Functions and Function Blocks in ISaGRAF for SCM03

This Technical Note contains detailed information on all the Function Blocks available for PLC programming in the SCM03 controller.

Contents

1. Introduction

This document contains the list of Functions and Function Blocks written for the ISaGRAF softlogic system installed on the SCM03 board. Before using it, please make sure that the version you have is the most recent one, otherwise you might miss recently added blocks and changes to previously written blocks.

This list is intended for use as a reference by PLC programmers writing applications for the HMIControl systems based on ISaGRAF PLC language interpreter.

Throughout the document, the generic term "Function Block" will be used as a reference to Functions, Conversion Functions and Function Blocks, as defined in the ISaGRAF User's Manual. This Manual should also be used as a reference to standard Function Blocks written by CJ International and delivered as a part of the ISaGRAF package.

2. Functions

A function has **at most one output** and **no internal memory**. Due to this lack of information transfer between calls, for the same set of inputs, a function will always return the same output value.

Each function belongs to one of the following classes:

- Arithmetic functions
- Math functions
- Trigonometric functions
- Boolean functions
- Logic functions
- Comparison functions
- Register control functions
- Data manipulation functions
- Data conversion functions
- String management functions
- Array manipulation functions
- System access functions
- Hardware specific functions

2.1 Arithmetic Functions

Standard Arithmetic Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

ADD	Addition (INTEGERs and REALs, extensible)
SUB	Subtraction (INTEGERs and REALs)
MUL	Multiplication (INTEGERs and REALs, extensible)
DIV	Division (INTEGERs and REALs)

Currently no functions written by EXOR have been added to this group.

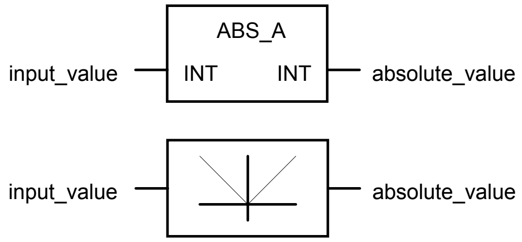
2.2 Math Functions

Standard Math Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

ABS	Absolute value of a REAL number
EXPT	Exponentiation of REAL base by the INTEGER exponent
LOG	Logarithm to the base 10 of a REAL number
POW	Power Calculation
SQRT	Square root of a REAL number
TRUNC	Truncation of a REAL number with REAL output

ABS_A



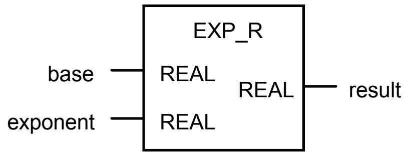
Short description: Absolute analog (integer) value

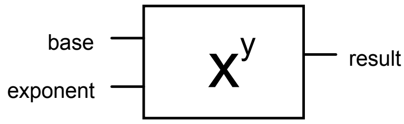
Description: -

Call parameters: input_value (INT)
Return parameter: absolute_value (INT)
Prototype: absolute := abs_a (value);

Remarks: This is the "analog" equivalent of the standard ABS function.

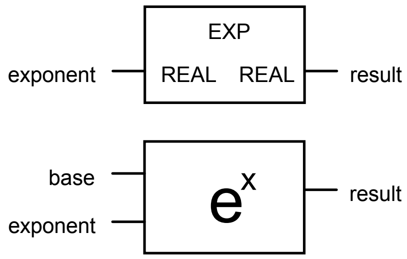
EXP_R





Short description:	Exponentiation: real base, real exponent
Description:	-
Call parameters:	base (REAL) exponent (REAL)
Return parameter:	result (REAL)
Prototype:	result := EXP_R (base, exp);
Remarks:	This is an extension of the corresponding standard function which allows only an integer exponent to be applied to a real base.

EXP_E

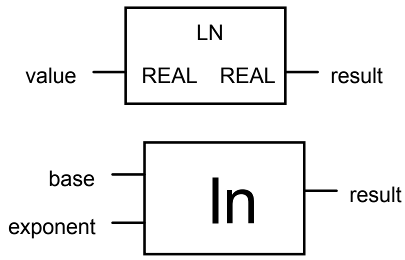


Short description: Natural exponential function (base e) with real exponent

Description: -

Call parameters: exponent (REAL)
Return parameter: result (REAL)
Prototype: result := EXP_E (rex);

LN_E



Short description: Natural logarithm (base e) of a real number

Description: -

Call parameters: value (REAL)
Return parameter: result (REAL)
Prototype: logval := ln_e (rval);

2.3 Trigonometric Functions

Standard Trigonometric Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

ACOS	Arc cosine of a REAL number
ASIN	Arc sine of a REAL number
ATAN	Arc tangent of a REAL number
COS	Cosine of a REAL number
SIN	Sine of a REAL number
TAN	Tangent of a REAL number

Currently no functions written by EXOR have been added to this group.

2.4 Boolean Functions

Standard Boolean Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

AND	Boolean AND (extensible)
OR	Boolean OR (extensible)
XOR	Boolean XOR (extensible)

Currently no functions written by EXOR have been added to this group.

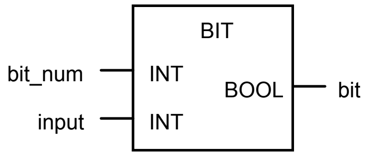
2.5 Logic Functions

Standard Logic Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

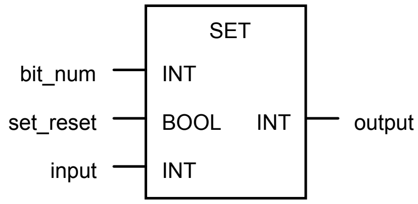
AND	Analog (INTEGER) bit to bit AND (extensible)
OR	Analog (INTEGER) bit to bit OR (extensible)
XOR	Analog (INTEGER) bit to bit XOR (extensible)

BIT



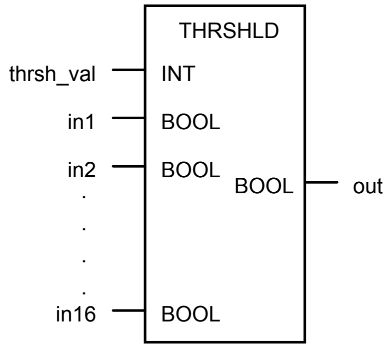
Short description:	Test indicated bit of given integer		
Description:	If bit_num is less than 0 or greater than 31, bit 0 is tested.		
Call parameters:	bit_num:	Bit number in range 0 to 31	(INT)
	input:	Integer whose bit is to be tested	(INT)
Return parameter:	bit:	Tested bit	(BOOL)
Prototype:	tbit := bit (n,int);		

SET



Short description:	Sets or resets indicated bit in an integer		
Description:	If bit_num is less than 0 or greater than 31, the input will be copied to the output unchanged.		
Call parameters:	bit_num:	Bit number in range 0 to 31	(INT)
	set_reset:	New value of the bit	(BOOL)
	input:	Integer whose bit is to be changed	(INT)
Return parameter:	output:	Modified integer	(INT)
Prototype:	new := set (bitnum,sr,old);		

THRSHLD

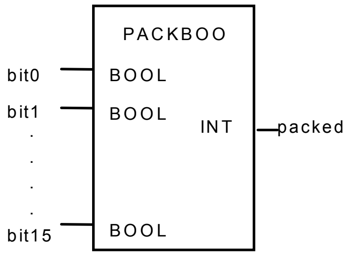


Short description:	Threshold element
Description:	out is set to TRUE if more than thrsh_val inputs are set to TRUE. thrsh_val should be in range 1 to 16. If it is not within range, out is set to TRUE.

Call parameters:

Call parameters:	thrsh_val	(INT)
	in1	(BOOL)
	in2	(BOOL)
	....	....
	in16	(BOOL)
Return parameter:	out	(BOOL)
Prototype:	alarm := THRSHLD (maxnum,in1,...,in16);	

PACKBOO



Short description:	Pack 16 boolean variables into one analog variable
Description:	Packing bits into a word is sometimes needed to prepare data for communication with other devices or I/O equipment.
Call parameters:	bit0 (BOOL) …………… bit15 (BOOL)
Return parameter:	packed (INT)
Prototype:	packed := packboo (b0,b1,b2,…….,b15);

2.6 Comparison Functions

Standard Comparison Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

LT	Less than (all data types)
LE	Less than or equal (all data types except TMR)
GT	Greater than (all data types)
GE	Greater than or equal (all data types except TMR)
EQ	Equal to (all data types except TMR)
NE	Not equal to (all data types except TMR)

Currently no functions written by EXOR have been added to this group.

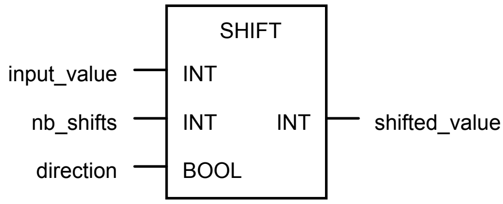
2.7 Register Control Functions

Standard Register Control Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

ROL	Rotate INTEGER left
ROR	Rotate INTEGER right
SHL	Shift INTEGER left
SHR	Shift INTEGER right

SHIFT



Short description:	Shifts an analog value left or right arithmetically
Description:	Input value is copied to the output without change if the number of shifts is less than or equal to zero. If the number of shifts is greater than or equal to 32, the result is equal to all zeros for left shift and either to all zeros or all ones for right shift, depending on the MSB of the input value. Shifting is done arithmetically, meaning that: - when a number is shifted to the left, zeros are filled in at the right end - when a number is shifted to the right, MSB is copied to the bit right of it. For right shift direction is FALSE, for left shift direction is TRUE.

