8837 (or I-7188EG).

```
8000> mask
MASK=255.255.0.0
8000>
```

10. Type "setmask xxx.xxx.xxx.xxx" to set to a new address mask.

```
8000> setmask 255.255.255.0
Set MASK=255.255.255.0
[ReadBack]MASK=255.255.255.0
8000>
```

11. Type "gateway" to see the current gateway address.

```
8000> gateway
Gateway=192.168.0.1
8000>
```

12. Type "setgateway xxx.xxx.xxx.xxx" to set to a new gateway address.

```
8000> setgateway 192.168.1.1
Set GATEWAY=192.168.1.1
[ReadBack]Gateway=192.168.1.1
8000>
```

13. Press ALT_X to exit "7188x" and close the DOS SHELL, or COM1/COM2 of the PC will be occupied.

14. Remove the connection between "INIT" - "INIT COM", reset the I-8437 / 8837 (or I-7188EG) 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.

Please refer to the respective "Getting Started" Manual for Updating driver of the I-7188EG, I-7188XG & Wincon-8xx7.

Our newly released driver can also be obtained from the following website.

<http://www.icpdas.com/products/PAC/i-8000/isagraf.htm>

Warning:

The copyright of the firmware and the ISaGRAF embedded driver belongs to ICP DAS CO., LTD. Only the I-8417, 8817, 8437 and 8837 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.

Make sure of your current OS & driver version before you upgrade it.

1. Create a file folder named "8000" in your hard drive. For example, "c:\8000".

*** We use driver 3.09 as an example in this document.

2. Copy \Napedos\ISaGRAF\8000\Driver\2.50\7188xw.exe", "7188xw.ini" , "isa.exe" , "autoexec.bat" & "8k031105.IMG" from the CD_ROM into your "8000" folder.

3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen will appear (Press F1 for help).

4. Link COM1 or COM2 of your PC to COM1 of the I-8xx7 controller through a RS232 cable.

5. Power off the I-8xx7 controller, connect pin "INIT" to "INIT COM" and then power it up.

6. If the connection is Ok, messages will appear on the 7188x screen.
8000>

7. Type "ver" to see the current OS version.
8000> ver

8. Type "isa *p=" to see the version No. & COMM setting of the ISaGRAF driver
8000> isa *p=

To upgrade an ISaGRAF embedded driver, follow the following steps.

9. Power off the I-8xx7 controller, connect pin "INIT" to "INIT COM" and then power it up.

10. The OS image should upgrade first. Type "upload" to load the OS image

8000> **upload**

press at **ALT+E** and type in **the image name** (for version 2.50 - 8k031105.IMG)

and then type "bios1"

8000> **bios1**

(WAIT ABOUT 30 SEC. ***DO NOT REMOVE THE POWER IN THESE 30 SEC.***)

11. To upgrade the ISaGRAF driver. Type "del" and reply "y" to delete the current driver.

8000> del

Total File number is 2, do you really want to delete(y/n)? **y**

12. Type "load", then press ALT_E and then type "isa_data.exe" .

8000> **load**

File will save to 8000:0000

StartAddr-->7000:FFFF

Press **ALT_E** to download file!

Input filename:**isa_data.exe**

13. Please type isa_data to run "isa_data.exe", and then type del to delete "isa_data.exe"

8000> **isa_data**

8000> **del**

Total File number is 1, do you really want to delete(y/n)? **y**

14. Type "load", then press ALT_E and then type "autoexec.bat" .

8000> **load**

File will save to 8000:0000

Press **ALT_E** to download file!

Input filename:**autoexec.bat**

15. Type "load" again, then press ALT_E and then type "isa.exe". Wait until it finished.

8000> **load**

File will save to 8003:0002

StartAddr-->8000:0031

Press **ALT_E** to download file!

Input filename:**isa.exe**

14. Type "dir" to make sure "autoexec.bat" and "isa.exe" are well burned.

8000> dir

15. Press ALT_X to exit "7188x".

16. Remove the connection between "INIT" - "INIT COM", reset the I-8xx7 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 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.

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-8417/8817/8437/8837 (or I-7188EG) by a RS232 cable.
5. Power off the I-8417/8817/8437/8837 (or I-7188EG), connect pin "INIT" to "INIT COM", then power it up.
6. If the connection is Ok, messages will appear on the 7188x screen.

8000>

7. Type "**isa *f=1**" to free COM1 (set COM1 as none-Modbus-Slave port)
(For I-7188EG, type "**isa7188e *f=1**")

8000> isa *f=1

8. Press ALT_X to exit "7188x", or COM1/COM2 of the PC will be occupied.
9. Remove the connection between "INIT" - "INIT COM", recycle the power of the controller.

Important Note:

If user wants COM1 to be back to a Modbus RTU Slave port again, follow the same step 1 to 6 & then type "isa *f=0" as below. (For I-7188EG, type "**isa7188e *f=0**")

8000> isa *f=0

Appendix D: Table of The Analog IO Value

I-87013, I-7013, I-7033

Range Code (Hex)	RTD Type	Data Format	Max Value	Min Value
20 (Default)	Platinum 100 a = 0.00385	Input Range (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
21	Platinum 100 a = 0.00385	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
22	Platinum 100 a = 0.00385	Input Range (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
23	Platinum 100 a = 0.00385	Input Range (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
24	Platinum 100 a = 0.003916	Input Range (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
25	Platinum 100 a = 0.003916	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
26	Platinum 100 a = 0.003916	Input Range (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
27	Platinum 100 a = 0.003916	Input Range (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
28	Nickel 120	Input Range (Celsius)	+100.0	-80.0
		Decimal Value	+32767	-262140
		2's complement HEX	7FFF	999A
29	Nickel 120	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
2A	Platinum 1000 a = 0.00385	Input Range (Celsius)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556

I-8017H

* Each channel can be configured to different range ID

Range 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-87017, I-7017

Range 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

I-87018, I-7011, I-7018

Range 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

Range Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
0E	J Type	Input Range (Celsius)	+760.0	-210.0
		Decimal Value	+32767	-9054
		2's Complement HEX	7FFF	DCA2
0F	K Type	Input Range (Celsius)	+1372.0	-270.0
		Decimal Value	+32767	-6448
		2's Complement HEX	7FFF	E6D0
10	T Type	Input Range (Celsius)	+400.0	-270.0
		Decimal Value	+32767	-22118
		2's Complement HEX	7FFF	A99A

11	E Type	Input Range (Celsius)	+1000.0	-270.0
		Decimal Value	+32767	-8847
		2's Complement HEX	7FFF	DD71
12	R Type	Input Range (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
13	S Type	Input Range (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
14	B Type	Input Range (Celsius)	+1820.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
15	N Type	Input Range (Celsius)	+1300.0	-270.0
		Decimal Value	+32767	-6805
		2's Complement HEX	7FFF	E56B
16	C Type	Input Range (Celsius)	+2320.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
17	L Type	Input Range (Celsius)	+800.0	-200.0
		Decimal Value	+32767	-8192
		2's Complement HEX	7FFF	E000
18	M Type	Input Range (Celsius)	+100.0	-200.0
		Decimal Value	+16384	-32768
		2's Complement HEX	4000	8000
19	L Type DIN43710	Input Range (Celsius)	+900.0	-200.0
		Decimal Value	+32767	-7281
		2's Complement HEX	7FFF	E38F

I-7021

Range 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 (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

I-8024

* Each channel can be configured to different range ID

Range 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

I-87024, I-7024

Range 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

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.**

© 1994 - 2002 **AlterSys Inc.** All rights reserved.

Published in Canada by **AlterSys Inc.**

ISaGRAF is a registered trademark of **AlterSys Inc.**

MS-DOS is a registered trademark of Microsoft Corporation.

Windows is a registered trademark of Microsoft Corporation.

Windows NT is a registered trademark of Microsoft Corporation.

OS-9 and ULTRA-C are registered trademarks of Microware Corporation.

VxWorks and Tornado are registered trademarks of Wind River Systems, Inc.

All other brand or product names are trademarks or registered trademarks of their respective holders.

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.

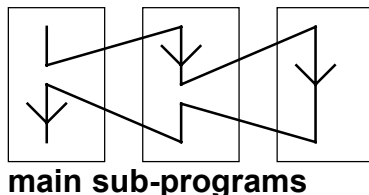
When a parent program starts a child **SFC** program, it puts an SFC **token** (activates) into each initial step of the child program. This command is described with the **GSTART** statement. When a parent program kills a child **SFC** program, it clears all the tokens existing in the **steps** of the 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:



Any program of any section may have one or more sub-programs. A sub-program is owned by only one father program. A sub-program may have local variables and defines. Any language but **SFC** or **FC** can be used to describe a sub-program. Programs of the **"Functions"** section are sub-programs that can be called by any other program in the project. Unlike other sub-programs, they are not dedicated to one father program. A

program of the **"Function"** section can call another program of this section. A function can be located in the Library.

Warning: The ISaGRAF system does not support **recursive function calls**. A run time error will occur if a program of the **"Functions"** section is called by itself or by one of its called sub-program.

Warning: A function or sub-program does not "store" the local value of its local variables. A function or sub-program is not instantiated and so can not call function blocks.

The interface of a sub-program must be explicitly defined, with a **type** and a **unique name** for each of its calling or return parameter. In order to support the **ST** language convention, the return parameter must have the same name as the sub-program.

The following table shows how to set the value of the return parameter in the body of a sub-program, in the different languages:

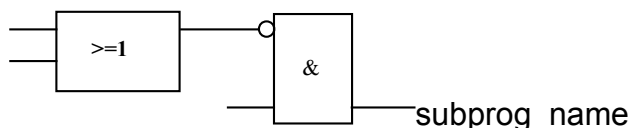
ST: assign the return parameter using its name
(the same name as the sub-program):

subprog_name := <expression>;

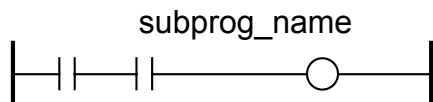
IL: the value of the current result (IL register)
at the end of the sequence is stored in the return parameter:

LD 10
ADD 20 (* return parameter value = 30 *)

FBD: set the return parameter using its name:

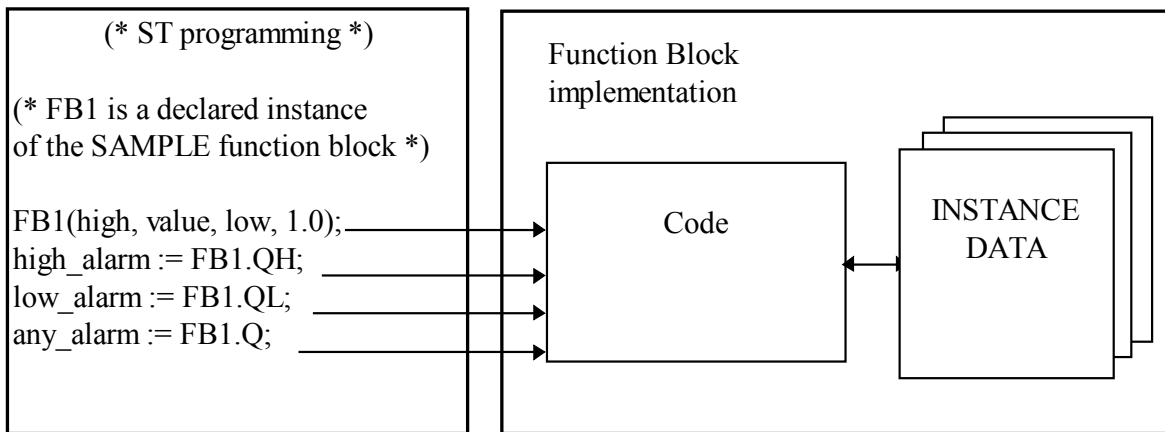


LD: use a coil symbol with the name of the return parameter:



E.1.5 Function blocks

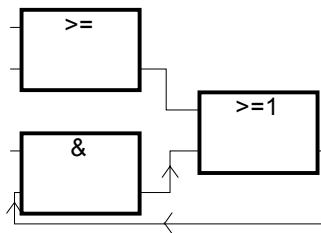
Function blocks can use the languages: LD, FBD, ST or IL. Function blocks are instantiated. It means local variables of a function block are copied for each instance. When calling a block in a program, you actually call the instance of the block: the same code is called but the data used are the one which have been allocated for the instance. Values of the variables of the instance are stored from one cycle to the other.



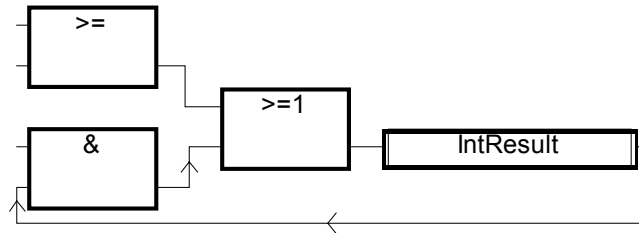
Warnings:

- A function block written with one of the IEC languages can not call other function blocks: the instantiation mechanism only manages the local variables of the block itself. Here is the list of standard function blocks that you cannot use inside an IEC function block:
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALARM, INTEGRAL, DERIVATE, BLINK, SIG_GEN
- For the same reason, you can not use Positive or Negative contact or coils, or Set and Reset coils.
- TSTART and TSTOP functions to start and stop timers cannot be used in a function block for 3.0x targets. It works since the 3.20 target.
- When you need loop in your function block, you must use local variable before doing the loop. See the example below:

This will not work:



This is OK:



E.1.6 Description language

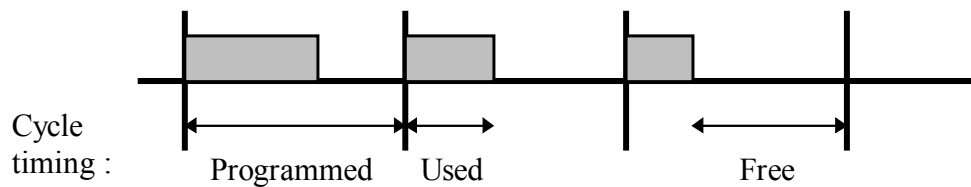
A program can be described with any of the following graphic or literal languages:

- Sequential Function Chart (SFC)** for high level operations
- Flow Chart (FC)** for high level operations
- Function Block Diagram (FBD)** for cyclic complex operations
- Ladder Diagram (LD)** for boolean operations only
- Structured Text (ST)** for any cyclic operations
- Instruction List (IL)** for low level operations

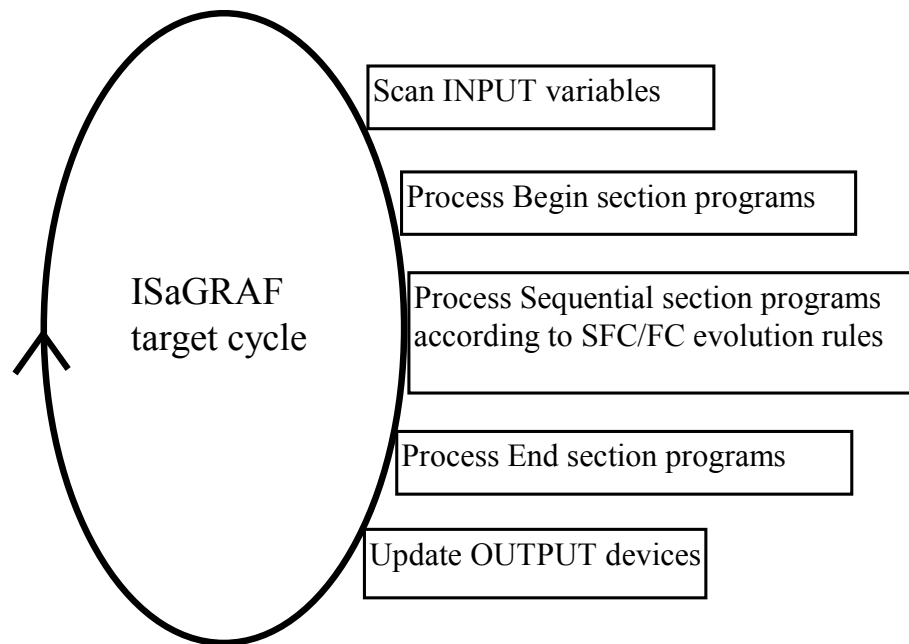
The same program cannot mix several languages. The language used to describe a program is chosen when the program is created, and cannot be changed later on. The exception is that it is possible to combine FBD and LD in a single program.