Call parameters:	input_value	(INT)	
	nb_shifts	(INT)	
	direction	(BOOL)	
Return parameter:	shifted_value	(INT)	
Prototype:	result := shift (ival, nshifts, dir);		
Example:	input_value:	0100....10101	
	shifted_value:	00100....1010	(1 shift right with MSB=0)
	input_value:	1100....10101	
	shifted_value:	11100....1010	(1 shift right with MSB=1)
	input_value:	10011....0011	
	shifted_value:	0011....00110	(1 shift left)

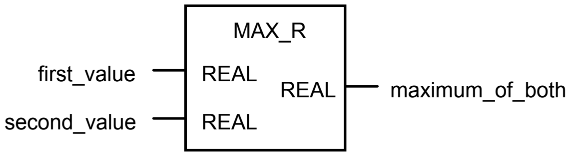
2.8 Data Manipulation Functions

Standard Data Manipulation Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

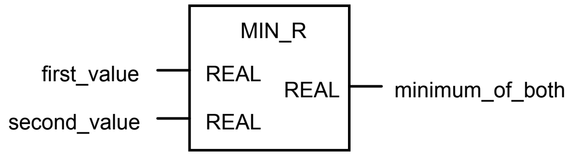
MIN	Minimum of INTEGERs (extensible)
MAX	Maximum of INTEGERs (extensible)
MOD	Modulo (INTEGER division remainder)
MUX4	Multiplexer (4 INTEGER inputs)
MUX8	Multiplexer (8 INTEGER inputs)
ODD	Odd parity for na INTEGRER
SEL	Binary selector
LIMIT	INTEGER limiter
RAND	Random INTEGER generator

MAX_R



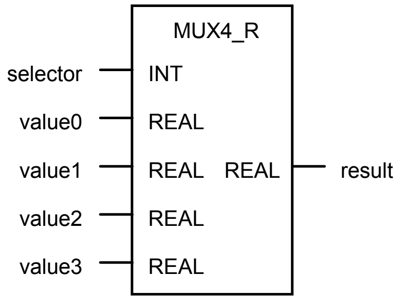
Short description:	Maximum of two real values	
Description:	-	
Call parameters:	first_value	(REAL)
	second_value	(REAL)
Return parameter:	maximum_of_both	(REAL)
Prototype:	maxval := max_r (val1, val2);	
Remarks:	This is the "real" equivalent of the standard MAX function. It does not support an extensible number of inputs.	

MIN_R



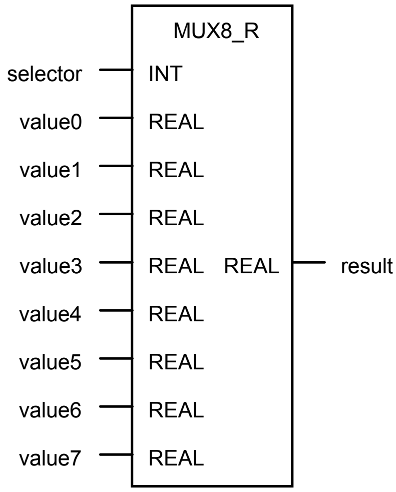
Short description:	Minimum of two real values	
Description:	-	
Call parameters:	first_value	(REAL)
	second_value	(REAL)
Return parameter:	minimum_of_both	(REAL)
Prototype:	minval := min_r (val1, val2);	
Remarks:	This is the "real" equivalent of the standard MIN function. It does not support an extensible number of inputs.	

MUX4_R



Short description:	Select one of four real values For any other selector value, result is set to 0.
Call parameters:	selector (INT) value0 (REAL) value1 (REAL) value2 (REAL) value3 (REAL)
Return parameter:	result (REAL)
Prototype:	result := mux4_r (select, val0, val1, val2, val3);
Remarks:	This is the "real" equivalent of the standard MUX4 function. It does not support an extensible number of inputs.

MUX8_R



Short description: Select one of eight real values

Description: If selector is : 0 then result = value0
1 value1
... ...
7 value7
For any other selector value, result is set to 0.

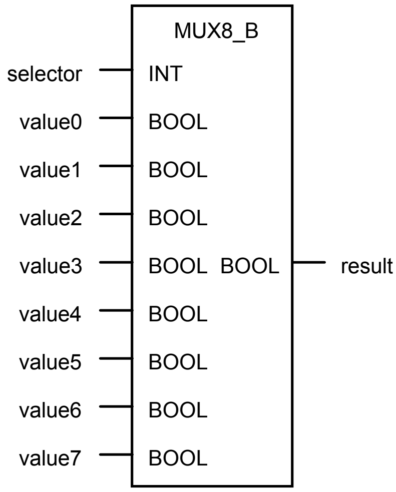
Call parameters:
selector (INT)
value0 (REAL)
value1 (REAL)
... ...
value7 (REAL)

Return parameter: result (REAL)

Prototype: result := mux8_r (select, val0, val1, val2, val3, val4, val5, val6, val7);

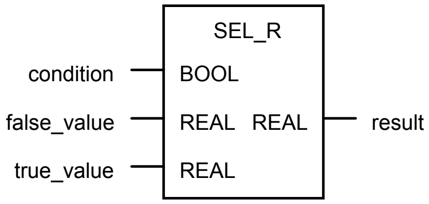
Remarks: This is the "real" equivalent of the standard MUX8 function. It does not support an extensible number of inputs.

MUX8_B



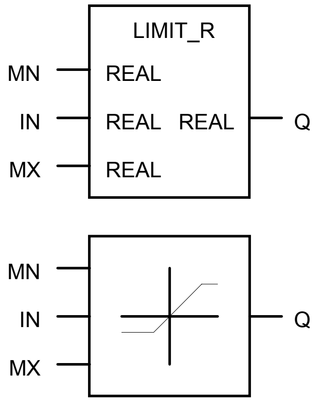
Short description:	Select one of eight boolean values
Description:	If selector is : 0 then result = value0 1 value1 ... ... 7 value7 For any other selector value, result is set to FALSE.
Call parameters:	selector (INT) value0 (BOOL) value1 (BOOL) ... ... value7 (BOOL)
Return parameter:	result (BOOL)
Prototype:	result := mux8_b (select, val0, val1, val2, val3, val4, val5, val6, val7);
Remarks:	This is the "boolean" equivalent of the standard MUX8 function. It does not support an extensible number of inputs.

SEL_R



Short description:	Select one of two real values
Description:	If condition = FALSE, then the result is equal to the false_value. If condition = TRUE, then the result is equal to the true_value.
Call parameters:	condition (BOOL) false_value (REAL) true_value (REAL)
Return parameter:	result (REAL)
Prototype:	result := sel_r (selector, value1, value2);
Remarks:	This is the "real" equivalent of the standard SEL function. It does not support an extensible number of inputs.

LIMIT_R



Short description:	Bounds a real value between a minimum and a maximum		
Description:	-		
Call parameters:	MN:	minimum value	(REAL)
	IN:	input value	(REAL)
	MX:	maximum value	(REAL)
Return parameter:	Q:	bound value	(REAL)
Prototype:	bound_value := limit (mini, value, maxi);		
Remarks:	This is the "real" equivalent of the standard LIMIT function.		

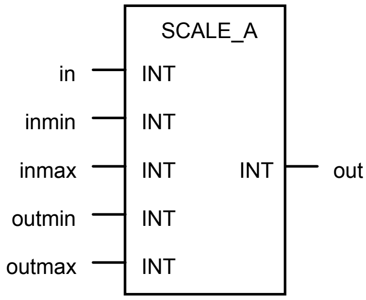
2.9 Data Conversion Functions

Standard Data Conversion Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

BOO	Convert to BOOLEAN (any input type)
ANA	Convert to ANALOG (INTEGER) (any input type)
REAL	Convert to REAL (any input type)
TMR	Convert to TIMER (any input type)
MSG	Convert to MESSAGE (any input type)
ASCII	Character to ASCII code
CHAR	ASCII code to character

SCALE_A



Short description: Scaling of analog value

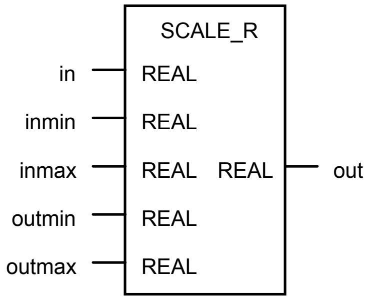
Description: This block scales the input value from range INMIN .. INMAX to the range OUTMIN .. OUTMAX using the following formula:

$$OUT = OUTMIN + \frac{(IN - INMIN) * (OUTMAX - OUTMIN)}{INMAX - INMIN}$$

If INMIN >= INMAX or OUTMIN >= OUTMAX the output is set to OUTMIN.
The IN - INMIN and OUTMAX - OUTMIN expressions MUST fall within the range -32768 to 32767. If they get out of that range, the OUT output is set to OUTMIN.

Call parameters:	IN:	input value	(INT)
	INMIN:	minimum input value	(INT)
	INMAX:	maximum input value	(INT)
	OUTMIN:	output value if IN=INMIN	(INT)
	OUTMAX:	output value if IN=INMAX	(INT)
Return parameter:	OUT :	output value	(INT)
Prototype:	scaled_value := SCALE_A (inp, imin, imax, omin, omax);		

SCALE_R



Short description:	Scaling of real value	

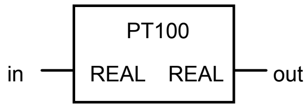
Short description: Scaling of real value

Description: This block scales the input value from range INMIN .. INMAX to the range OUTMIN .. OUTMAX using the following formula:

$$OUT = OUTMIN + \frac{IN - INMIN}{INMAX - INMIN} * (OUTMAX - OUTMIN)$$

Call parameters:	IN:	input value	(REAL)
	INMIN:	minimum input value	(REAL)
	INMAX:	maximum input value	(REAL)
	OUTMIN:	output value if IN=INMIN	(REAL)
	OUTMAX:	output value if IN=INMAX	(REAL)
Return parameter:	OUT :	output value	(REAL)
Prototype:	scaled_value := SCALE_R (inp, imin, imax, omin, omax);		

PT100



Short description: Converts PT100 resistance value to temperature value

Description: The temperature for input value >= 100 ohms is calculated exactly from formula:

Rt = 100 * (1 + A*t + B*t^2)

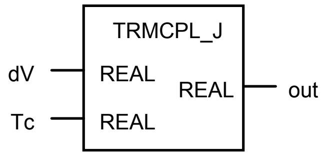
The temperature for input value < 100 ohms is approximately calculated from the formula:

Rt = 100 * (1 + A*t + B*t^2 - 100*C'*t^3)

where A = 3.90802E-3, B = -5.802E-7 and C' = -1.216532358E-11 (C' is the 'corrected' value of C = -4.2735E-12).
Compared with correct formula Rt = 100 * (1 + A*t + B*t^2 + C*(t-100)*t^3), this formula gives error of maximum +- 0.0524141 Ohm in temperature range of 0 to -200 degrees Celsius, which produces error of maximum -0.126668 and +0.122284 degrees Celsius in resistance range 100 to 18.49316 Ohm.

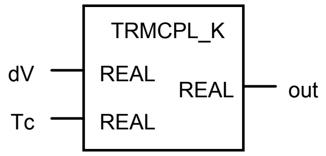
Call parameters:	IN:	input value in ohms	(REAL)
Return parameter:	OUT:	output value in degrees Celsius	(REAL)
Prototype:	temp := PT100 (res);		

TRMCPL_J



Short description:	Thermocouple linearization/compensation for J type		
Description:	For conversion from millivolts to degrees Celsius, the standard J-type thermocouple conversion table is used with supporting points at every 10 degrees Celsius. Between supporting points, linear interpolation is used. We could not estimate maximum errors that result from this 10-degree spacing of supporting points since we had neither a table with more dense spacing nor a polynomial describing the voltage-to-temperature mapping.		
Call parameters:	dV:	Voltage diff. between thermocouple junctions (mV)	(REAL)
	Tc:	Thermocouple cold junction temperature (degrees C)	(REAL)
Return parameter:	out:	Thermocouple hot junction temperature (degrees C)	(REAL)
Prototype:	temp := TRMCPL_J (delta_v, tcold);		

TRMCPL_K



Short description:	Thermocouple linearization/compensation for K type		
Description:	For conversion from millivolts to degrees Celsius, the standard K-type thermocouple conversion table is used with supporting points at every 10 degrees Celsius. Between supporting points, linear interpolation is used. We could not estimate maximum errors that result from this 10-degree spacing of supporting points since we had neither a table with more dense spacing nor a polynomial describing the voltage-to-temperature mapping.		
Call parameters:	dV:	Voltage diff. between thermocouple junctions (mV)	(REAL)
	Tc:	Thermocouple cold junction temperature (degrees C)	(REAL)
Return parameter:	out:	Thermocouple hot junction temperature (degrees C)	(REAL)
Prototype:	temp := TRMCPL_K (delta_v, tcold);		

2.10 String Management Functions

Standard String Management Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

DELETE	Delete substring
FIND	Find substring
REPLACE	Replace substring
MLEN	String length
INSERT	Insert string
LEFT	Extract left substring
MID	Extract middle substring
RIGHT	Extract right substring
CAT	String Concatenation
DAY_TIME	Time of Day

Currently no functions written by EXOR have been added to this group.

2.11 Array Manipulation Functions

Standard Array Manipulation Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these functions, containing the function name and short description:

ARCREATE	Create INTEGER array
ARREAD	Read INTEGER array element
ARWRITE	Write INTEGER array element

Currently no functions written by EXOR have been added to this group.

2.12 System Access Functions

Standard System Access Functions delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

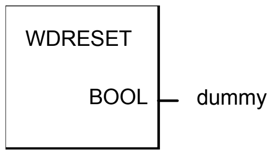
For quick reference, here is just a brief listing of these functions, containing the function name and short description:

SYSTEM	System access
OPERATE	Operate I/O Channel

Currently no functions written by EXOR have been added to this group.

2.13 Hardware Specific Functions

WDRESET



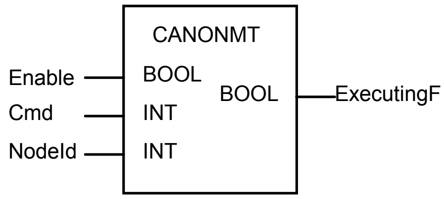
Short description: Reset the WatchDog timer.

Description: The watch dog timer will reset the processor if a PLC cycle will exeed 1.6sec duration. Calling WDRESET inside the program, will restart the timer. ATTENTION, the use of WDRESET inside program loops can be dangerous.

Call parameters:
Return parameter: dummy

Prototype: dummy := WDRESET ();

CANONMT



Short description:	Send NMT command to a CANopen node	
Description:	CANopen nodes can be controlled by a master using the NMT protocol. The master can send NMT commands to cause a change of state in the remote node.	
Call parameters:	Enable: enable the function	(BOOL)
	Cmd: NMT command, can assume the following values:	(INT)
	1 = START node	
	2 = STOP node	
	128 = enter PRE-OPERATIONAL mode	
	129 = RESET node	
	130 = RESET COMMUNICATION	
	NodeID: node number from 1 to 127	(INT)
	0 will send command to **ALL** nodes	
Return parameter:	ExecutingF: TRUE while executing	(BOOL)

Prototype: Exec := CANONMT (TRUE, command, node);

3. Function Blocks

Function blocks can have **more than one output** and can contain **internal memory** that lets certain data **be preseved from one execution of the block to another**. Therefore, a function block may return different values in two invocations with the same input parameters.

Each function block belongs to one of the following classes:

1. Boolean data manipulation FBs
2. Counting FBs
3. Timer FBs
4. Analog (integer) data manipulation FBs
5. Real data manipulation FBs
6. Signal generation FBs
7. Variable interface FBs
8. Hardware Specific FBs

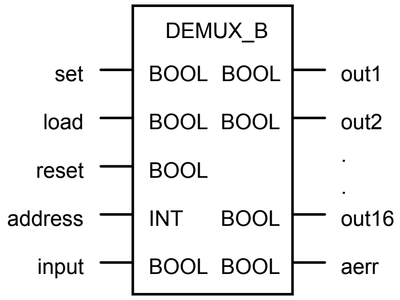
3.1 Boolean Data Manipulation FBs

Standard Boolean Data Manipulation Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description::

SR	Set dominant bistable
RS	Reset dominant bistable
R_TRIG	Rising edge detection
F_TRIG	Falling edge detection
SEMA	Semaphore

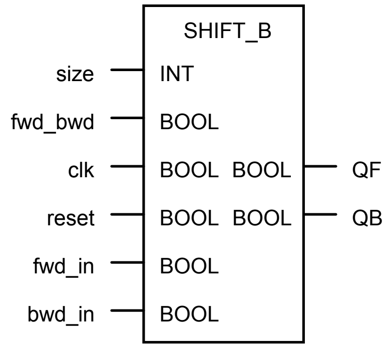
DEMUX_B

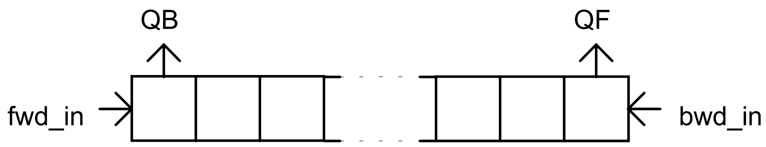


Short description:	Boolean demultiplexer with memory		
Description:	RESET overrides SET and LOAD inputs. SET overrides LOAD input. If ADDRESS is 0, all outputs are set to 0, just as if RESET input was active. See also DEMUX_R, DEMUX_A and DEMUX_T blocks.		
Call parameters:	set	if TRUE, new input value is loaded in each cycle	(BOOL)
	load	new input value is loaded on rising edge	(BOOL)
	reset	if TRUE, all outputs are set to 0	(BOOL)
	address	address of output (range 1 to 16)	(INT)
	input	input value to be demultiplexed	(BOOL)
Return params:	out1...out16	outputs	(BOOL)
	aerr	address error: set if address <0 or >16	(BOOL)

Prototype:	DEMUX_B (fs, fl, fr, addr, in);
	o1 := DEMUX_B.out1;
	err := DEMUX_B.aerr;		

SHIFT_B





Short description: Bidirectional boolean shift register of programmable length

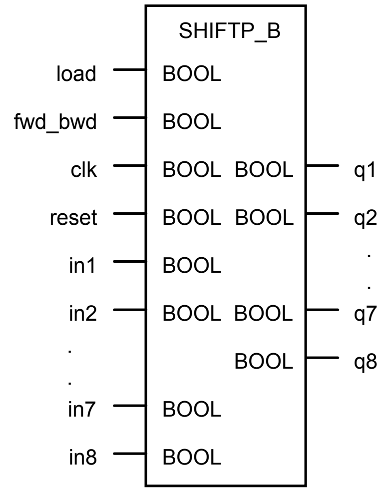
Description: At each end of the shift register, there is one input and one output. When a forward shift is executed, the value applied to the fwd_in input appears immediately at the QB output. Likewise, when a backward shift is executed, the value applied to the bwd_in input appears immediately at the QF output.
Initially (after power-up) and during reset, the whole register contains only zeros.
If a number less than 2 is applied to the size input, the shift register will have the length of 2. If a number greater than 256 is applied to the size input, the shift register will have the length of 256. The length of the register cannot be changed dynamically; value applied to the size input is read only in first cycle after power-up or reset.
See also SHIFT_R, SHIFT_A and SHIFT_T function blocks.

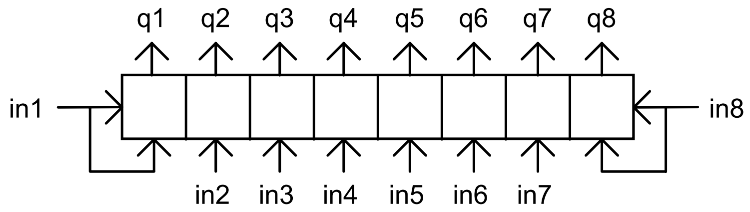
Call parameters:	size:	register length (range 2...256)	(INT)
	fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
	clk:	shifts one place on rising edge	(BOOL)
	reset:	when TRUE, clears register to 0	(BOOL)
	fwd_in:	forward shift data input	(BOOL)
	bwd_in:	backward shift data input	(BOOL)
Return params:	QF:	forward shift data output	(BOOL)
	QB:	backward shift data output	(BOOL)

Prototype:

```
SHIFT_B (100, TRUE, FALSE, FALSE, TRUE, TRUE);
outfwd := SHIFT_B.QF;
outbwd := SHIFT_B.QB;
```

SHIFTP_B





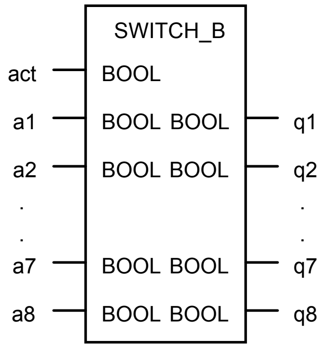
Short description:	Bidirectional boolean shift register with 8 parallel inputs and outputs
Description:	Except that it is of fixed length and has parallel inputs and outputs, the functioning of this block is similar to that of SHIFT_B block. Initially (after power-up) and during reset, the whole register contains only zeros. Inputs in2...in7 are parallel inputs only, while inputs in1 and in8 are both parallel and serial inputs. See also SHIFTP_R, SHIFTP_A and SHIFTP_T function blocks.