E.1.7 Execution rules

ISaGRAF is a **synchronous** system. All the operations are triggered by a clock. The basic duration of the clock is called the cycle timing:



Basic operations processed during a target cycle are:



This system makes it possible to:

- guarantee that an input variable keeps the same value within a cycle,
- guarantee that an output device is not updated more than once in a cycle,
- work safely on the same global variable from different programs,
- estimate and control the response time of the complete application.

E.2 Common objects

These are main features and common **objects** of the ISaGRAF programming database. Such objects can be used in any program written with any of the **SFC, FC, FBD, LD, ST** or **IL** languages.

E.2.1 Basic types

Any constant, expression or variable used in a program (written in any language) must be characterised by a type. Type coherence must be followed in graphic operations and literal statements. These are the available basic types for programming objects:

BOOLEAN: logic (true or false) value

ANALOG: integer or real (floating) continuous value

TIMER: time value

MESSAGE: character string

Note: Timers contain values less than one day and cannot be used to store dates.

E.2.2 Constant expressions

Constant expressions are relative to one type. The same notation cannot be used to represent constant expressions of different types.

E.2.2.1 Boolean constant expressions

There are only two boolean constant expressions:

TRUE is equivalent to the integer value 1

FALSE is equivalent to the integer value 0

"True" and "False" keywords are case insensitive.

E.2.2.2 Integer analog constant expressions

Integer constant expressions represent signed long integer (32 bit) values: from **-2147483647** to **+2147483647**. Integer analog constants may be expressed with one of the following **bases**. Integer constants must begin with a **prefix** that identifies the bases used:

Base	Prefix	Example
DECIMAL	(none)	-908
HEXADECIMAL	"16#"	16#1A2B3C4D
OCTAL	"8#"	8#1756402
BINARY	"2#"	2#1101_0001_0101_1101

The underscore character ('_') may be used to separate groups of digits. It has no particular significance, and is used to increase constant expression readability.

E.2.2.3 Real analog constant expressions

Real analog constant expressions can be written with either **decimal** or **scientific** representation. The **decimal point** ('.') separates the integer and decimal parts. The decimal point must be used to differentiate a real constant expression from an integer one. The scientific representation uses the 'E' or 'F' letter to separate the **mantissa** part and the **exponent**. Exponent part of a real scientific expression must be a signed integer value from -37 to +37. Below are examples of real analog constant expressions:

3.14159 -1.0E+12
+1.0 1.0F-15
-789.56 +1.0E-37

The expression "**123**" does not represent a real constant expression. Its correct real representation is "**123.0**".

E.2.2.4 Timer constant expressions

Timer constant expressions represent time values from **0 second** to **23h59m59s999ms**. The lowest allowed unit is a millisecond. Standard time units used in constant expressions are:

Hour The "h" letter must follow the number of hours
Minute The "m" letter must follow the number of minutes
Second The "s" letter must follow the number of seconds
Millisecond The "ms" letters must follow the number of milliseconds

The time constant expression must begin with "**T#**" or "**TIME#**" prefix. Prefixes and unit letters are case insensitive. Some units may not appear. These are examples of timer constant expressions:

T#1H450MS 1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes

The expression "0" does not represent a time value, but an analog constant.

E.2.2.5 Message string constant expressions

String or message constant expressions represent character strings. Characters must be preceded by a quote and followed by an apostrophe. For example:

'THIS IS A MESSAGE'

Warning: The apostrophe "'" character cannot be used within a string constant expression. A string constant expression must be expressed on one line of the program source code. Its length cannot exceed 255 characters, including spaces.

Empty string constant expression is represented by two apostrophes, with no space or tab character between them:

" (* this is an empty string *)

The special character dollar ('\$'), followed by other special characters, can be used in a string constant expression to represent a non-printable character:

Sequence	Meaning	ASCII (hexa)	Example
\$\$	'\$' character	16#24	'I paid \$\$\$5 for this'
\$'	apostrophe	16#27	'Enter \$'Y\$' for YES'
\$L	line feed	16#0a	'next \$L line'
\$R	carriage return	16#0d	' llo \$R He'
\$N	new line	16#0d 0a	'This is a line\$N'
\$P	new page	16#0c	'lastline \$P first line'
\$T	tabulation	16#09	'name\$Tsize\$Td ate'
\$hh (*)	any character	16#hh	'ABCD = \$41\$42\$43\$4 4'

(*) "**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, GSKILL, 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:

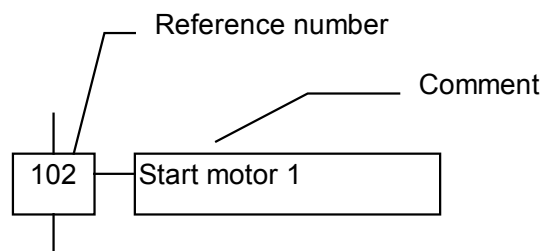
- A step cannot be followed by another step
- A transition cannot be followed by another transition

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.

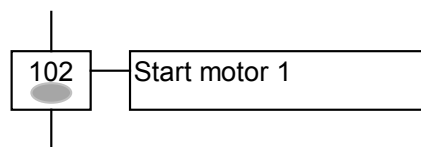
E.3.2.1 Steps and initial steps

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:

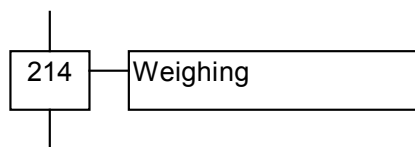


At run time, a **token** indicates that the step is **active**:

Active step:

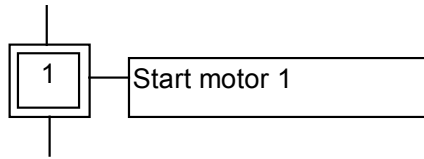


Inactive step:



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:

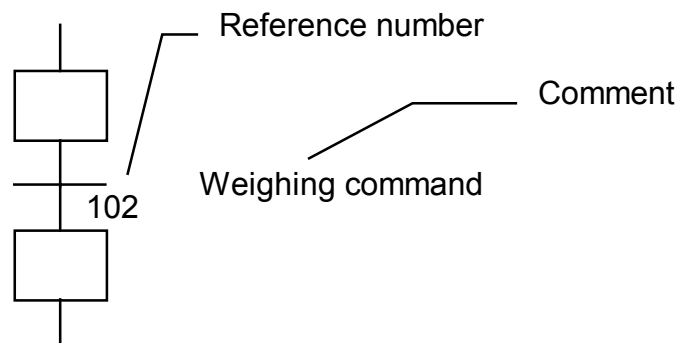
GSnnn.x activity of the step (boolean value)

GSnnn.t activation duration of the step (time value)

(where **nnn** is the reference number of the step)

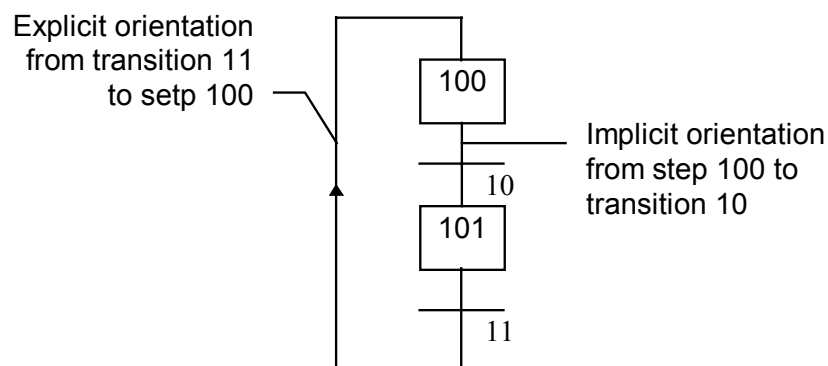
E.3.2.2 Transitions

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:



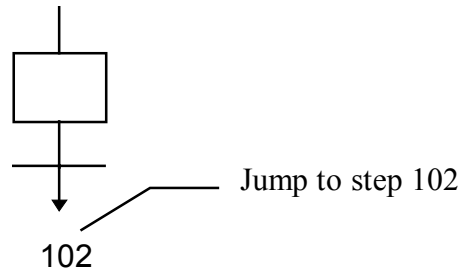
E.3.2.3 Oriented links

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.

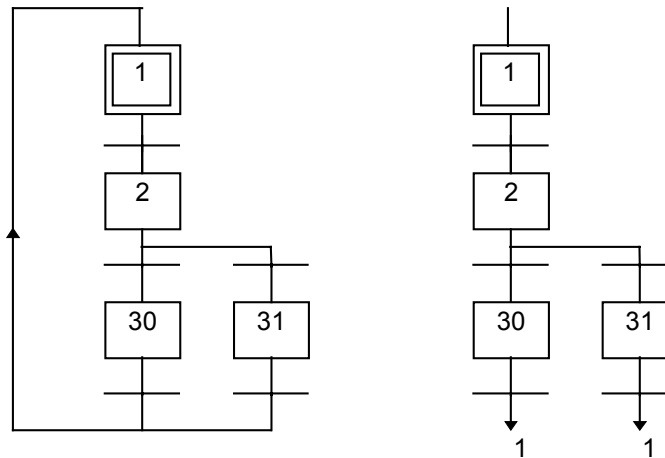


E.3.2.4 Jump to a step

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:



A jump symbol cannot be used to represent a link from a step to a transition. Example of jumps - the following charts are equivalent:

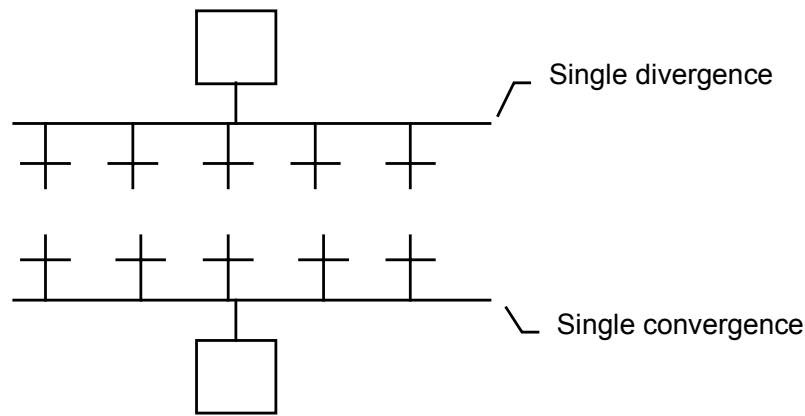


E.3.3 Divergences and convergences

Divergences are **multiple connection links** from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

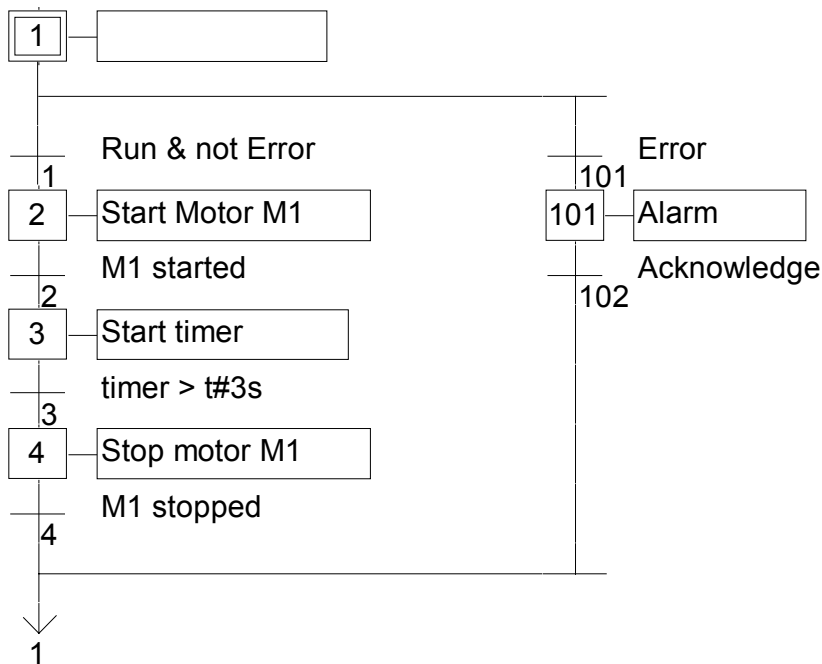
E.3.3.1 single divergences

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence. Single divergences and convergences are represented by single horizontal lines.