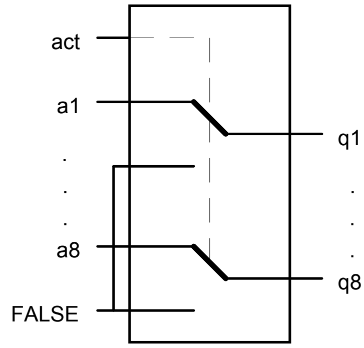
Call parameters:	load:	on rising edge, register is loaded from parallel inputs	(BOOL)
	fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
	clk:	shifts one place on rising edge	(BOOL)
	reset:	when TRUE, clears register to 0	(BOOL)
	in1:	parallel input 1 and forward shift data input	(BOOL)
	in2:	parallel input 2	(BOOL)
	...		
	in7:	parallel input 7	(BOOL)
	in8:	parallel input 8 and backward shift data input	(BOOL)
Return params:	q1:	output 1	(BOOL)
	...		
	q8:	output 8	(BOOL)

Prototype:

```
SHIFTP_B (FALSE, TRUE, TRUE, FALSE, TRUE, FALSE, ... TRUE, TRUE);
o1 := SHIFTP_B.q1;
...
o8 := SHIFTP_B.q8;
```

SWITCH_B



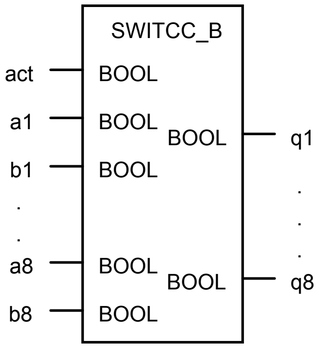


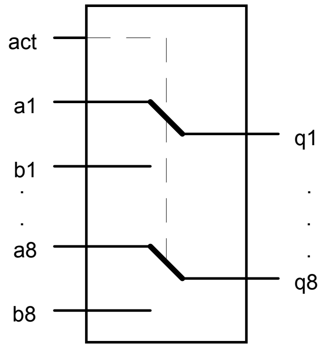
Short description:	8 single switches for analog (integer) data		
Description:	-		
Call parameters:	act:	TRUE: inputs connected to outputs FALSE: FALSE output on all outputs	(BOOL)
	a1:	input to switch 1	(BOOL)
	...		
	a8:	input to switch 8	(BOOL)
Return params:	q1:	output of switch 1	(BOOL)
	...		
	q8:	output of switch 8	(BOOL)

Prototype:	SWITCH_B (TRUE, TRUE, FALSE, ... TRUE, TRUE);
	out1 := SWITCH_B.q1;
	...
	out8 := SWITCH_B.q8;		

Remarks:	a) See also SWITCH_A, SWITCH_R and SWITCH_T function blocks.		

SWITCC_B





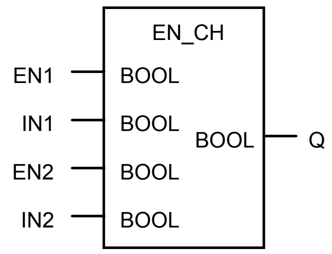
Short description:	8 changeover switches for boolean data		
Description:	-		
Call parameters:	act:	TRUE: A inputs connected to outputs FALSE: B inputs connected to outputs	(BOOL)
	a1:	switch 1, input A	(BOOL)
	b1:	switch 1, input B	(BOOL)
	...		
	a8:	switch 8, input A	(BOOL)
	b8:	switch 8, input B	(BOOL)
Return params:	q1:	output of switch 1	(BOOL)
	...		
	q8:	output of switch 8	(BOOL)

Prototype:

```
SWITCC_B (TRUE, FALSE, TRUE, FALSE, FALSE, ...  FALSE, FALSE);
out1 := SWITCC_B.q1;
...
out8 := SWITCC_B.q8;
```

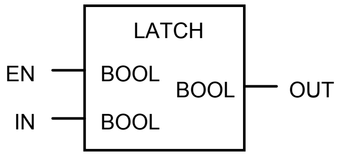
Remarks: a) See also SWITCH_R, SWITCH_A and SWITCH_T function blocks.

EN_CH



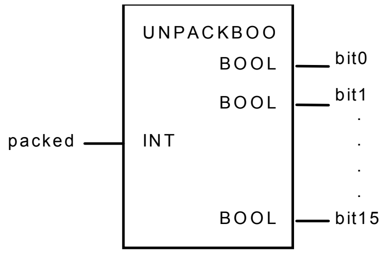
Short description:	Set output to last changed input		
Description:	Whenever any of IN1, IN2 inputs is changed, with its corresponding enable input (EN1, EN2) set to TRUE, output is set to the new (changed) state of that input. If both inputs are changed at the same time (in the same PLC cycle) and both are enabled, the new state of the input IN1 will be output. State changes on a disabled input (ENx = FALSE) cannot change the output. This block is used where one boolean value (Q) should be changed from two or more sources. If more than two sources exist, blocks of this type can be cascaded.		
Call parameters:	EN1:	enable input1	(BOOL)
	IN1:	input1	(BOOL)
	EN2:	enable input2	(BOOL)
	IN2:	input2	(BOOL)
Return params:	Q:	output	(BOOL)
Prototype:	EN_CH (en1, in1, en2, in2); out := EN_CH.Q;		

LATCH



Short description:	Binary latch		
Description:	If enable input EN is TRUE, output follows input IN, otherwise output remains unchanged. If EN is FALSE at power-up, the initial value of OUT will be FALSE.		
Call parameters:	EN:	enable input	(BOOL)
	IN:	input	(BOOL)
Return parameter:	OUT:	output	(BOOL)
Prototype:	LATCH (en, in); out := LATCH.OUT;		

UNPACKBOO



Short description:	Unpack a word into bits	
Description:	Unpacking a word in bits is sometimes needed when managing data coming from communication with other devices or I/O equipments.	
Call parameters:	word	(INT)
Return parameter:	bit0	(BOOL)
	……………	
	bit15	(BOOL)
Prototype:	unpackboo (word); b0 := unpackboo.bit0;	

3.2 Counting FBs

Standard Counting Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description::

CTU	Up counter
CTD	Down counter
CTUD	Up-down counter

Currently no functions written by EXOR have been added to this group.

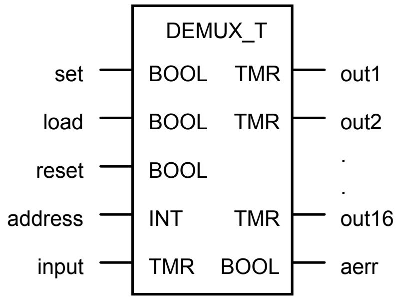
3.3 Timer FBs

Standard Timer Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description:

TON	On-delay timing
TOFF	Off-delay timing
TP	Pulse timing

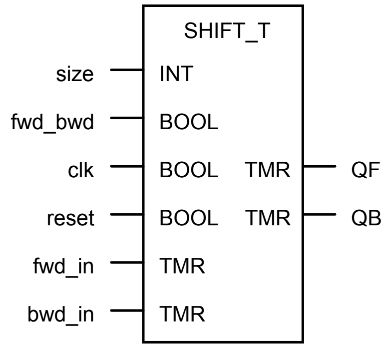
DEMUX_T

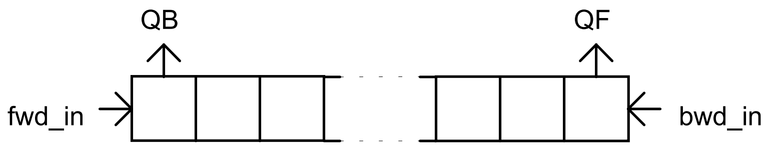


Short description:	Timer demultiplexer with memory		
Description:	RESET overrides SET and LOAD inputs. SET overrides LOAD input. If ADDRESS is 0, all outputs are set to 0, just as if RESET input was active. See also DEMUX_A, DEMUX_B and DEMUX_R blocks.		
Call parameters:	set:	if TRUE, new input value is loaded in each cycle	(BOOL)
	Load:	new input value is loaded on rising edge	(BOOL)
	seset:	if TRUE, all outputs are set to 0	(BOOL)
	address:	address of output (range 1 to 16)	(INT)
	input:	input value to be demultiplexed	(TMR)
Return params:	out1...out16:	outputs	(TMR)
	aerr	address error: set if address <0 or >16	(BOOL)

Prototype:	DEMUX_T (fs, fl, fr, addr, in);
	o1 := DEMUX_T.out1;
	err := DEMUX_T.aerr;		

SHIFT_T





Short description: Bidirectional timer shift register of programmable length

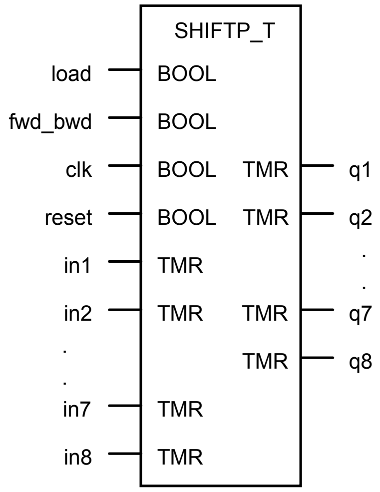
Description: At each end of the shift register, there is one input and one output. When a forward shift is executed, the value applied to the fwd_in input appears immediately at the QB output. Likewise, when a backward shift is executed, the value applied to the bwd_in input appears immediately at the QF output. Initially (after power-up) and during reset, the whole register contains only zeros.
If a number less than 2 is applied to the size input, the shift register will have the length of 2. If a number greater than 256 is applied to the size input, the shift register will have the length of 256. The length of the register cannot be changed dynamically; value applied to the size input is read only in first cycle after power-up or reset.
See also SHIFT_A, SHIFT_B and SHIFT_R function blocks.

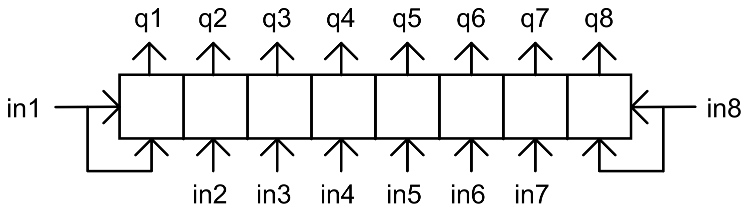
Call parameters:	size:	register length (range 2...256)	(INT)
	fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
	clk:	shifts one place on rising edge	(BOOL)
	reset:	when TRUE, clears register to 0	(BOOL)
	fwd_in:	forward shift data input	(TMR)
	bwd_in:	backward shift data input	(TMR)
Return params:	QF:	forward shift data output	(TMR)
	QB:	backward shift data output	(TMR)

Prototype:

```
SHIFT_T (100, TRUE, FALSE, FALSE, 4m30s, 0s50);
outfwd := SHIFT_T.QF;
outbwd := SHIFT_T.QB;
```

SHIFTP_T





Short description:	Bidirectional timer shift register with 8 parallel inputs and outputs

Description:

Except that it is of fixed length and has parallel inputs and outputs, the functioning of this block is similar to that of SHIFT_T block.
Initially (after power-up) and during reset, the whole register contains only zeros.
Inputs in2...in7 are parallel inputs only, while inputs in1 and in8 are both parallel and serial inputs.
See also SHIFTP_A, SHIFTP_B and SHIFTP_R function blocks.

Call parameters:

load:	on rising edge, register is loaded from parallel inputs	(BOOL)
fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
clk:	shifts one place on rising edge	(BOOL)
reset:	when TRUE, clears register to 0	(BOOL)
in1:	parallel input 1 and forward shift data input	(TMR)
in2:	parallel input 2	(TMR)
...		
in7:	parallel input 7	(TMR)
in8:	parallel input 8 and backward shift data input	(TMR)

Return params:

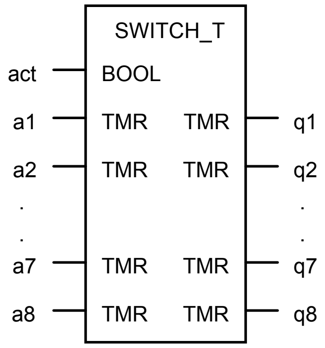
q1:	output 1	(TMR)
...		
q8:	output 8	(TMR)

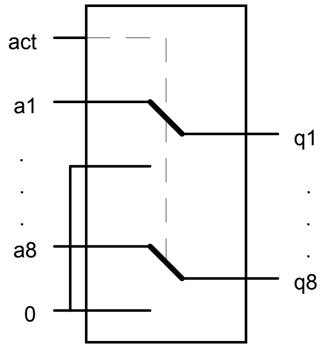
Prototype:

SHIFTP_T (FALSE, TRUE, TRUE, FALSE, 2s, 15h30m, ... 5m20s, 120ms);
o1 := SHIFTP_T.q1;

```
...
o8 := SHIFTP_T.q8;
```

SWITCH_T



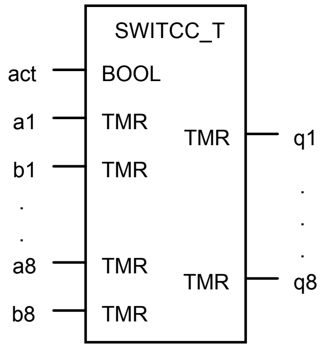


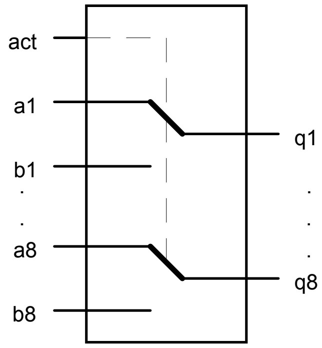
Short description:	8 single switches for timer data		
Description:	-		
Call parameters:	act:	TRUE: inputs connected to outputs FALSE: zero output on all outputs	(BOOL)
	a1:	input to switch 1	(TMR)
	...		
	a8:	input to switch 8	(TMR)
Return params:	q1:	output of switch 1	(TMR)
	...		
	q8:	output of switch 8	(TMR)

Prototype:	SWITCH_T (TRUE, 1s, 22m, 17h, 4m30s, 5s12, 90ms, 1s20, 1h20m);
	out1 := SWITCH_T.q1;
	...
	out8 := SWITCH_T.q8;		

Remarks:	a) See also SWITCH_A, SWITCH_B and SWITCH_R function blocks.		

SWITCC_T





Short description: 8 changeover switches for timer data

Description: -

Call parameters:	act:	TRUE: A inputs connected to outputs FALSE: B inputs connected to outputs	(BOOL)
	a1:	switch 1, input A	(TMR)
	b1:	switch 1, input B	(TMR)
	...		
	a8:	switch 8, input A	(TMR)
	b8:	switch 8, input B	(TMR)
Return params:	q1:	output of switch 1	(TMR)
	...		
	q8:	output of switch 8	(TMR)

Prototype:

```
SWITCC_T (TRUE, 1s, 220ms, 17h, 4m30s, ...  100ms, 0s20);
out1 := SWITCC_T.q1;
...
out8 := SWITCC_T.q8;
```

Remarks: a) See also SWITCH_A, SWITCH_B and SWITCH_R function blocks.

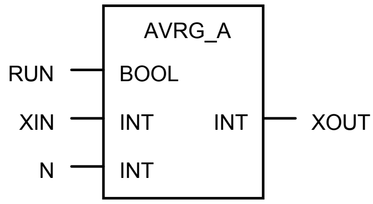
3.4 Analog (Integer) Data Manipulation FBs

Standard Analog (INTEGER) Data Manipulation Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description::

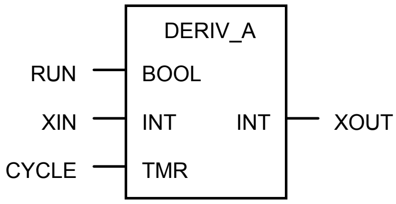
CMP	Full comparison
STACKINT	Stack of INTEGERs

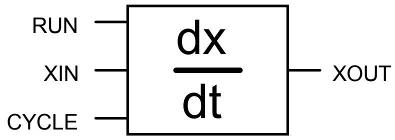
AVRG_A



Short description:	Running average over N integer (analog) samples		
Description:	This is the "analog" equivalent of the standard AVERAGE function block. Except for changed input and output types, its functioning is exactly the same as the original block. For further details, please refer to the description of the original block in the ISaGRAF User's Manual.		
Call parameters:	RUN:	enable command, reset average if FALSE	(BOOL)
	XIN:	input sample	(INT)
	N:	number of samples for averaging	(INT)
Return params:	XOUT:	running average	(INT)
Prototype:	AVRG_A (average_enable, sample_value, 4); clean_value := AVRG_A.XOUT;		

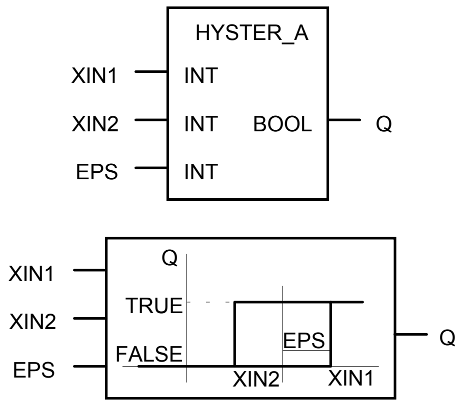
DERIV_A





Short description:	Differentiation with respect to time
Description:	This is the "analog" equivalent of the standard DERIVATE function block. Except for changed input and output types, its functioning is exactly the same as the original block. Derivation is output in units of 1/10ms, i.e. the output numerical quantity represents the change of the input signal in the interval of 10ms. The value applied to the CYCLE input does not influence the output value, but is only used to execute calculation and output updating not more often than it states. For further details, please refer to the description of the original block in the ISaGRAF User's Manual.
Call parameters:	RUN: enable command, reset output if FALSE (BOOL) XIN: sample of the function to be differentiated (INT) CYCLE: sampling period (TMR)
Return params:	XOUT: output = differentiated input (INT)
Prototype:	DERIV_A (TRUE, temp_5, period_5); speed_5 := DERIV_A.XOUT;
Example:	If the rate of change of input is 200 units per second (200/s), the value that will be output is 2 (200/s = 200/(100*10ms) = (200/100)*(1/10ms) = 2*(1/10ms)). ATTENTION! For an input with rate of change less than 100 units per second (100/s), output will be 0 (99/s = 99/(100*10ms) = (99/100)*(1/10ms) = INTEGER ARITHMETIC!! = 0*(1/10ms) = 0).

HYSTER_A

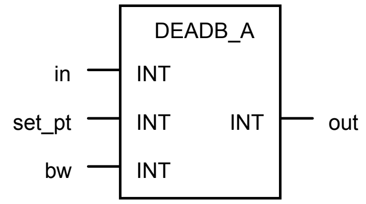


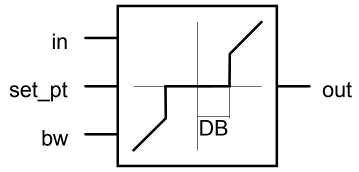
Short description: Boolean hysteresis on the difference of analog inputs

Description: This is the "analog" equivalent of the standard HYSTER function block. Except for changed input and output types, its functioning is exactly the same as the original block. For details, please refer to the above drawing and to the description of the original block in the ISaGRAF User's Manual.

Call parameters:	XIN1:	input signal	(INT)
	XIN2:	hysteresis centerpoint	(INT)
	EPS:	hysteresis halfwidth	(INT)
Return params:	Q:	output	(BOOL)
Prototype:	HYSTER_A (pressure, press_limit, 21);		
	too_high := HYSTER_A.Q;		

DEADB_A





Short description: Deadband for analog (integer) input

Description: The deadband of total width 2∗bw is positioned symetrically around the center point. If the value of "in" input falls within the deadband, "set_pt" value is output, otherwise "in" value is output.
See also DEADB_R and DEADBH_A function blocks.