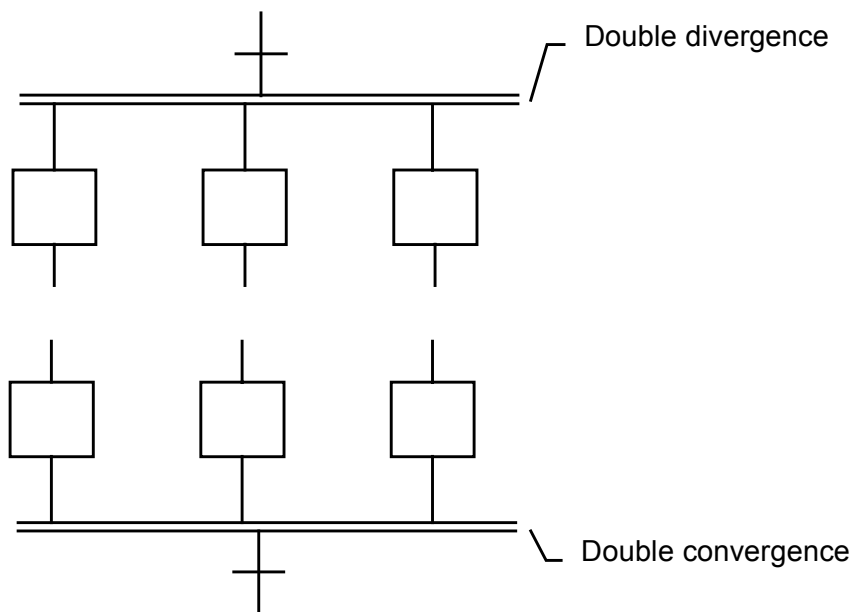
Warning: The conditions attached to the different transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of the transitions to ensure that only one token progresses in one branch of the divergence at run time. Below is an example of single divergence and convergence:

(* SFC program with single divergence and convergence *)



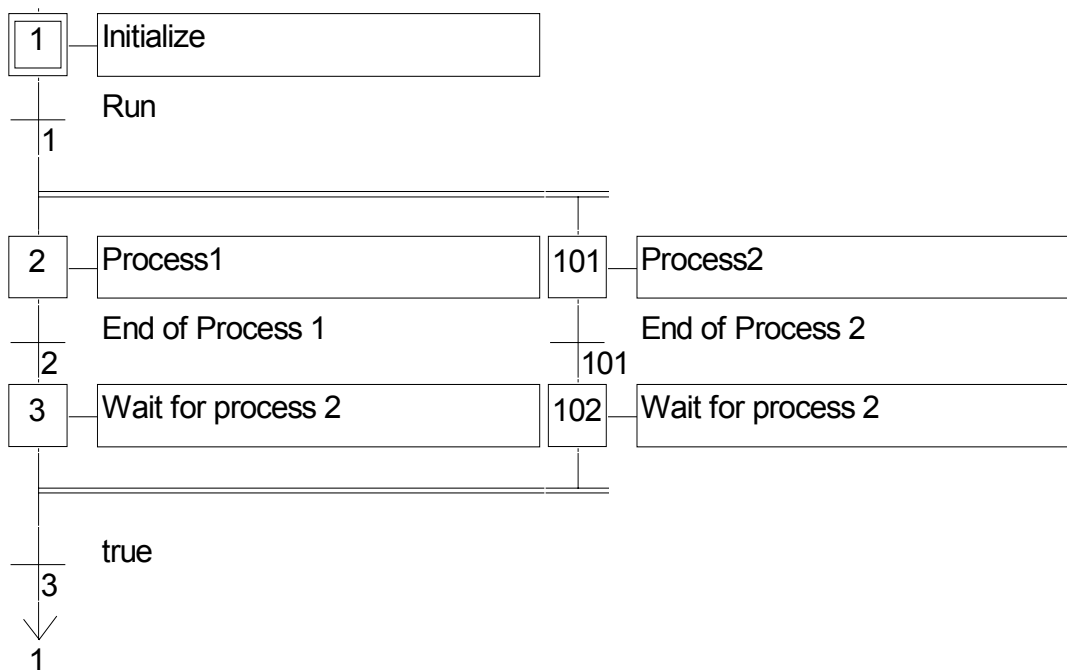
E.3.3.2 Double divergences

A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence. Double divergences and convergences are represented by double horizontal lines.



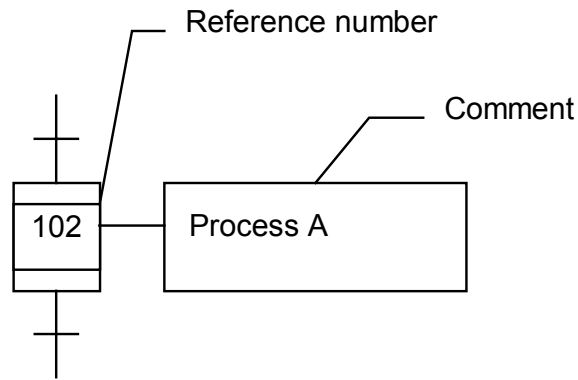
Example of double divergence and convergence:

(* SFC program with double divergence and convergence *)



E.3.4 Macro steps

A macro step is a unique representation of a unique group of steps and transitions. The body of the macro step is described separately, elsewhere in the same SFC program. It appears as a single symbol in the main SFC chart. This is the symbol used for a macro step:



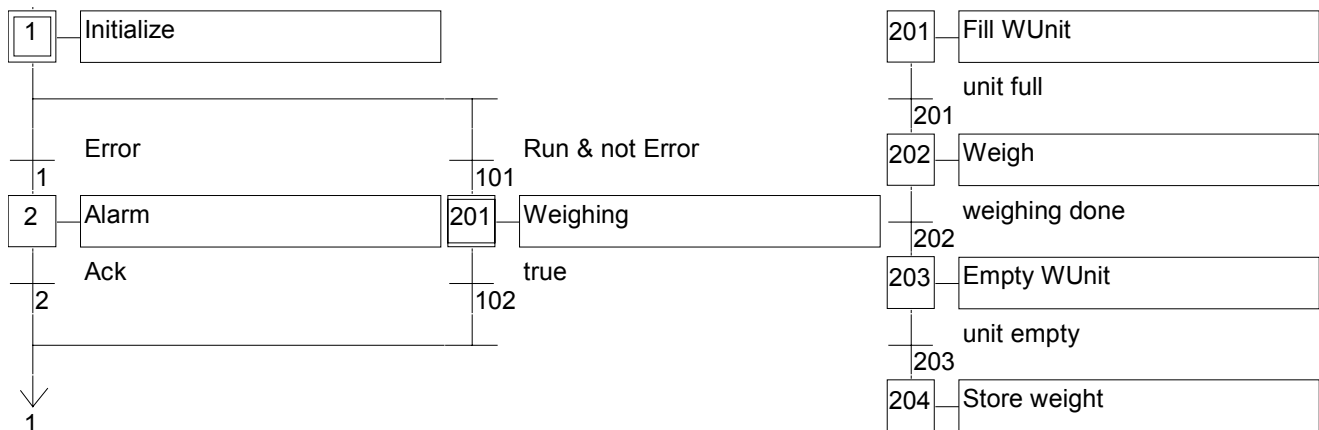
The reference number written in the macro step symbol is the reference number of the first step in the body of the macro step. The macro step body must begin with a **beginning step** and terminate with an **ending step**. The chart must be self-contained. A beginning step has no upper link (no backward transition). An ending step has no lower link (no forward transition). A macro step symbol may be put in the body of another macro step.

Warning: Because macro step is a **unique** set of steps and transitions, the same macro step cannot be used more than once in an SFC program.

Example of macro step:

(* SFC program with macro step *)

(* Main chart *) (* Body of the macro step *)



E.3.5 Actions within the steps

The **level 2** of an SFC step is the detailed description of the **actions** executed **during the step activity**. This description is made by using **SFC literal features**, and other languages such as Structured Text (**ST**). The basic types of actions are:

- Boolean actions
- Pulse actions programmed in ST
- Non-stored actions programmed in ST
- SFC actions

Several actions (with same or different types) can be described in the same step. The special features that enable the use of any of the other languages are:

- Calling sub-programs
- Instruction List (IL) language convention

E.3.5.1 Boolean actions

Boolean actions assign a boolean variable with the activity of the step. The boolean variable can be an output or an internal. It is assigned each time the step activity starts or stops. This is the syntax of the basic boolean actions:

<boolean_variable> (N) ; assigns the step activity signal to the variable

<boolean_variable> ; same effect (N attribute is optional)

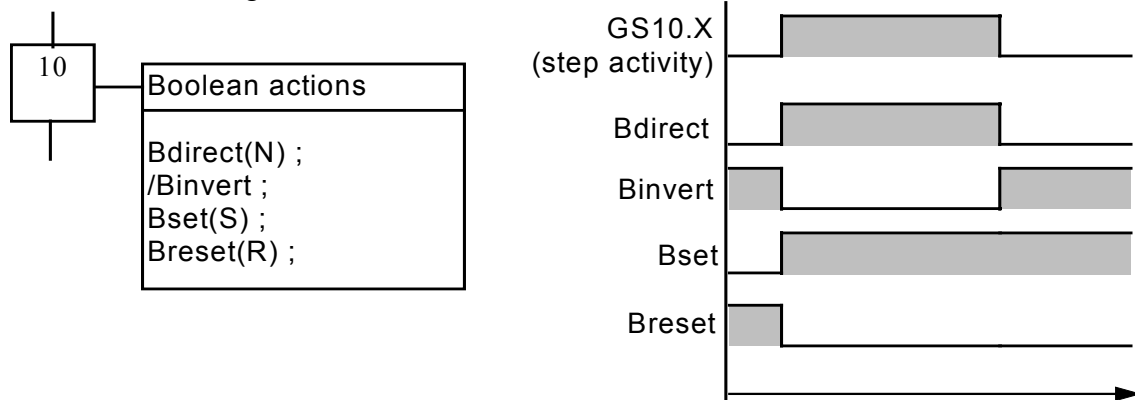
/ <boolean_variable> ; assigns the negation of the step activity signal to the variable

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

<boolean_variable> (S) ; sets the variable to TRUE when the step activity signal becomes TRUE

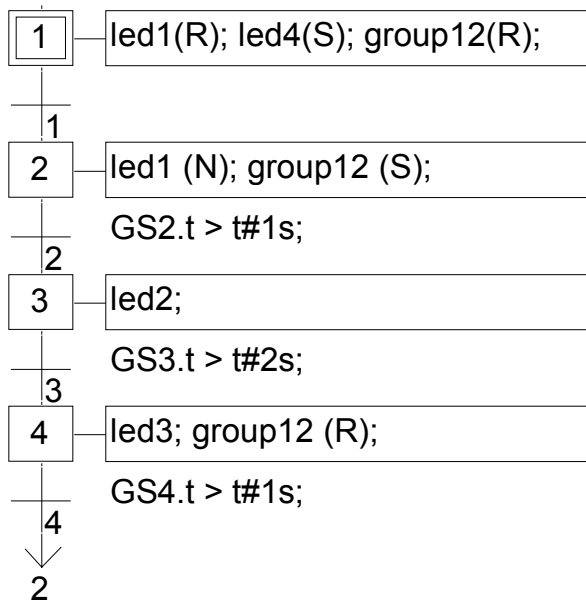
<boolean_variable> (R) ; resets the variable to FALSE when the step activity signal becomes TRUE

The boolean variable must be an OUTPUT or an INTERNAL. The following SFC programming leads to the following behaviour:



Example of boolean actions:

(* SFC program using BOOLEAN actions *)

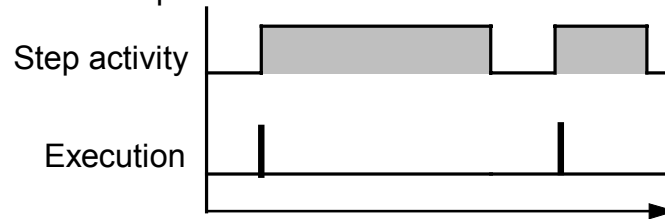


E.3.5.2 Pulse actions

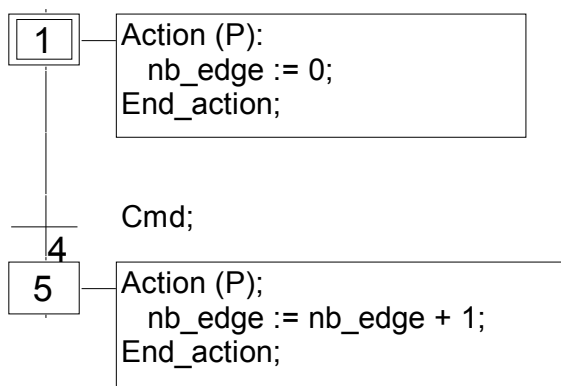
A pulse action is a list of ST or IL instructions, which are executed only **once** at the **activation** of the step. Instructions are written according to the following SFC syntax:

ACTION (P) :
 (* ST statements *)
END_ACTION ;

The following shows the results of a pulse action:



Example of pulse action:

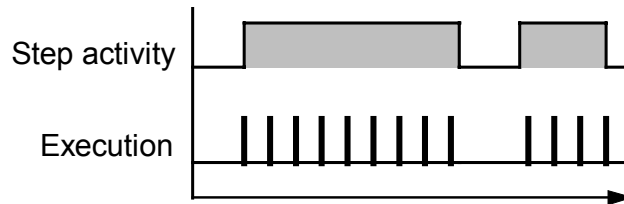


E.3.5.3 Non-stored actions

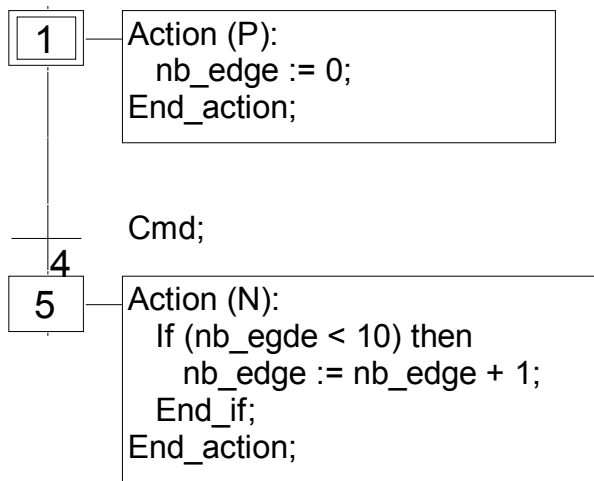
A non-stored (normal) action is a list of ST or IL instructions which are executed **at each cycle** during the whole **active** period of the step. Instructions are written according to the following SFC syntax:

```
ACTION (N) :  
(* ST statements *)  
END_ACTION ;
```

The following is the results of a non-stored action:



Example of non-stored action:



E.3.5.4 SFC actions

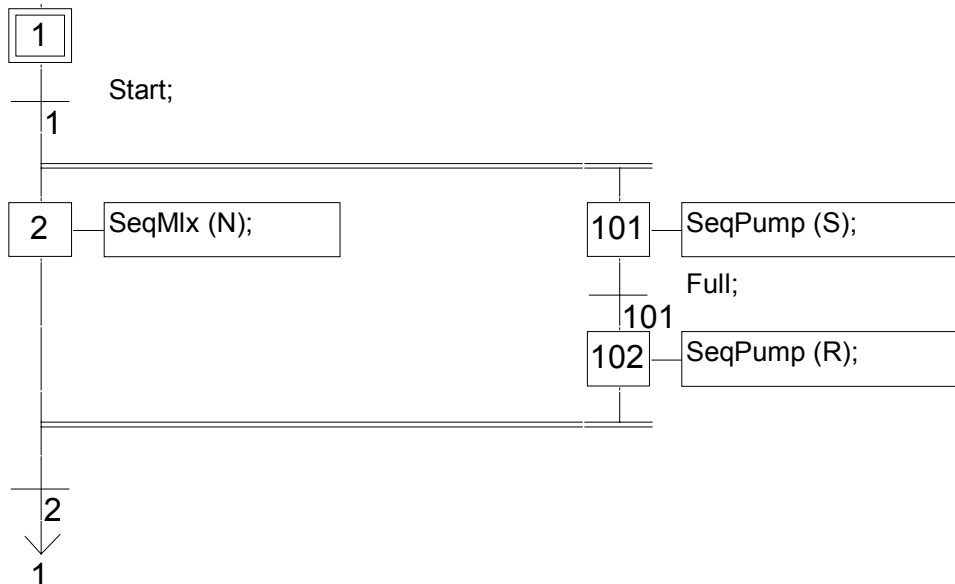
An SFC action is a child SFC sequence, started or killed according to the change of the step activity signal. An SFC action can have the **N** (Non stored), **S** (Set), or **R** (Reset) qualifier. This is the syntax of the basic SFC actions:

- <child_prog> (N);** starts the child sequence when the step becomes active, and kills the child sequence when the step becomes inactive
- <child_prog> ;** same effect (N attribute is optional)
- <child_prog> (S);** starts the child sequence when the step becomes active. Nothing is done when the step becomes inactive
- <child_prog> (R);** kills the child sequence when the step becomes active. Nothing is done when the step becomes inactive

The SFC sequence specified as an action must be a **child SFC program** of the program currently being edited. Note that using the **S** (Set) or **R** (Reset) qualifiers for an SFC action has exactly the same effect as the **GSTART** and **GKILL** statements, programmed in an **ST** pulse action.

Below is an example of an SFC action. The main SFC program is named **Father**. It has two SFC children, called **SeqMlx** and **SeqPump**. The SFC programming of the father SFC program is:

(* SFC program using SFC actions *)



E.3.5.5 Calling function and function blocks from an action

Sub-programs, functions or function blocks (written in ST, IL, LD or FBD language) or "C" functions and "C" function blocks, can be directly called from an SFC action block, based on the following syntax:

For sub-programs, functions and "C" functions:

```

ACTION (P) :
  result := sub_program ( ) ;
END_ACTION;

```

or

```

ACTION (N) :
  result := sub_program ( ) ;
END_ACTION;

```

For function blocks in "C" or in ST, IL, LD, FBD:

```

ACTION (P) :
  Fbinst(in1, in2);
  result1 := Fbinst.out1;

```

```
result2 := Fbinst.out2;  
END_ACTION;
```

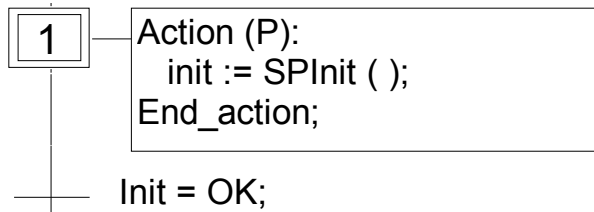
or

```
ACTION (N) :  
Fbinst(in1, in2);  
result1 := Fbinst.out1;  
result2 := Fbinst.out2;  
END_ACTION;
```

Detailed syntax can be found in the ST language section.

Example of a sub-program call in action blocks:

(* SFC program with a sub-program call in an action block *)



E.3.5.6 IL convention

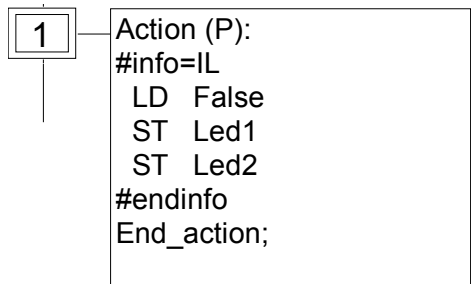
Instruction List (IL) programming may be directly entered in an SFC action block, based on the following syntax:

```
ACTION (P) :    (* or N *)  
#info=IL  
    <instruction>  
    <instruction>  
    ....  
#endinfo  
END_ACTION;
```

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords.

Below is an example of an IL program in an action block:

(* SFC program with an IL sequence in an action block *)



E.3.6 Conditions attached to transitions

At each transition, a **boolean expression** is attached that conditions the clearing of the transition. The condition is usually expressed with ST language or using the LD language (Quick LD editor). This is the **Level 2** of the transition. Other structures may, however, be used:

- ST language convention
- LD language convention
- IL language convention
- Calling function from a transition

Warning: When no expression is attached to the transition, the default condition is **TRUE**.

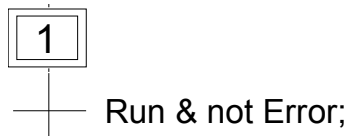
E.3.6.1 ST convention

The **Structured Text** (ST) language can be used to describe the **condition** attached to a transition. The complete expression must have **boolean** type and must be terminated by a **semicolon**, according to the following syntax:

< boolean_expression > ;

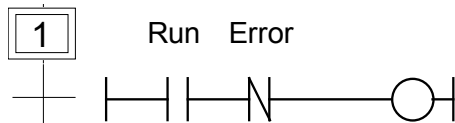
The expression may be a TRUE or FALSE constant expression, a single input or an internal boolean variable, or a combination of variables that leads to a boolean value. Below is an example of ST programming for transitions:

(* SFC program with ST programming for transitions *)



E.3.6.2 LD convention

The **Ladder Diagram** (LD) language can be used to describe the **condition** attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transition value. Below is an example of LD programming for transitions:



E.3.6.3 IL convention

Instruction List (IL) programming may be directly used to describe an SFC transition, according to the following syntax:

```
#info=IL
<instruction>
<instruction>
```

....

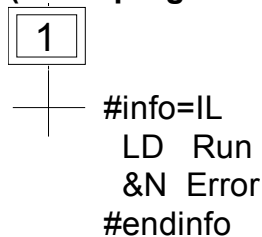
```
#endinfo
```

The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the transition:

current result = 0	➔	condition is FALSE
current result <> 0	➔	condition is TRUE

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords. Below is an example of IL programming for transitions:

(* SFC program with an IL program for transitions *)



E.3.6.4 Calling functions from a transition

Any sub-program or a function (written in FBD, LD, ST or IL language), or a "C" function can be called to evaluate the condition attached to a transition, according to the following syntax:

```
< sub_program > ( ) ;
```

The value returned by the sub-program or the function must be boolean and yields the resulting condition:

return value = FALSE	➔	condition is FALSE
return value = TRUE	➔	condition is TRUE

Example of a sub-program called in a transition:

(* SFC program with sub-program call for transitions *)



—+— 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:



The basic rules implied by the hierarchy structure are:

- SFC programs which have no father are called "**main**" SFC programs
- Main SFC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can only be controlled by its father
- A program cannot control the children of one of its own children

The basic actions that a father SFC program can take to control its child program are:

- | | |
|-----------------------|--|
| Start | (GSTART) Starts the child program: activates each of its initial steps. Children of this child program are not automatically started. |
| Kill (GKILL) | Kills the child program by deactivating each of its active steps. All the children of the child program are also killed. |
| Freeze | (GFREEZE) Suspends the execution of the program (deactivates actions of each of the active steps and suspend transition calculation), and memorises the status of the program steps so the program can be restarted. All the children of the child program are also frozen. |
| Restart | (GRST) Restarts a frozen SFC program by reactivating all the suspended steps. Children of the program are not automatically restarted. |
| Get status | (GSTATUS) Gets the current status (active, inactive or frozen) of a child program. |

E.4 Flow Chart language

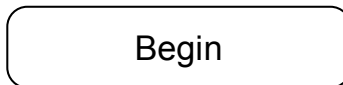
Flow Chart (FC) is a graphic language used to describe **sequential operations**. A Flow Chart diagram is composed of **Actions** and **Tests**. Between Actions and test are **oriented links** representing data flow. Multiple connection links are used to represents divergences and convergences. Actions and Tests can be described with ST, LD or IL languages. Functions and Function blocks of any language (except SFC) can be called from actions and tests. A Flow Chart program can call another Flow Chart program. The called FC program is a **sub-program** of the calling FC program.

E.4.1 FC components

Below are graphic components of the Flow Chart language:

▬ *Beginning of FC chart*

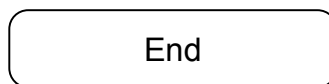
A "**begin**" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated. Below is the drawing of a "begin" symbol:



The "Begin" symbol always has a connection (on the bottom) to the other objects of the chart. A flow chart is not valid if no connection is drawn from "Begin" to another object.

▬ *Ending of FC chart*

An "**end**" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It is possible that no connection is drawn to the "End" symbol (always looping chart), but "End" symbol is still drawn anyway at the bottom of the chart. It represents the final state of the chart, when its execution has been completed. Below is the drawing of an "end" symbol:



The "End" symbol generally has a connection (on the top) to the other objects of the chart. A flow chart may have no connection to the "End" object (always looping chart). The "End" object is still visible at the bottom of the chart in this case.

▬ *FC flow links*

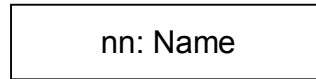
A flow **link** is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Below is the drawing of a flow link:



Two links cannot start from the same source connection point.

FC actions

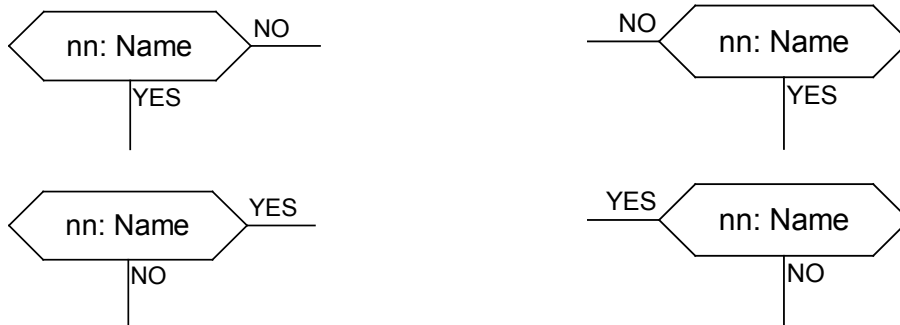
An **action** symbol represents actions to be performed. An action is identified by a number and a name. Below is the drawing of an "action" symbol:



Two different objects of the same chart cannot have the same name or logical number. Programming language for an action can be ST, LD or IL. An action is always connected with links, one arriving to it, one starting from it.

FC conditions

A **condition** represents a boolean **test**. A condition is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. Below are the possible drawings for a condition symbol:



Two different objects of the same chart cannot have the same name or logical number. The programming of a test is either

- an expression in ST, or
- a single rung in LD, with no symbol attached to the unique coil, or
- several instructions in IL. The IL register (or current result) is used to evaluate the condition.

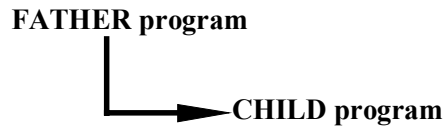
When programmed in ST text, the expression may optionally be followed by a semicolon. When programmed in LD, the unique coil represents the condition value. A condition equal to:

- 0 or FALSE directs the flow to NO
- 1 or TRUE directs the flow to YES

A test is always connected with an arriving link, and both forward connections must be defined.

FC sub-program

The system enables the description of the vertical structure of FC programs. FC programs are organised in a **hierarchy tree**. Each FC program can call other FC programs. Such a program is called a **child program** of the FC program which calls them. FC programs which call FC sub-programs are called **father program**. FC programs are linked together into a main hierarchy tree, using a "father - child" relation:



A **sub-program** symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete. A Flow Chart sub-program is identified by a number and a name, as other programs, functions or function blocks. Below is the drawing of a "sub-program call" symbol:



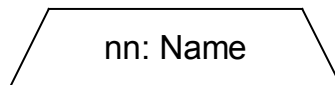
Two different objects of the same chart cannot have the same logical number. The basic rules implied by the FC hierarchy structure are:

- FC programs which have no father are called main FC programs.
- Main FC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can be called only by its father
- A program cannot call the children of one of its own children

The same sub-program may appear several times in the father chart. A Flow Chart sub-program call represents the complete execution of the sub chart. The father chart execution is suspended during the child chart is performed. The sub-program calling blocks must follow the same connection rules as the ones defined for action.

FC I/O specific action

An **I/O specific action** symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. Below is the drawing of an "I/O specific action" symbol:



I/O specific blocks have exactly the same behaviour as standard actions. This covers their properties, ST, LD or IL programming, and connection rules.

FC connectors

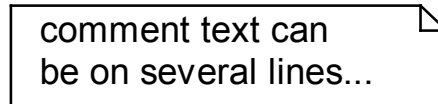
Connectors are used to represent a link between two points of the diagram without drawing it. A connector is represented as a circle and is connected to the source of the flow. The drawing of the connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). Below is the standard drawing of a connector:



A connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

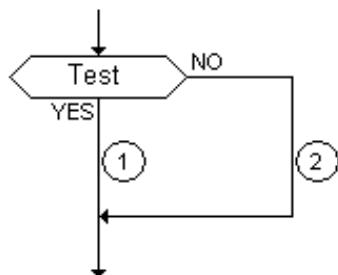
FC comments

A **comment** block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on an unused space of the Flow Chart document window, and is used to document the program. Below is the drawing of a "comment" symbol:



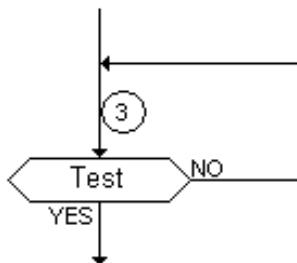
E.4.2 FC complex structures

This section shows **complex structure** examples that can be defined in a Flow Chart diagram. Such structures are combinations of basic objects linked together.