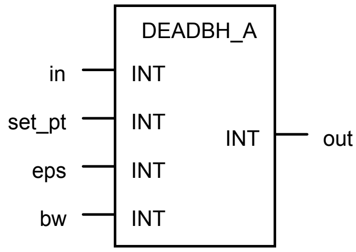
Call parameters:

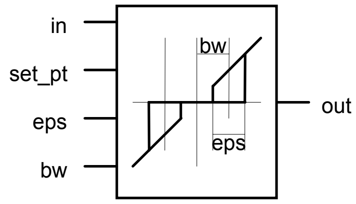
	in:	input signal	(INT)
	set_pt:	center point	(INT)
	bw:	halfwidth of deadband	(INT)
Return params:	out:	output signal	(INT)

Prototype:

```
DEADB_A (input, 10, 2);
out := DEADB_A.out;
```

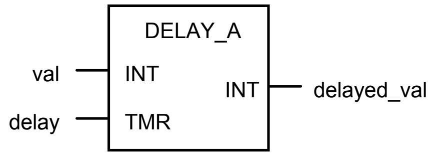
DEADBH_A





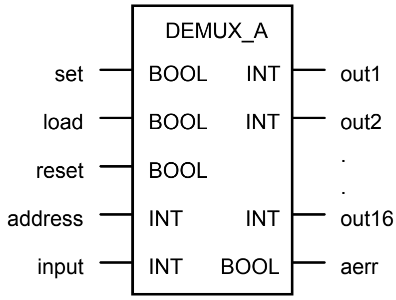
Short description:	Deadband with hysteresis for analog (integer) input		
Description:	To reduce the frequency of switching operations, it is usual to provide final control elements with a hysteresis or differential gap. This hysteresis prevents minor deviations of input signal from the center point from being forwarded to the output. If the system deviation exceeds the switching differential, the input value is passed unchanged to the output. The deadband of width 2∗bw is positioned symetrically around the center point and flanked on both sides by hysteresis regions of width eps. Width of deadband bw is measured from the origin of the coordinate system to the center of any hysteresis region.		
Call parameters:	in:	input signal	(INT)
	set_pt:	center point	(INT)
	eps:	width of hysteresis	(INT)
	bw:	width of deadband	(INT)
Return params:	out:	output signal	(INT)
Prototype:	DEADBH_A (input, cpt, hyst, bw); out := DEADBH_A.out;		

DELAY_A



Short description:	Time delay of analog value		
Description:	If delay is smaller than the duration of one program execution cycle, DELAY_A block just passes the unmodified input value to the output. The maximum delay value is limited only by ISaGRAF limit on variables of TMR type, i.e. it is 24 hours. If the specified delay is shorter than 100 cycles, val measured in each cycle is put into FIFO and is output after the delay elapses. However, if this is not the case, max. delayed_val update period is delay/100, otherwise the FIFO through which the input values pass before being output would be too long. Inside one update period, the values of val input in all cycles belonging to it are averaged to produce the value that is eventually put into FIFO and output later. Averaging is correct for up to 10 cycles per update period, but for longer update periods, certain values are weighted with varying weights in order to keep the needed memory space limited.		
Call parameters:	val:	value to delay	(INT)
	delay:	delay time	(TMR)
Return params:	delayed_val:	delayed value	(INT)
Prototype:	DELAY_A (value, deltat); d_val := DELAY_A.delayed_val		

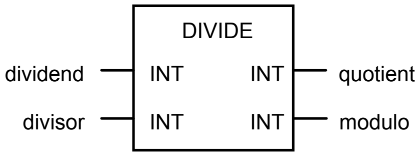
DEMUX_A



Short description:	Integer demultiplexer with memory		
Description:	RESET overrides SET and LOAD inputs. SET overrides LOAD input. If ADDRESS is 0, all outputs are set to 0, just as if RESET input was active. See also DEMUX_R, DEMUX_B and DEMUX_T blocks.		
Call parameters:	set	if TRUE, new input value is loaded in each cycle	(BOOL)
	load	new input value is loaded on rising edge	(BOOL)
	reset	if TRUE, all outputs are set to 0	(BOOL)
	address	address of output (range 1 to 16)	(INT)
	input	input value to be demultiplexed	(INT)
Return params:	out1...out16	outputs	(INT)
	aerr	address error: set if address <0 or >16	(BOOL)

Prototype:	DEMUX_A (fs, fl, fr, addr, in);
	o1 := DEMUX_A.out1;
	err := DEMUX_A.aerr;		

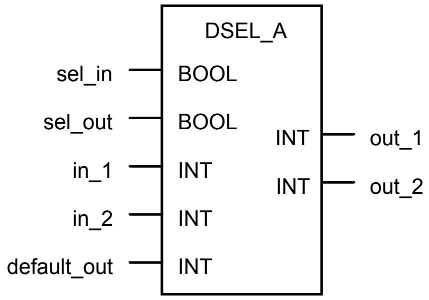
DIVIDE_A

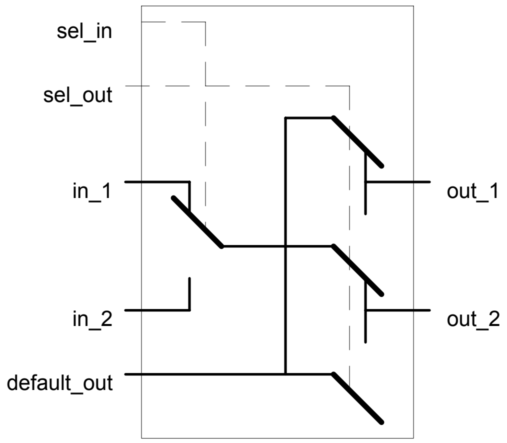


Short description:	Full integer divider (quotient and remainder)		
Description:	Returns -1 on both outputs if divisor is less than or equal to 0.		
Call parameters:	dividend:	number to be divided	(INT)
	divisor:	number to divide with	(INT)
Return params:	quotient:	result of division	(INT)
	modulo:	remainder value	(INT)

Prototype:	DIVIDE_A (dend, sor);
	res := DIVIDE_A.quotient;
	rem := DIVIDE_A.modulo;		

DSEL_A

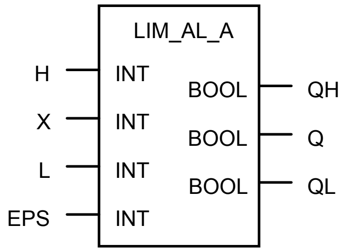




Short description:	Double independently operated analog switch with two inputs	
Description:	Please refer to the above relay diagram which should be clear enough.	
Call parameters:	sel_in: Selects in_1 (FALSE) or in_2 (TRUE)	(BOOL)
	sel_out: Selects out_1 (FALSE) or out_2 (TRUE)	(BOOL)
	in_1: Analog input 1	(INT)
	in_2: Analog input 2	(INT)
	default_out: Value to be placed at non-selected output	(INT)
Return params:	out_1: Analog output 1	(INT)
	out_2: Analog output 2	(INT)

Prototype:	DSEL_A (selin, selout, inval1, inval2, defout);
	outval1 := DSEL_A.out_1;
	outval2 := DSEL_A.out_2;	

LIM_AL_A



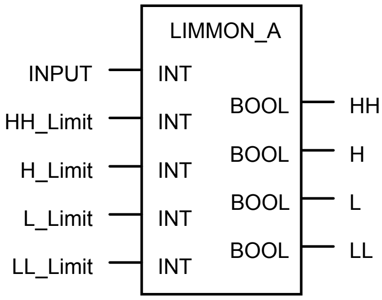
Short description:	Alarm detection for an analog (integer) variable		
Description:	-		
Call parameters:	H:	High limit	(INT)
	X:	Variable value	(INT)
	L:	Lower limit	(INT)
	EPS:	Hysteresis around limits	(INT)
Return params:	QH:	High alarm	(BOOL)
	Q:	Any alarm (QH or QL)	(BOOL)
	QL:	Low alarm	(BOOL)

Prototype:

```
LIM_AL_A (215.0, temp_5, 120.5, 30);
hot   := LIM_ALRM.QH;
alarm := LIM_ALRM.Q;
cold  := LIM_ALRM.QL;
```

Remarks: a) This is the "analog" equivalent of the standard LIM_ALRM function block.

LIMMON_A



Short description: Extended limit monitor of integer value

Description: This function block implements the standard industrial 4-level limit monitor, supporting high alarm (HH), high prealarm (H), low prealarm (L) and low alarm (LL) levels. The 4 outputs indicate in which of the 5 regions the input value currently is:

above HH	HH output TRUE
between HH and H	H output TRUE
between L and H (inside "normal" band)	all outputs FALSE
between LL and L	L output TRUE
below LL	LL output TRUE

At most one of the outputs will be TRUE at any time, except when the limit values are not in increasing order, i.e. when the inequality
LL_Limit <= L_Limit < H_Limit <= HH_Limit
is not satisfied, in which case all 4 outputs will be set to TRUE.

Call parameters:

INPUT	(INT)
HH_Limit	(INT)
H_Limit	(INT)
L_Limit	(INT)
LL_Limit	(INT)

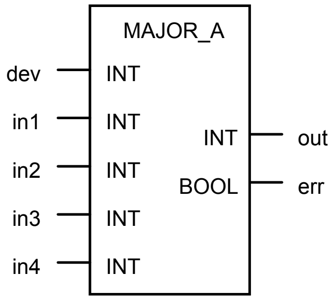
Return params:

HH	(BOOL)
H	(BOOL)
L	(BOOL)
LL	(BOOL)

Prototype:

```
LIMMON_A (in, hh_l, h_l, l_l, ll_l);
hh_alarm := LIMMON_A.HH;
h_alarm := LIMMON_A.H;
l_alarm := LIMMON_A.L;
ll_alarm := LIMMON_A.LL;
```

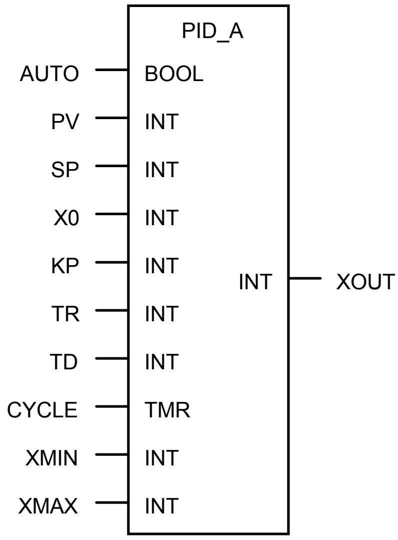
MAJOR_A



Short description:	Majority selector for integer inputs
Description:	The majority selector calculates the mean value of all inputs. If exactly one input differs from the calculated mean value by more than dev, mean value is calculated once more, but that input is excluded from the calculation. If more than one input deviates by more than dev, the mean value of all of the input values is calculated and the output err is set. See also MAJOR_R function block.
Call parameters:	dev: Max. permissible deviation between any input and the calculated mean value (INT) in1: Input 1 (INT) ... in4: Input 4 (INT)
Return params:	out: Mean value of inputs not deviating by more than dev from itself (INT) err: set when majority selection is impossible (BOOL)

Prototype:	MAJOR_A (deviation, i1, i2, i3, i4);
	error := MAJOR_A.err;
	mean := MAJOR_A.out;

PID_A



Short description: PID Controller with analog (integer) inputs and output

Description: This is the "analog" version of the standard PID_REX function block: all inputs and the output which are of type REAL in the original block are here of type ANALOG (INTEGER).