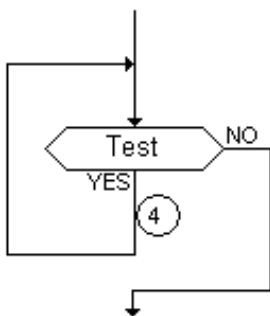
IF / THEN / ELSE

-
- (1) place for "THEN" actions to be inserted
 - (2) place for "ELSE" actions to be inserted



REPEAT / UNTIL

- (3) place for repeated actions to be inserted



WHILE / DO

-
- (3) place for repeated actions to be inserted

E.4.3 FC dynamic behaviour

The **execution** of a Flow Chart diagram can be explained as follows:

- The Begin symbol takes one target cycle
- The End symbol takes one target cycle and ends the execution of the chart. After this symbol is reached, no more actions of the chart are executed.
- The flow is broken each time an item (action, decision) is encountered that has already been reached in the same cycle. In such a case the flow will continue on the next cycle.

Note: Contrary to SFC, an action is not a stable state. There is no repetition of instructions while the action symbol is highlighted.

E.4.4 FC checking

Apart of attached ST, LD or IL programming, some other **syntactic rules** apply to flow chart itself. Below is the list of main rules:

- All "connection" points of all symbols must be wired. (connection to "End" symbol may be omitted)
- All symbols must be linked together (no isolated part should appear)
- All connectors should have valid destination

Other minor syntax errors can be reported:

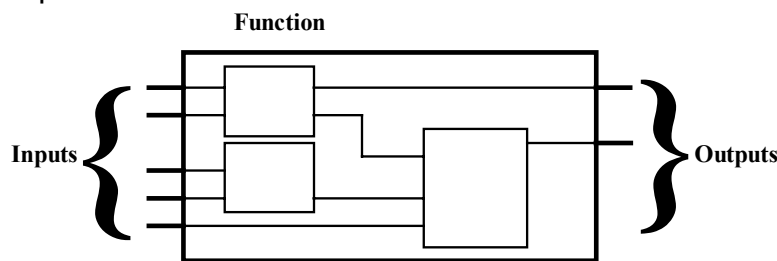
- Empty actions (no programming) are considered as steps during run time scheduling
- Empty tests (no programming) are considered as "always true"

E.5 FBD language

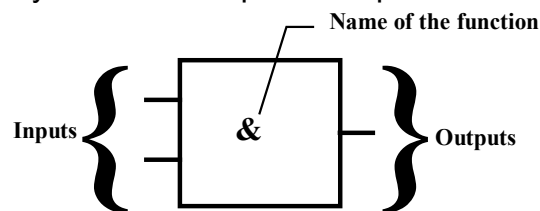
The **Functional Block Diagram** (FBD) is a graphic language. It allows the programmer to build complex procedures by taking existing **functions** from the ISaGRAF library and **wiring** them together in the graphic diagram area.

E.5.1 FBD diagram main format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of **elementary function blocks**. Input and output variables are connected to blocks by **connection lines**. An output of a function block may also be connected to an input of another block.



An entire function operated by an FBD program is built with standard **elementary** function blocks from the ISaGRAF library. Each function block has a fixed number of **input connection points** and a fixed number of **output connection points**. A function block is represented by a single **rectangle**. The inputs are connected on its **left border**. The outputs are connected on its **right border**. An elementary function block performs a single **function** between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well-defined **type**.



Input variables of an FBD program must be connected to input connection points of function blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a **constant** expression, any **internal** or **input** variable, or an **output** variable.

Output variables of an FBD program must be connected to output connection points of function blocks. The type of each variable must be the same as the type expected for the associated block output. An Output for FBD diagram can be any **internal** or **output** variable, or the name of the program (for **sub-programs** only). When an output is the name of the currently edited sub-program, it represents the assignment of the return value for the sub-program (returned to the calling program).

Input and output variables, inputs and outputs of the function blocks are wired together with **connection lines**. Single lines may be used to **connect** two logical points of the diagram:

- An input variable and an input of a function block
- An output of a function block and an input of another block
- An output of a function block and an output variable

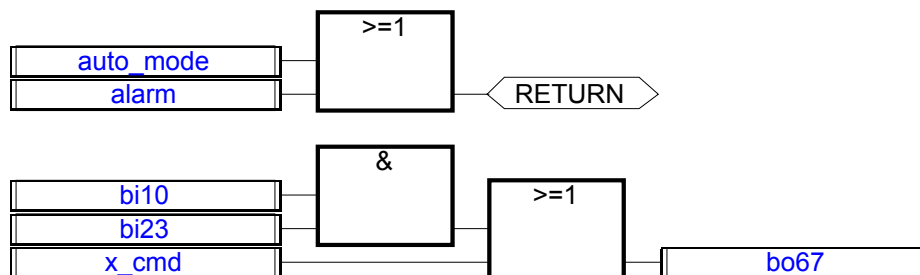
The connection is **oriented**, meaning that the line carries associated data from the left extremity to the right extremity. The left and right extremities of the connection line must be of the **same type**.

Multiple right connection can be used to broadcast an information from its left extremity to each of its right extremities. All the extremities of the connection must be of the same type.

E.5.2 RETURN statement

The "<RETURN>" keyword may occur as a diagram output. It must be connected to a boolean output connection point of a function block. The RETURN statement represents a **conditional end** of the program: if the output of the box connected to the statement has the boolean value **TRUE**, the end (remaining part) of the diagram is not executed.

(* Example of an FBD program using RETURN statement *)



(* ST equivalence: *)

If auto_mode OR alarm Then

Return;

End_if;

bo67 := (bi10 AND bi23) OR x_cmd;

E.5.3 Jumps and labels

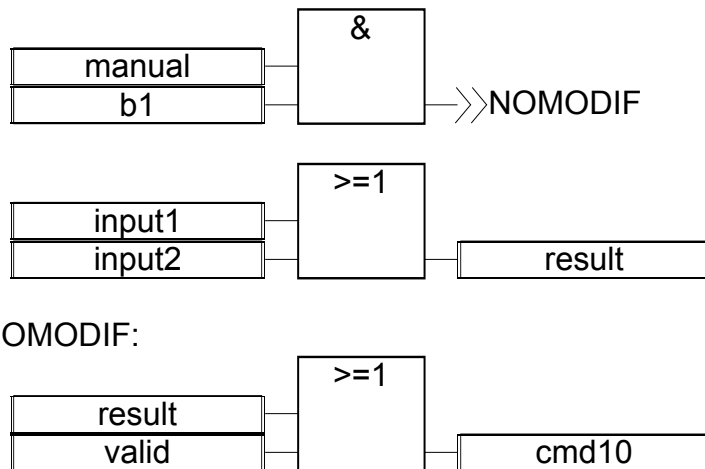
Labels and jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notations are used:

>>LAB jump to a label (label name is "LAB")

LAB: definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the boolean state **TRUE**, the execution of the program directly jumps after the corresponding label symbol.

(* Example of an FBD program using labels and jumps *)



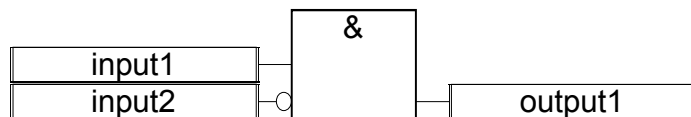
(* IL Equivalence: *)

```
ld    manual
and   b1
jmpc  NOMODIF
ld    input1
or    input2
st    result
NOMODIF: ld result
or    valid
st    cmd10
```

E.5.4 Boolean negation

A single connection line with its right extremity connected to an input of a function block can be terminated by a **boolean negation**. The negation is represented by a small circle. When a boolean negation is used, the left and right extremities of the connection line must have the **BOOLEAN** type.

(* Example of an FBD program using a boolean negation *)



(* ST equivalence: *)

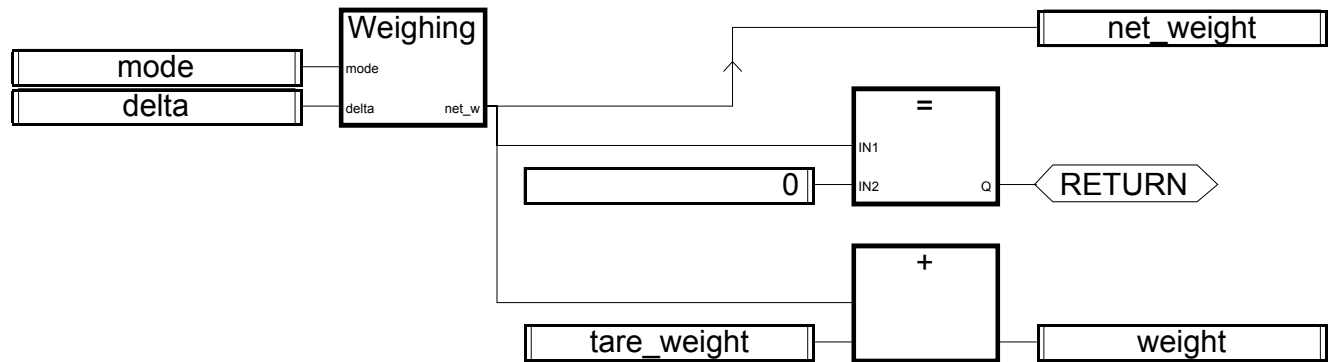
```
output1 := input1 AND NOT (input2);
```

E.5.5 Calling function or function blocks from the FBD

The FBD language enables the calling of sub-programs, functions or function blocks. A sub-program, or function or function block is represented by a function box. The name written in the box is the name of the sub-program or function or function blocks.

In case of a sub-program or a function, the return value is the only output of the function box. A function block can have more than one output.

(* Example of an FBD program using SUB PROGRAM block *)



(* ST Equivalence *)

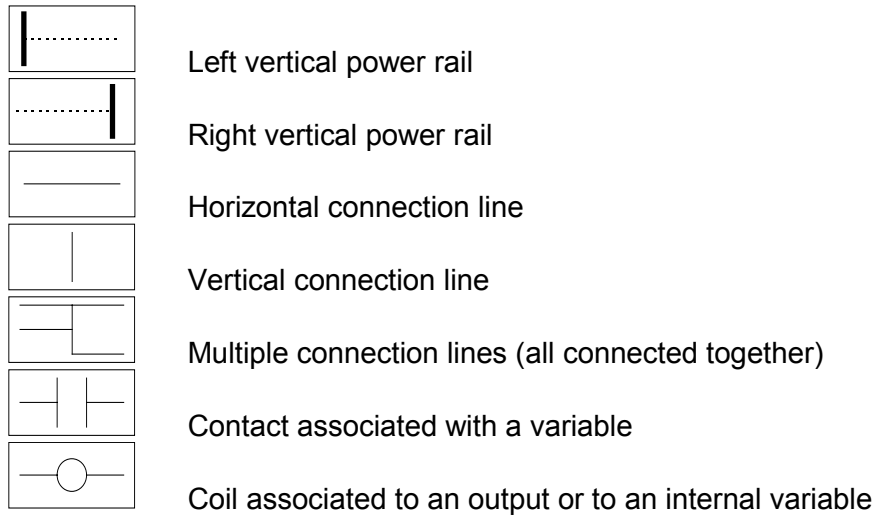
net_weight := Weighing (mode, delta); (* call sub-program *)

If (net_weight = 0) Then Return; End_if;

weight := net_weight + tare_weight;

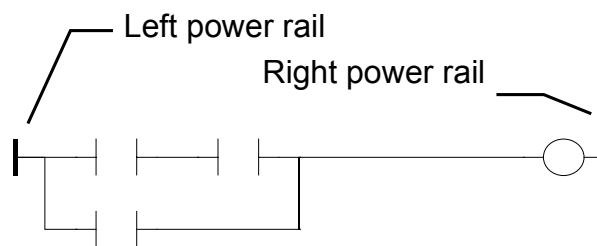
E.6 LD language

Ladder Diagram (LD) is a graphic representation of boolean equations, combining **contacts** (input arguments) with **coils** (output results). The LD language enables the description of tests and modifications of **boolean** data by placing **graphic symbols** into the program chart. LD graphic symbols are organized within the chart exactly as an electric contact diagram. LD diagrams are connected on the left side and on the right side to vertical **power rails**. These are basic graphic components of an LD diagram:

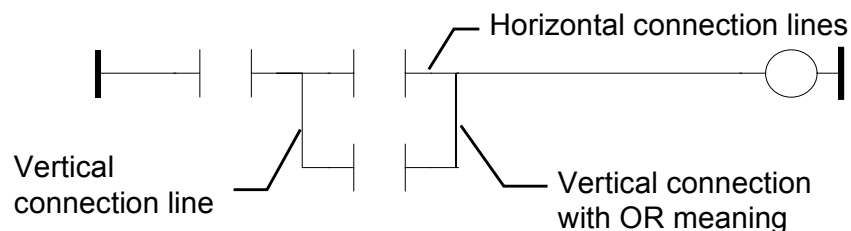


E.6.1 Power rails and connection lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.



LD diagram graphic symbols are connected to power rails or to other symbols by **connection lines**. Connection lines are horizontal or vertical.



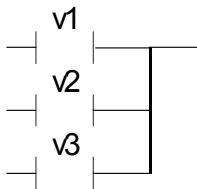
Each line segment has a boolean state **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

E.6.2 Multiple connection

The boolean state given to a single horizontal connection line is the same on the left and on the right extremities of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The boolean state of the extremities of a multiple connection follows logic rules.

A **multiple connection on the left** combines **more than one** horizontal lines connected on the **left** side of a vertical line, and **one** line connected on its **right** side. The boolean state of the right extremity is the **LOGICAL OR** between all the left extremities.

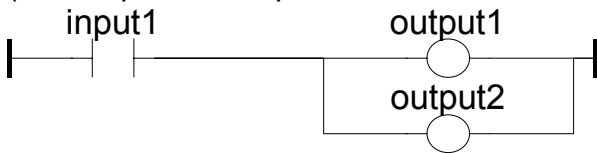
(* Example of multiple LEFT connection *)



(* right extremity state is (v1 OR v2 OR v3) *)

A **multiple connection on the right** combines **one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of the left extremity is propagated into each of the right extremities.