Call parameters:

AUTO:	Auto (TRUE)/Manual (FALSE) mode	(BOOL)
PV:	Process variable (X)	(INT)
SP:	Setpoint (W)	(INT)
X0:	Value to be output in Manual mode	(INT)
KP:	Proportional gain	(INT)
TR:	Integral time	(INT)
TD:	Derivative time	(INT)
CYCLE :	Calculation and output updating period	(TMR)
XMIN:	Min. value of output quantity (Y)	(INT)
XMAX:	Max. value of output quantity (Y)	(INT)

Return params: XOUT: Output quantity (Y) (INT)

Prototype:

```
PID_A (TRUE, temp_5, 1200, manual_temp, kp, tr, td, 0s40, 0, 10000);
heater := PID_A.XOUT;
```

Remarks: Algorithm implemented in this block is the so-called "independent" PID algorithm. Kp multiplies all three terms (proportional, integral and derivative) in the following way:

error = SP - PV
XOUT = KP * (error + (1/TR)*integral(error) + TD*derivative(error))

For this type of algorithm, optimum KP, TR, TD parameters according to the Ziegler-Nichols method are:

for P controller:	KP = 0.5 *KPosc		
for PI controller:	KP = 0.45*KPosc	TR = 0.83*Tosc	
for PID controller:	KP = 0.6 *KPosc	TR = 0.5 *Tosc	TD = 0.125*Tosc

where KPosc is that KP which causes constant-amplitude closed-loop oscillations with only P-action enabled and Tosc is the period of these oscillations.

WARNINGS FOR THE USER:

With respect to PID algorithm using real (floating-point) arithmetic, PID algorithm using integer arithmetic suffers from the following additional problems:

1. Overflow

Overflow is a major problem in integer PID algorithms using 16-bit variables for internal data storage. However, since here variables are of the type "signed long integer", which are 32-bit entities in C-compilers for 80x86 processors, the problem is much less pronounced.

Having the range of -2 147 483 648 to +2 147 483 647, with reasonable values for process value (X), set-point (W) and output value (Y), as well as proportional gain (Kp), integral (Ti) and derivative (Td) times, the probability of exceeding the "signed long" range is extremely low. For this reason, no range checking is done, which makes the algorithm faster.

2. Rounding noise

Integer arithmetic rounds off all division results. Obviously, accuracy is lost in this way. Divisions cannot be avoided, but it is important to make sure that:
a) the ratio between the integer division result and the truncated decimal part is as large as possible and that
b) rounding-off errors are not cumulative.

The problem of keeping the dividends much larger than divisors is in stand-alone PID controllers usually solved by appropriate scaling (normalization). This is easily done, since, although X, W and Y can each be expressed in its own physical units, they are usually represented by already normalized input or output signals (0-10V, 0-20mA, 4-20mA), transferred into digital domain also as normalized quantities (0-4095 for 12-bit A/D converters).

OUR CASE IS DIFFERENT. Our PID is a function block having numerical inputs and outputs for which no fixed range is defined in advance. For this reason, no reasonable normalization can be done and this step remains AT THE RESPONSIBILITY OF THE BLOCK USER.

In extreme cases, rounding errors can make an otherwise stable system become unstable. In less critical cases, permanent small oscillations of the output value (Y) in the stable state result. They can adversely affect the actuator, if not filtered out by an external dead-band block.

To avoid problems of this kind and worse, it is recommended to choose Y range to be at least 1-1000 and to scale X and W to values greater than 100 before applying them to the inputs of the analog PID block.

3. Input parameters resolution

Since input parameters of an analog PID are also integer quantities, they too are subject to loss od resolution.

For example, Kp is often in the range 1-10. If Kp were input without scaling, with Kp = 1,5 we could only choose whether to apply 1 or 2 to the Kp input, with Kp = 1 probably giving too slow rise-time and Kp = 2 too high and too broad an overshoot.

Ti and Td are subject to similar resolution-related problems.

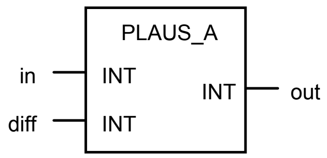
Standard "real" PID block supplied by CJ International already uses Ti and Td in 10ms units, which is equal to the basic resolution of the whole system. Therefore, 100 is applied to Ti input to indicate Ti = 1 second. This approach was not changed; on "analog" PID, the same units are used, so that no problems with Td and Ti resolution emerge.

Kp, however, was subject to input scaling: what is input is NOT the actual value od Kp, but the value Kp*100. Therefore, for Kp = 1,5, the value applied to the Kp input should be 150.

This is consistent with Ti and Td and helps improve resolution in the most useful range of Kp values.

See also PID_REX function block.

PLAUS_A



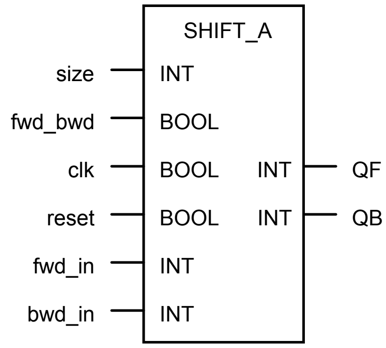
Short description:	Plausibility checking block for analog (integer) input
Description:	The block compares the difference between two succesive values sampled on the "in" input with the value on the "diff" input. If the difference of successive samples is less than "diff", the actual "in" value is forwarded to the output. If the difference exceeds "diff", the value to be output is calculated as the mean value of "in" samples in the 3 preceding cycles. In the cycle following this one, "in" is compared not to the preceding value, but with the last plausible value, i.e. one before it. If the difference in the cycle following the cycle in which the mean value was output is still above "diff", this is taken as the proof that both this and previous "in" values are plausible and the "in" value is normally forwarded to the output.

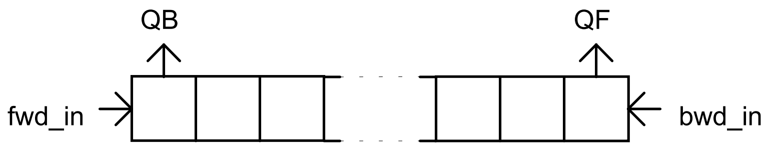
Call parameters:	in:	input	(INT)
	diff:	allowed difference	(INT)
Return params:	out:	output	(INT)

Prototype:

```
PLAUS_A (input, difference);
o := PLAUS_A.out;
```

SHIFT_A





Short description: Bidirectional analog (integer) shift register of programmable length

Description: At each end of the shift register, there is one input and one output. When a forward shift is executed, the value applied to the fwd_in input appears immediately at the QB output. Likewise, when a backward shift is executed, the value applied to the bwd_in input appears immediately at the QF output. Initially (after power-up) and during reset, the whole register contains only zeros.
If a number less than 2 is applied to the size input, the shift register will have the length of 2. If a number greater than 256 is applied to the size input, the shift register will have the length of 256. The length of the register cannot be changed dynamically; value applied to the size input is read only in first cycle after power-up or reset.
See also SHIFT_R, SHIFT_B and SHIFT_T function blocks.

Call parameters:

	size:	register length (range 2...256)	(INT)
	fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
	clk:	shifts one place on rising edge	(BOOL)
	reset:	when TRUE, clears register to 0	(BOOL)
	fwd_in:	forward shift data input	(INT)
	bwd_in:	backward shift data input	(INT)

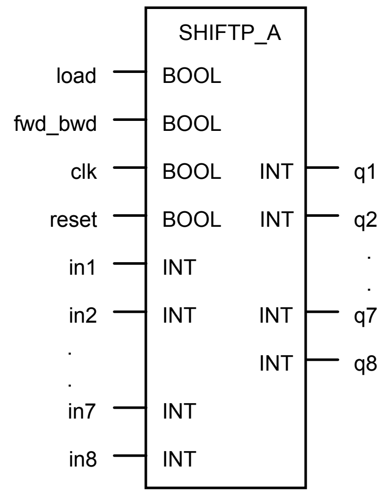
Return params:

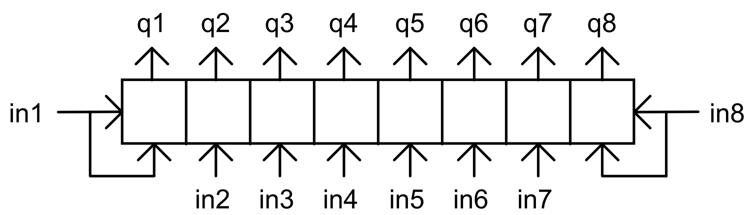
	QF:	forward shift data output	(INT)
	QB:	backward shift data output	(INT)

Prototype:

```
SHIFT_A (100, TRUE, FALSE, FALSE, 123, 47);
outfwd := SHIFT_A.QF;
outbwd := SHIFT_A.QB;
```

SHIFTP_A





Short description:	Bidirectional analog (integer) shift register with 8 parallel inputs and outputs

Description:

Except that it is of fixed length and has parallel inputs and outputs, the functioning of this block is similar to that of SHIFT_A block.
Initially (after power-up) and during reset, the whole register contains only zeros.
Inputs in2...in7 are parallel inputs only, while inputs in1 and in8 are both parallel and serial inputs.
See also SHIFTP_R, SHIFTP_B and SHIFTP_T function blocks.