(* Example of multiple RIGHT connection *)



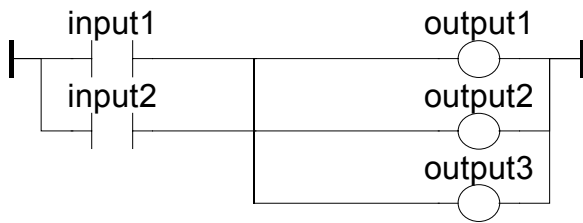
(* ST equivalence: *)

output1 := input1;

output2 := input1;

A **multiple connection on the left and on the right** combines **more than one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of each of the right extremities is the **LOGICAL OR** between all the left extremities

(* Example of multiple LEFT and RIGHT connection *)



(* ST Equivalence: *)

output1 := input1 OR input2;

output2 := input1 OR input2;

output3 := input1 OR input2;

E.6.3 Basic LD contacts and coils

There are several symbols available for input contacts:

- Direct contact
- Inverted contact
- Contacts with edge detection

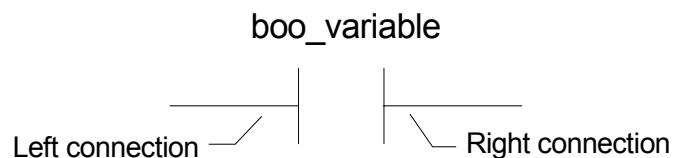
There are several symbols available for output coils:

- Direct coil
- Inverted coil
- SET coil
- RESET coil
- Coils with edge detection

The name of the variable is written above any of these graphic symbols:

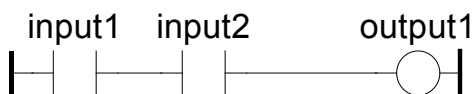
▢ **Direct contact**

A direct contact enables a **boolean operation** between a **connection line** state and a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the value of the variable associated with the contact.

(* Example using DIRECT contacts *)

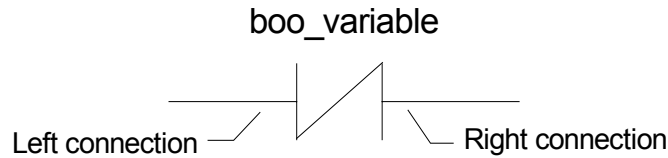


(* ST Equivalence: *)

output1 := input1 AND input2;

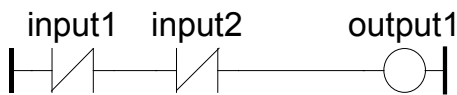
▢ **Inverted contact**

An inverted contact enables a **boolean operation** between a **connection line** state and the boolean negation of a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the **boolean negation** of the value of the variable associated with the contact.

(* Example using INVERTED contacts *)

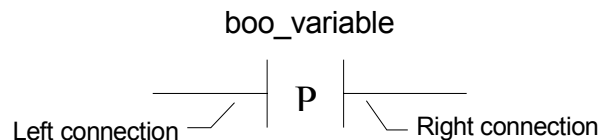


(* ST Equivalence: *)

output1 := NOT (input1) AND NOT (input2);

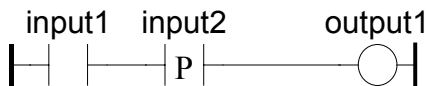
▬ **Contact with rising edge detection**

This contact (positive) enables a **boolean operation** between a **connection line** state and the rising edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **rises** from FALSE to TRUE. It is reset to FALSE in all other cases.

(* Example using RISING EDGE contacts *)



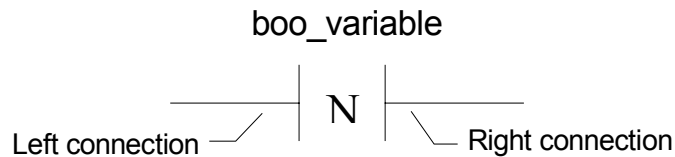
(* ST Equivalence: *)

output1 := input1 AND (input2 AND NOT (input2prev));

(* input2prev is the value of input2 at the previous cycle *)

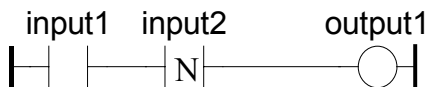
▬ **Contact with falling edge detection**

This contact (negative) enables a **boolean operation** between a **connection line** state and the falling edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **falls** from TRUE to FALSE. It is reset to FALSE in all other cases.

(* Example using FALLING EDGE contacts *)



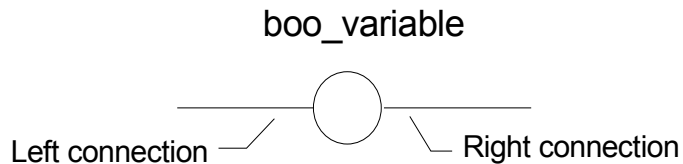
(* ST Equivalence: *)

output1 := input1 AND (NOT (input2) AND input2prev);

(* input2prev is the value of input2 at the previous cycle *)

≡ **Direct coil**

Direct coils enable a **boolean output** of a **connection line** boolean state.

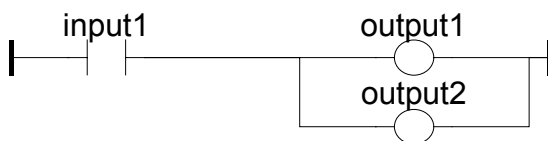


The associated variable is assigned with the boolean **state of the left connection**. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

(* Example using DIRECT coils *)



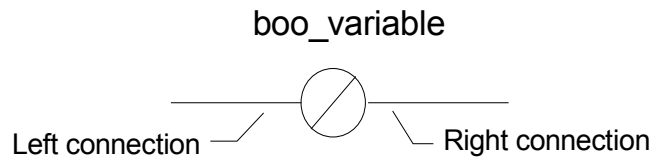
(* ST Equivalence: *)

output1 := input1;

output2 := input1;

≡ **Inverted coil**

Inverted coils enable a **boolean output** according to the boolean **negation** of a **connection line** state.

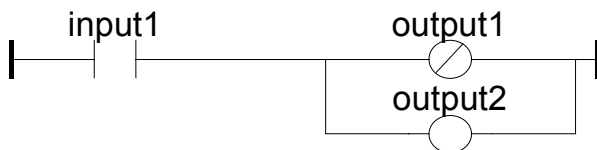


The associated variable is assigned with the boolean **negation** of the **state of the left connection**. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

(* Example using INVERTED coils *)



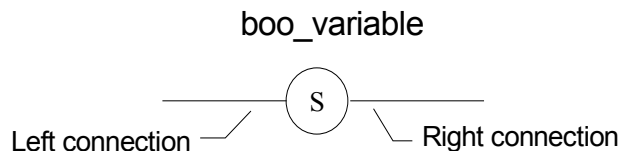
(* ST Equivalence: *)

output1 := NOT (input1);

output2 := input1;

= **SET coil**

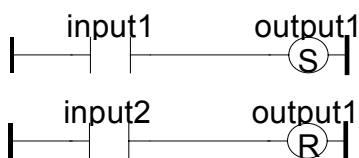
"Set" coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is **SET TO TRUE** when the boolean **state of the left connection** becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)



(* ST Equivalence: *)

IF input1 THEN

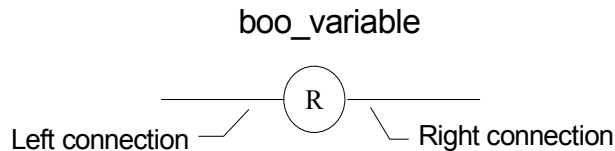
```

output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;

```

= **RESET coil**

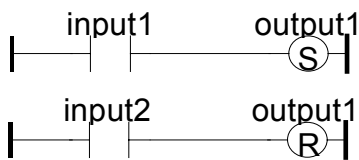
"Reset" coils enable **boolean output** of a **connection line** boolean state.



The associated variable is **RESET TO FALSE** when the boolean **state of the left connection** becomes **TRUE**. The output variable keeps this value until an inverse order is made by a "SET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)



(* ST Equivalence: *)

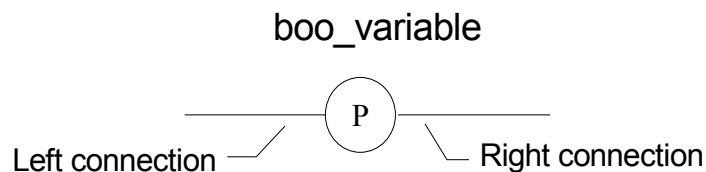
```

IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;

```

= **Coil with rising edge detection**

"Positive" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.

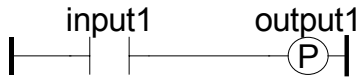


The associated variable is set to **TRUE** when the boolean **state of the left connection** rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of

the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)



(* ST Equivalence: *)

IF (input1 and NOT(input1prev)) THEN

output1 := TRUE;

ELSE

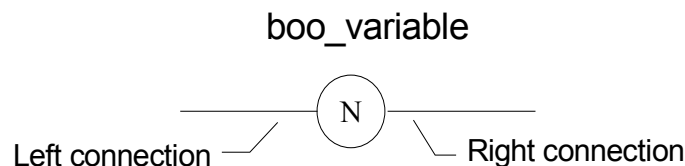
output1 := FALSE;

END_IF;

(* input1prev is the value of input1 at the previous cycle *)

Coil with falling edge detection

"Negative" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)



(* ST Equivalence: *)

IF (NOT(input1) and input1prev) THEN

output1 := TRUE;

ELSE

output1 := FALSE;

END_IF;

(* input1prev is the value of input1 at the previous cycle *)

E.6.4 RETURN statement

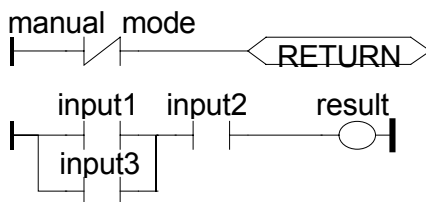
The **RETURN** label can be used as an output to represent a conditional end of the program. No connection can be put on the right of a RETURN symbol.



If the **left connection** line has the **TRUE** boolean state, the program ends without executing the equations entered on the following lines of the diagram.

Note: When the LD program is a sub-program, its name has to be associated with an output coil to set the return value (returned to the calling program).

(* Example using RETURN symbol *)



(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;  
result := (input1 OR input3) AND input2;
```

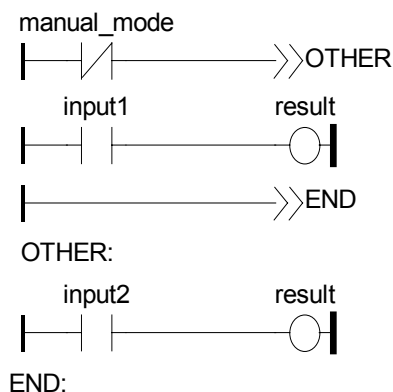
E.6.5 Jumps and labels

Labels, conditional and unconditional JUMPS symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB jump to label named "LAB"
LAB: definition of the label named "LAB"

If the **connection on the left** of the jump symbol has the **TRUE** boolean state, the program execution is driven after the label symbol.

(* Example using JUMP and LABEL symbols *)



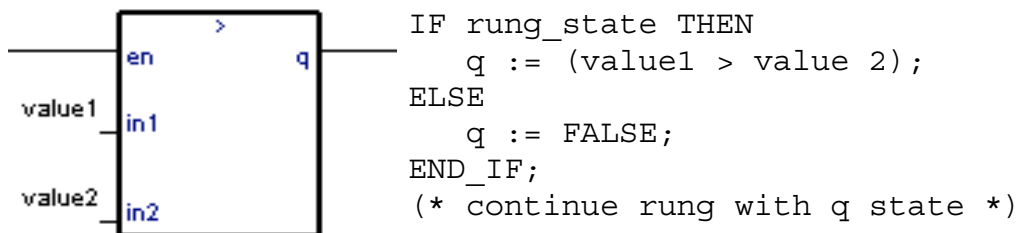
```
(* IL Equivalence: *)
ldn    manual_mode
jmpc   other
ld     input1
st     result
jmp    END
OTHER: ld    input2
st     result
END:   (* end of program *)
```

E.6.6 Blocks in LD

Using the Quick LD editor, you connect function boxes to boolean lines. A function can actually be an operator, a function block or a function. As all blocks do not have always a boolean input and/or a boolean output, inserting blocks in an LD diagram leads to the addition of new parameters EN, ENO to the block interface. The EN, ENO parameters are not added if you use the FBD/LD editor as you can connect the variable with the required type.

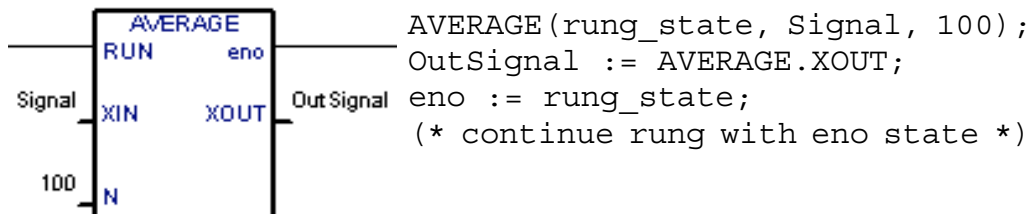
▬ The "EN" input

On some operators, functions or function blocks, the first input does not have boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE. Below is the example of a comparison operator, and the equivalent code expressed in ST:



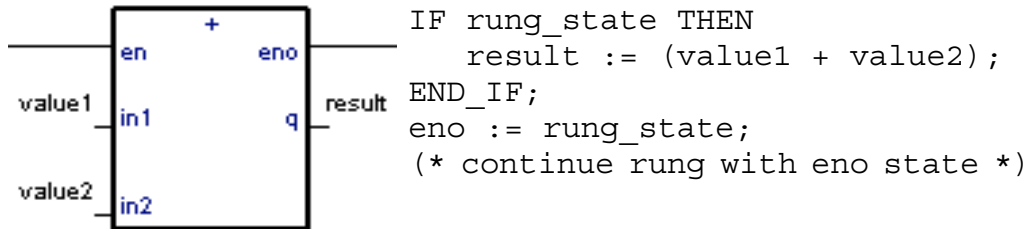
▬ The "ENO" output

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:



▬ 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:



E.7 ST language

ST (**Structured Text**) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST is the default language for the description of the actions within the steps and conditions attached to the transitions of the **SFC** language.

E.7.1 ST main syntax

An ST program is a list of ST **statements**. Each statement ends with a semi-colon (";") separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with **inactive separators** (space character, end of line or tab stops) or by **active separators**, which have a well defined significance (for example, the ">" separator indicates a "greater than" comparison. Comments may be freely inserted into the text. A comment must begin with "(" and ends with ")". Each statement terminates with a semi-colon (";") separator. These are basic types of ST statements:

- **assignment** statement (variable := expression;)
- **sub-program** or **function** call
- **function block** call
- **selection** statements (IF, THEN, ELSE, CASE...)
- **iteration** statements (FOR, WHILE, REPEAT...)
- **control** statements (RETURN, EXIT...)
- special statements for links with other languages such as **SFC**

Inactive separators may be freely entered between active separators, constant expressions and identifiers. ST inactive separators are: **Space** (blank) character, **Tabs** and **End of line** character. Unlike line-formatted languages such as IL, end of lines may be entered anywhere in the program. The rules shown below should be followed when using inactive separators to increase ST program readability:

- Do not write more than one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase readability of lines or paragraphs

E.7.1 Expression and parentheses

ST expressions combine ST **operators** and variable or constant **operands**. For each single expression (combining operands with one ST operator), the **type** of the operands must be the same. This single expression has the same type as its operands, and can be used in a more complex expression. For example :

```
(boo_var1 AND boo_var2)      has BOO type
not (boo_var1)                has BOO type
(sin (3.14) + 0.72) has REAL ANALOG type
(t#1s23 + 1.78)               is an invalid expression
```

Parentheses are used to isolate sub parts of the expression, and to explicitly order the priority of the operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default **priority** between ST operators. For example:

$2 + 3 * 6$ equals $2+18=20$ because multiplication operator has a higher priority

$(2+3) * 6$ equals $5*6=30$ priority is given by parenthesis

Warning: A maximum number of **8** levels of parentheses can be nested within an expression.

E.7.3 Function or function block calls

Standard ST function calls may be used for each of following objects:

- Sub-programs
- Library functions and function blocks written in IEC languages
- "C" functions and function blocks
- Type conversion functions

▬ *Calling sub-programs or functions*

Name: name of the called sub-program
or library function written in IEC language or in "C"

Meaning: calls a ST, IL, LD or FBD sub-program or function or a "C" function
and gets its return value

Syntax: **<variable> := <subprog> (<par1>, ... <parN>);**

Operands: The type of return value and calling parameters must follow
the interface defined for the sub-program.

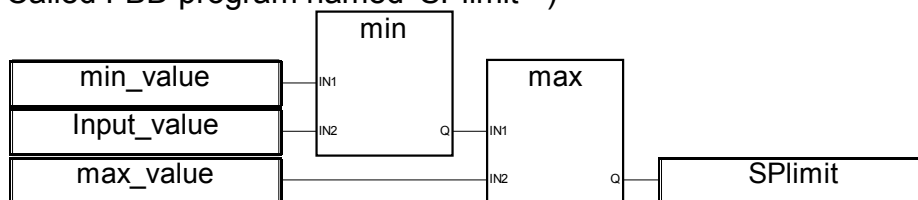
Return value: value returned by the sub-program

Sub-program calls may be used in any expression. They also may be used in an SFC transition.

Example1: Sub-program call

```
(* Main ST program *)
(* gets an analog value and converts it into a limited time value *)
ana_timeprog := SPLimit ( tprog_cmd );
appl_timer := tmr (ana_timeprog * 100);
```

(* Called FBD program named 'SPLimit' *)



Example2: Function call

(* functions used in complex expressions: min, max, right, mlen and left are standard "C" functions *)

```
limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

▬ **Calling function blocks**

Name: name of the function block instance

Meaning: calls a function block from the ISaGRAF library or from the user's library and accesses its return parameters

Syntax: (* call of the function block *)
 <blockname> (<p1>, <p2> ...);
 (gets its return parameters *)
 <result> := <blockname>. <ret_param1>;
 ...
 <result> := <blockname>. <ret_paramN>;

Operands: parameters are expressions which match the type of the parameters specified for that function block

Return value: See Syntax to get the return parameters.

Consult the ISaGRAF library to find the meaning and type of each function block parameter.
 The function block instance (name of the copy) must be declared in the dictionary

Example :

```
(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;
```

E.7.4 ST specific boolean operators

The following boolean operators are specific to the ST language:

- REDGE rising edge detection
- FEDGE falling edge detection

Other standard boolean operators such as:

- NOT boolean negation
- AND (&) logical AND
- OR logical OR
- XOR logical exclusive OR

can be used. Their description is to be found in the section 'Standard operators, function blocks and functions'.

▬ **"REDGE" operator**

Name: REDGE

Meaning: evaluates the rising edge of a complete boolean expression

Syntax: <edge> := REDGE (<boo_expression>, <memo_variable>);

Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store the last state of the expression

Return value: TRUE when the expression changes from FALSE to TRUE
FALSE for all other cases

The rising edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. This operator can be used to describe the condition attached to an SFC transition.

Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "xxx", a unique internal variable named "EDGE_xxx" should be declared and used it in the REDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other REDGE evaluations.

Example:

(* ST program using REDGE operator *)

(* this program counts the rising edges of a boolean input *)

(* Bi120 is an input boolean variable *)

(* Edge_Bi120 is the memory of the Bi120 variable state *)

If REDGE (Bi120, Edge_Bi120) Then

Counter := Counter + 1;

End_if;

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of R_TRIG standard block. It has been kept for compatibility reasons.

▬ "FEDGE" operator

Name: FEDGE

Meaning: evaluates the falling edge of a boolean expression

Syntax: <edge> := FEDGE (<boo_expression>, <memo_variable>);

Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store the last state of the expression

Return value: TRUE when the expression changes from TRUE to FALSE
FALSE for all other cases

The falling edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. The operator can be used to describe the condition attached to an SFC transition.

Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the FEDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other FEDGE evaluations.

Example:

(* ST program using FEDGE operator *)

(* this program counts the falling edges of a boolean input *)

(* Bi120 is an input boolean variable *)

(* Edge_Bi120 is the memory of the Bi120 variable state *)

If FEDGE (Bi120, Edge_Bi120) Then

Counter := Counter + 1;

End_if;

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of F_TRIG standard block. It has been kept for compatibility reasons.

E.7.5 ST basic statements

The basic statements of the ST language are:

- Assignment
- RETURN statement
- IF-THEN-ELSIF-ELSE structure
- CASE statement
- WHILE iteration statement
- REPEAT iteration statement
- FOR iteration statement
- EXIT statement

▣ *Assignment*

Name: :=

Meaning: assigns a variable to an expression

Syntax: <variable> := <any_expression> ;

Operands: variable must be internal or output
 variable and expression must have the same type

The expression can be a call to a sub-program or a function from the ISaGRAF library

Example:

(* ST program with assignments *)

(* variable <=> variable *)

bo23 := bo10;


```
(* variable <=> expression *)
bo56 := bx34 OR alm100 & (level >= over_value);
result := (100 * input_value) / scale;
```

```
(* assignment with sub-program return value *)
rc := PSelect ( );
```

```
(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

▬ **RETURN statement**

Name: **RETURN**

Meaning: terminates the execution of the current program

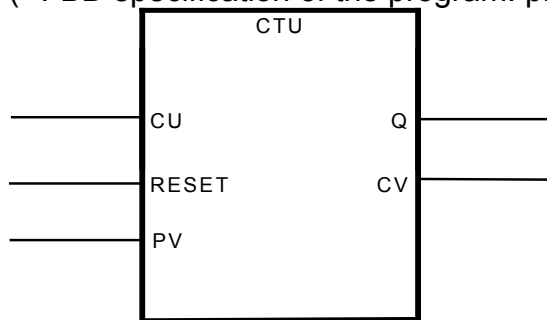
Syntax: **RETURN ;**

Operands: (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

Example:

(* FBD specification of the program: programmable counter *)



(* ST implementation of the program, using RETURN statement *)

```
If not (CU) then
    Q := false;
    CV := 0;
    RETURN; (* terminates the program *)
end_if;
```

```
if R then
    CV := 0;
else
    if (CV < PV) then
        CV := CV + 1;
    end_if;
end_if;
Q := (CV >= PV);
```

▬ **IF-THEN-ELSIF-ELSE statement**

Name: IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF

Meaning: executes one of two lists of ST statements
selection is made according to the value
of a boolean expression

Syntax: IF <boolean_expression> THEN

<statement> ;

<statement> ;

...

ELSIF <boolean_expression> THEN

<statement> ;

<statement> ;

...

ELSE

<statement> ;

<statement> ;

...

END_IF;

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE.

Example:

(* ST program using IF statement *)

IF manual AND not (alarm) THEN

level := manual_level;

bx126 := bi12 OR bi45;

ELSIF over_mode THEN

level := max_level;

ELSE

level := (lv16 * 100) / scale;

END_IF;

(* IF structure without ELSE *)

If overflow THEN

alarm_level := true;

END_IF;

▢ *CASE statement*

Name: CASE ... OF ... ELSE ... END_CASE

Meaning: executes one of several lists of ST statements
selection is made according to an integer expression

Syntax: CASE <integer_expression> OF

<value> : <statements> ;

<value> , <value> : <statements> ;

...

ELSE

<statements> ;

END_CASE;

Case values must be integer constant expressions. Several values, separated by comas, can lead to the same list of statements. The ELSE statement is optional.

Example:

(* ST program using CASE statement *)

```
CASE error_code OF
  255:  err_msg := 'Division by zero';
        fatal_error := TRUE;
  1:err_msg := 'Overflow';
  2, 3:  err_msg := 'Bad sign';
ELSE
  err_msg := 'Unknown error';
END_CASE;
```

≡ WHILE statement

Name: **WHILE ... DO ... END_WHILE**

Meaning: iteration structure for a group of ST statements
 the "continue" condition is evaluated BEFORE any iteration

Syntax: **WHILE <boolean_expression> DO**

 <statement> ;

 <statement> ;

 ...

END_WHILE ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

Example:

(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)

(* on a serial port *)

string := ""; (* empty string *)

nbchar := 0;

```
WHILE ((nbchar < 16) & ComIsReady ( )) DO
```

```
  string := string + ComGetChar ( );
```

```
  nbchar := nbchar + 1;
```

```
END_WHILE;
```

≡ REPEAT statement

Name: **REPEAT ... UNTIL ... END_REPEAT**

Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration

Syntax: **REPEAT**
 <statement> ;
 <statement> ;
 ...
 UNTIL <boolean_condition>
 END_REPEAT ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

Example:

```
(* ST program using REPEAT statement *)  
  
(* this program uses specific "C" functions to read characters *)  
(* on a serial port *)
```

```
string := ""; (* empty string *)  
nbchar := 0;  
IF ComIsReady ( ) THEN  
  REPEAT  
    string := string + ComGetChar ( );  
    nbchar := nbchar + 1;  
  UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )  
  END_REPEAT;  
END_IF;
```

▬ **FOR statement**

Name: **FOR ... TO ... BY ... DO ... END_FOR**

Meaning: executes a limited number of iterations,
 using an integer analog index variable

Syntax: **FOR <index> := <mini> TO <maxi> BY <step> DO**
 <statement> ;
 <statement> ;
 END_FOR;

Operands: **index:** internal analog variable increased at any loop
 mini: initial value for index (before first loop)
 maxi: maximum allowed value for index
 step: index increment at each loop

The [BY step] statement is optional. If not specified, the increment step is 1

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during FOR iterations.

This is the "while" equivalent of a FOR statement:

```

index := mini;
while (index <= maxi) do
    <statement> ;
    <statement> ;
    index := index + step;
end_while;

```

Example:

(* ST program using FOR statement *)
 (* this program extracts the digit characters of a string *)

```

length := mlen (message);
target := ""; (* empty string *)
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
    END_IF;
END_FOR;

```

EXIT statement

Name: EXIT
Meaning: exit from a FOR, WHILE or REPEAT iteration statement
Syntax: EXIT;
Operands: (none)

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

Example:

(* ST program using EXIT statement *)
 (* this program searches for a character in a string *)

```

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code = searched_char) THEN
        found := YES;
        EXIT;
    END_IF;
END_FOR;

```

E.7.6 ST extensions

The following functions are extensions of the ST language:

- TSTART - TSTOP: timer control

The following statements and functions are available to control the execution of the SFC child programs. They may be used inside ACTION(): ... END_ACTION; blocks in SFC steps.

- GSTART starts an SFC program
- GKILL kills an SFC program
- GFREEZE freezes an SFC program
- GRST restarts a frozen SFC program
- GSTATUS gets current status of an SFC program

Warning: These functions are not in the IEC 1131-3 norm.

Easy equivalent can be found for GSTART and GKILL using the following syntax in the SFC step:

```
child_name(S); (* equivalent to GSTART(child_name); *)
child_name(R); (* equivalent to GKILL(child_name); *)
```

The following fields can be used to access the status of an SFC step:

GSnnn.x boolean value that represents the activity of the step

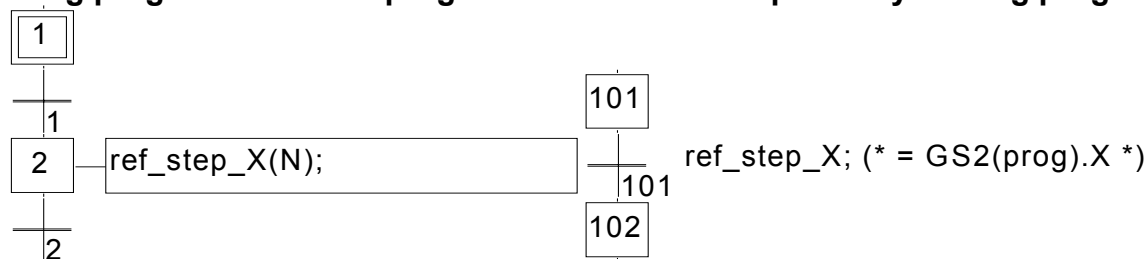
GSnnn.t time elapsed since the last activation of the step
 ("nnn" is the reference number of the SFC step)

It is also possible to test the activity of a step declared in another SFC program, by using the following syntax:

GSnnn(progname).x

Warning: referencing a step of an other program, using this syntax is not in the IEC 1131-3 norm. An easy way to do the same respecting IEC rules, is to declare a global boolean variable in the dictionary which will represent the step activity to be tested (for example ref_step_X). Then you insert in the step, the variable with the N qualifier (ref_step_X(N);). Then in the program which wants to test the activity of the step, you use the variable.

Prog program the other program which needs step activity of Prog program



▣ **TSTART statement**

Name: TSTART

Meaning: starts the counting of a timer variable

timer value is not modified by the TSTART command, i.e. the counting starts from the current value of the timer.

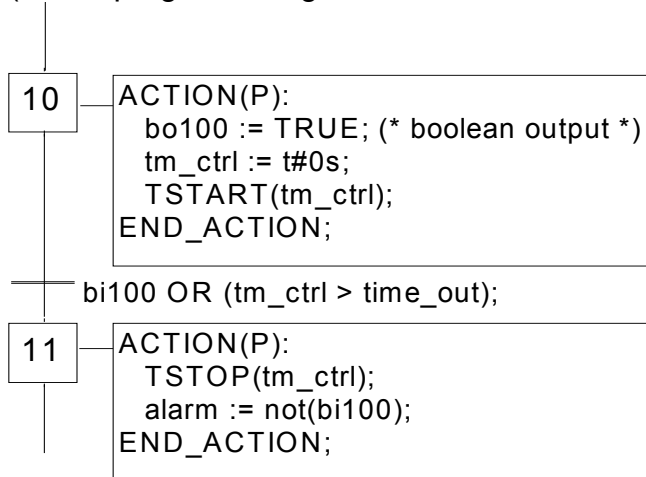
Syntax: TSTART (<timer_variable>);

Operands: any inactive timer variable

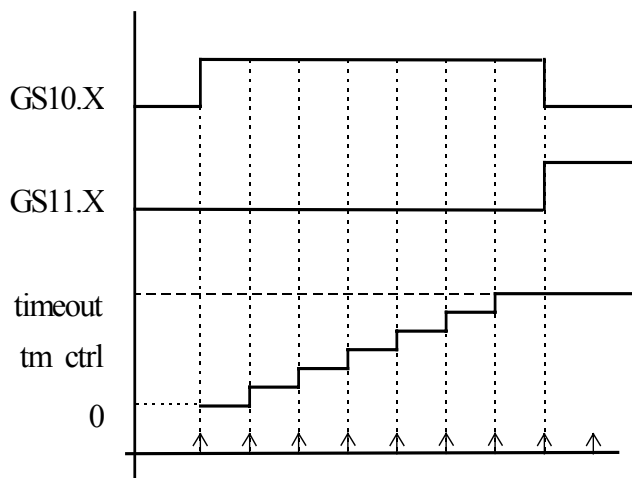
Return value: (none)

Example:

(* SFC program using TSTART and TSTOP statements *)



Time diagram if bi100 is always FALSE:



The timer keeps the same value during one cycle.

▢ **TSTOP statement**

Name: TSTOP

Meaning: stops updating a timer variable
timer value is not modified by the TSTOP command

Syntax: TSTOP (<timer_variable>);

Operands: any active timer variable

Return value: (none)

Example: See TSTART (the function is described above)

▢ **GSTART statement**

Name: GSTART

Meaning: starts a child SFC program by putting a token
into each of its initial steps

Syntax: **GSTART (<child_program>);**

Operands: the specified SFC program must be a child of the one
 in which the statement is written

Return value: (none)

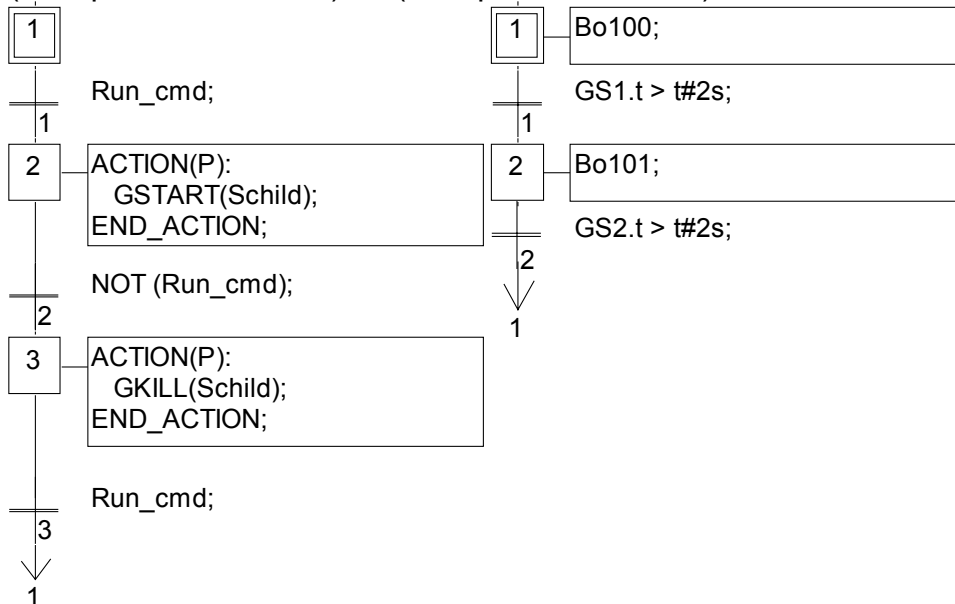
Children of the child program are not automatically started by the GSTART statement.

Note: As GSTART is not in the IEC 1131-3 norm, prefer the use of the S qualifier, with the
 following syntax to start a child SFC:

Child_name(S);

Example: Use of GSTART and GKILL

(* 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>);**

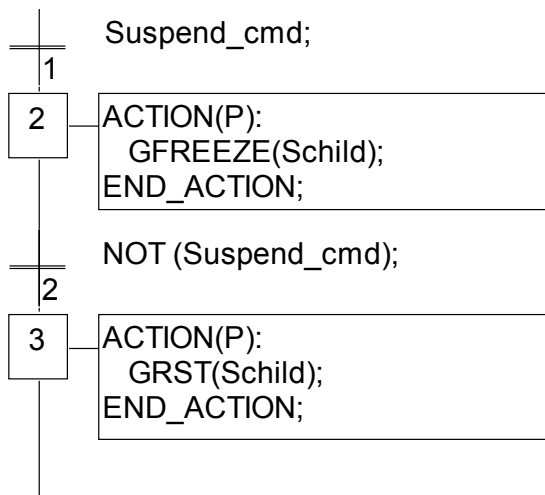
Operands: the specified SFC program must be a child of the one
 in which the statement is written

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:



≡ **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

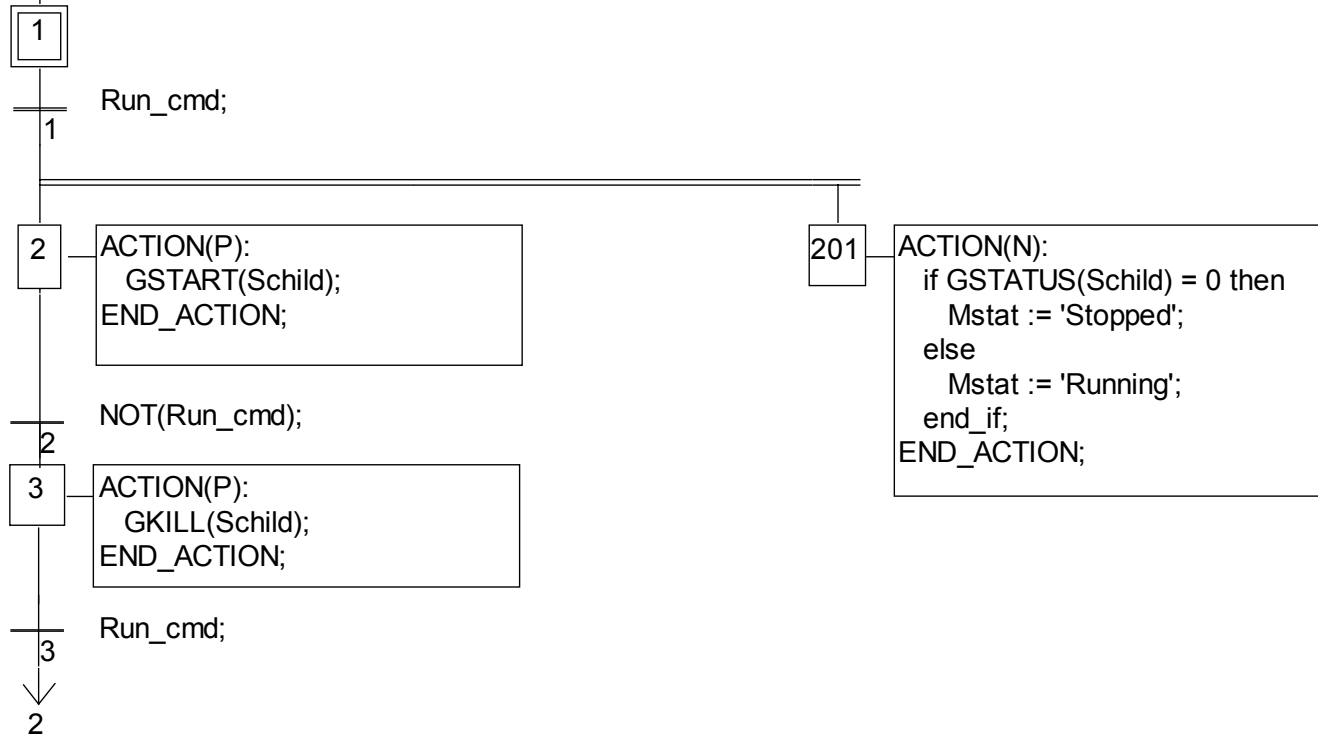
Return value: 0 = program is inactive (killed)

 1 = program is active (started)

 2 = program is frozen

Note: GSTATUS is not in the IEC 1131-3 norm.

Example:



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:

<i>Label</i>	<i>Operator</i>	<i>Operand</i>	<i>Comments</i>
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

▬ **Labels**

A **label** followed by a colon (':') may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Naming labels must conform to the following rules:

- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters**, **digits** or '_' character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a variable.

▬ **Operator modifiers**

The available operator modifiers are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

- N** boolean negation of the operand
- (** delayed operation
- C** conditional operation

The '**N**' modifier indicates a boolean negation of the operand. For example, the instruction **ORN IX12** is interpreted as: **result := result OR NOT (IX12)**.

The parenthesis '(' modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis ')' operator is encountered.

The '**C**' modifier indicates that the attached instruction must be executed only if the current result has the boolean value TRUE (different than 0 for non-boolean values). The '**C**' modifier can be combined with the '**N**' modifier to indicate that the instruction must be executed only if the current result has the boolean value FALSE (or 0 for non-boolean values).

▬ **Delayed operations**

Because there is only one IL register (current result), some operations may have to be delayed, so that the execution order or the instructions can be changed. Parentheses are used to indicate delayed operations:

'(' is a modifier indicates the operation to be delayed
)' is an operator executes the delayed operation

The opening parenthesis '(' modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis ')' operator is encountered. For example, following sequence:

```
AND( IX12
OR  IX35
)
```

is interpreted as:

result := result AND (IX12 OR IX35)

E.8.2 IL operators

The following table summarizes the standard operators of the IL language:

<i>Operator</i>	<i>Modifiers</i>	<i>Operand</i>	<i>Description</i>
LD	N	Variable, constant	Loads operand
ST	N	Variable	Stores current result
S R		BOO variable BOO variable	Sets to TRUE Resets to FALSE
AND & OR XOR	N (N (N (N (N (BOO BOO BOO BOO	boolean AND boolean AND boolean OR exclusive OR
ADD SUB MUL DIV	((((variable, constant variable, constant variable, constant variable, constant	Addition Subtraction Multiplication Division

GT	(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 Jumps to label Returns from sub-program
JMP	C N	instance name	
RET	C N	Label	
)			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:

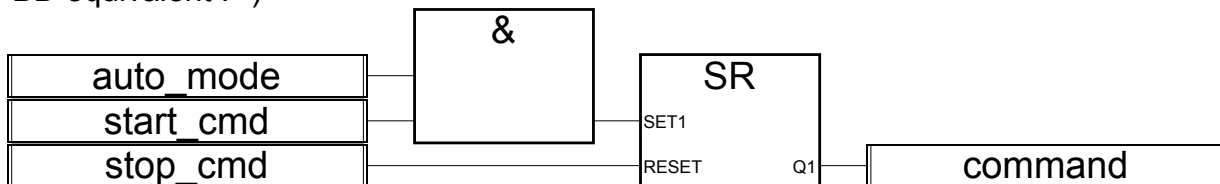
(* Calling function block SR : SR1 is an instance of SR *)

```

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

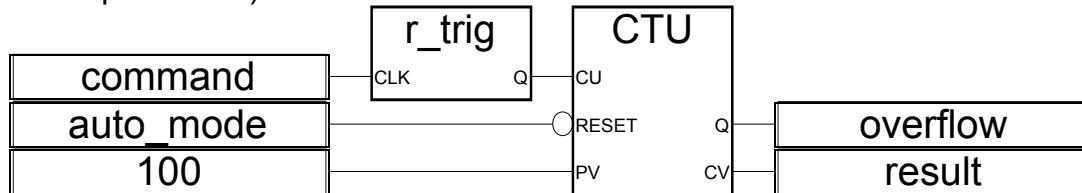
```

```

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: *)



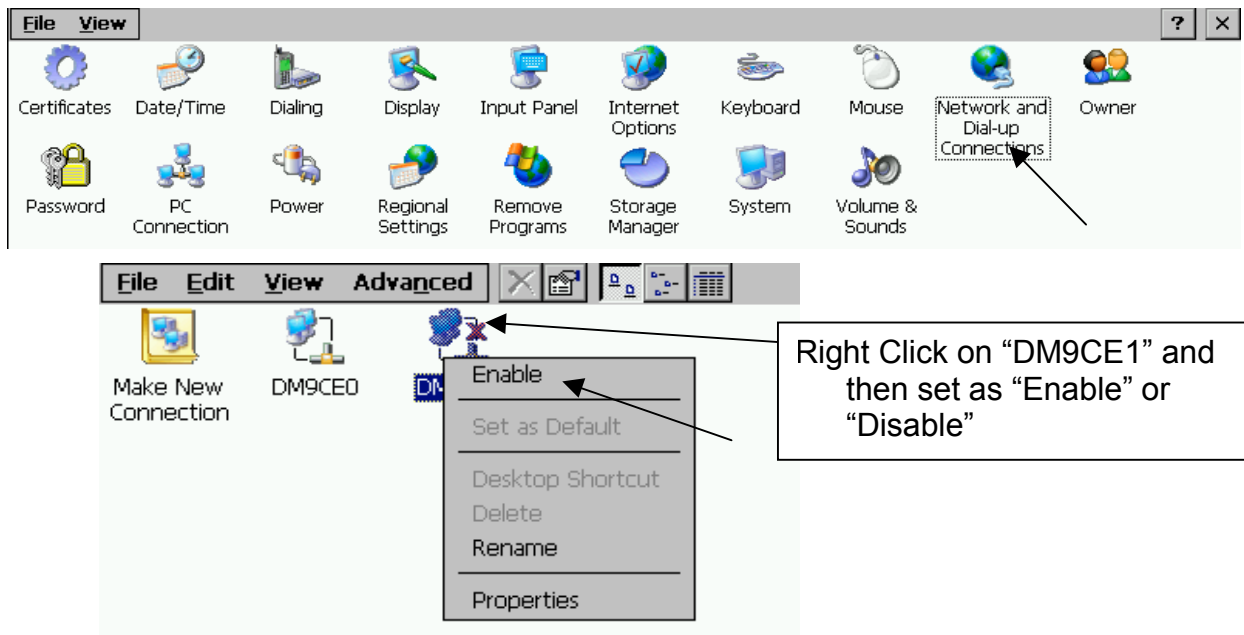
Appendix F: How to Enable/Disable W-8x47's LAN2

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 Dial-up Connections” to set as LAN2: DM9CE1 Enable or Disable



Then run “Start” – “Programs” – “Wincon Utility”, click “Save and Reboot” to save the setting.