Call parameters:

load:	on rising edge, register is loaded from parallel inputs	(BOOL)
fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
clk:	shifts one place on rising edge	(BOOL)
reset:	when TRUE, clears register to 0	(BOOL)
in1:	parallel input 1 and forward shift data input	(INT)
in2:	parallel input 2	(INT)
...		
in7:	parallel input 7	(INT)
in8:	parallel input 8 and backward shift data input	(INT)

Return params:

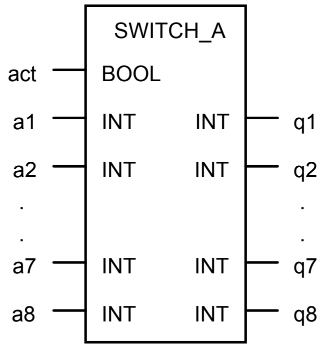
q1:	output 1	(INT)
...		
q8:	output 8	(INT)

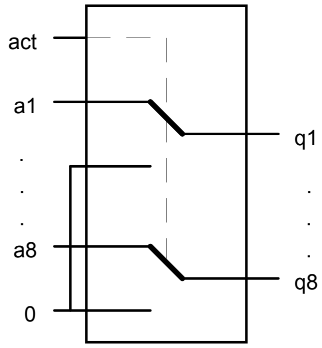
Prototype:

```
SHIFTP_A (FALSE, TRUE, TRUE, FALSE, 2, 15, -4, 0, 100, 1, 52, -12);
o1 := SHIFTP_A.q1;
...
```

```
o8 := SHIFTP_A.q8;
```

SWITCH_A



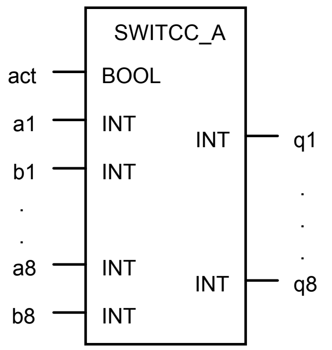


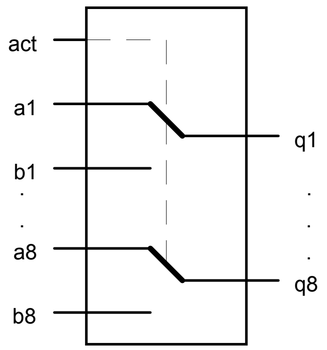
Short description:	8 single switches for analog (integer) data		
Description:	-		
Call parameters:	act:	TRUE: inputs connected to outputs FALSE: zero output on all outputs	(BOOL)
	a1:	input to switch 1	(INT)
	...		
	a8:	input to switch 8	(INT)
Return params:	q1:	output of switch 1	(INT)
	...		
	q8:	output of switch 8	(INT)

Prototype:	SWITCH_A (TRUE, 1, 22, -17, 4, 512, -93, 100, 0);
	out1 := SWITCH_A.q1;
	...
	out8 := SWITCH_A.q8;		

Remarks:	a) See also SWITCH_R, SWITCH_B and SWITCH_T function blocks.		

SWITCC_A





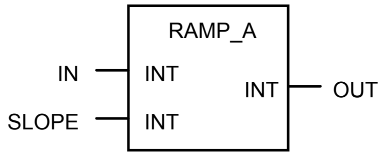
Short description:	8 changeover switches for analog (integer) data		
Description:	-		
Call parameters:	act:	TRUE: A inputs connected to outputs FALSE: B inputs connected to outputs	(BOOL)
	a1:	switch 1, input A	(INT)
	b1:	switch 1, input B	(INT)
	...		
	a8:	switch 8, input A	(INT)
	b8:	switch 8, input B	(INT)
Return params:	q1:	output of switch 1	(INT)
	...		
	q8:	output of switch 8	(INT)

Prototype:

```
SWITCC_A (TRUE, 1, 22, -17, 4, ...  100, 0);
out1 := SWITCC_A.q1;
...
out8 := SWITCC_A.q8;
```

Remarks: a) See also SWITCH_R, SWITCH_B and SWITCH_T function blocks.

RAMP_A



Short description: Ramp limiter for analog signals

Description: Output (OUT) follows the input signal (IN) as long as the absolute value of its rate of change is below the value applied to the SLOPE input. When the absolute value of rate of change of input exceeds SLOPE, the rate of change of output il limited to +SLOPE or -SLOPE until the moment when OUT again becomes equal to IN. At that moment, tracking continues.
SLOPE is expressed in units of 1/10ms, i.e. the numerical value applied to this input represents the maximum allowed change of the IN signal in the interval of 10ms. This makes the block compatible with blocks delivered by CJ International (e.g. derivator). However, care should be taken to appropriately scale the IN signal, since due to integer arithmetic, the least SLOPE supported is 100 units per second (100/s = 1/10ms).

Call parameters:	IN:	input	(INT)
	SLOPE:	allowed input change	(INT)
Return params:	OUT:	output	(INT)
Prototype:	RAMP_A (inp, slope);		
	outp := RAMP_A.out;		

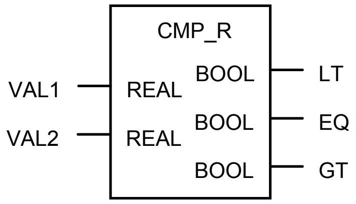
3.5 Real Data Manipulation FBs

Standard Real Data Manipulation Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description::

AVERAGE	Running average of REAL samples
HYSTER	Boolean hysteresis on the difference of REALs
LIM_ALRM	High/low limit alarm with hysteresis
INTEGRAL	Integration over time
DERIVATE	Differentiation with respect to time

CMP_R

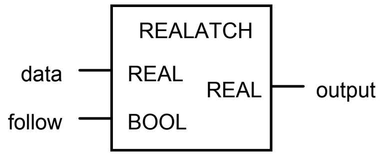


Short description:	Full comparison of two real numbers		
Description:	-		
Call parameters:	VAL1, VAL2:	numbers to be compared	(REAL)
Return params:	LT:	TRUE if VAL1 is lower than VAL2	(BOOL)
	EQ:	TRUE if VAL1 is equal to VAL2	(BOOL)
	GT:	TRUE if VAL1 is greater than VAL2	(BOOL)

Prototype:

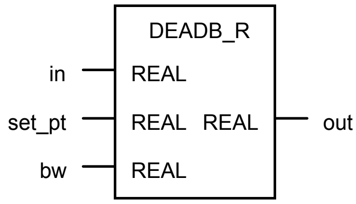
```
CMP_R (value, reference);
is_lower = CMP_R.LT;
is_equal = CMP_R.EQ;
is_greater = CMP_R.GT;
```

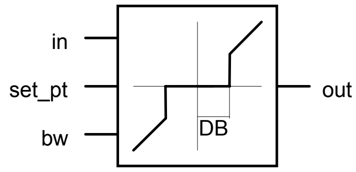
REALATCH



Short description:	Real data latch		
Description:	For follow = TRUE, output follows data. For follow = FALSE, output holds the value present at the data input at the moment of the TRUE-to-FALSE transition.		
Call parameters:	data:	real data input	(REAL)
	follow:	enable input following	(INT)
Return params:	output:		(REAL)
Prototype:	REALATCH (flow, pass); flow_max := REALATCH.OUTPUT;		

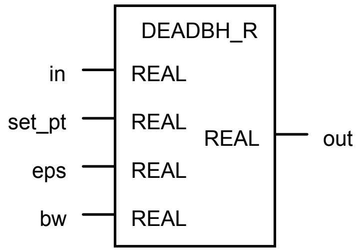
DEADB_R

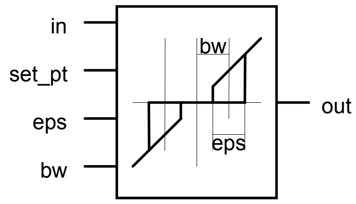




Short description:	Deadband for real input
Description:	The deadband of total width 2∗bw is positioned symetrically around the center point. If the value of "in" input falls within the deadband, "set_pt" value is output, otherwise "in" value is output. See also DEADB_A and DEADBH_R function blocks.
Call parameters:	in: input signal (REAL) set_pt: center point (REAL) bw: halfwidth of deadband (REAL)
Return params:	out: output signal (REAL)
Prototype:	DEADB_R (input, 10., 2.); out := DEADB_R.out;

DEADBH_R





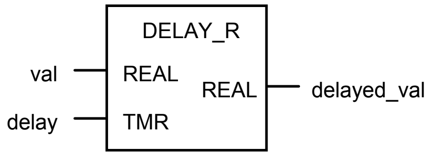
Short description:	Deadband with hysteresis for real input
Description:	To reduce the frequency of switching operations, it is usual to provide final control elements with a hysteresis or differential gap. This hysteresis prevents minor deviations of input signal from the center point from being forwarded to the output. If the system deviation exceeds the switching differential, the input value is passed unchanged to the output. The deadband of width 2*bw is positioned symetrically around the center point and flanked on both sides by hysteresis regions of width eps. Width of deadband bw is measured from the origin of the coordinate system to the center of any hysteresis region.

Call parameters:	in:	input signal	(REAL)
	set_pt:	center point	(REAL)
	eps:	width of hysteresis	(REAL)
	bw:	width of deadband	(REAL)
Return params:	out:	output signal	(REAL)

Prototype:

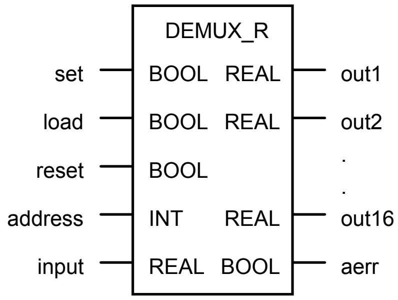
```
DEADBH_R (input, cpt, hyst, bw);
out := DEADBH_R.out;
```

DELAY_R



Short description:	Time delay of real value
Description:	If delay is smaller than the duration of one program execution cycle, DELAY_R block just passes the unmodified input value to the output. The maximum delay value is limited only by ISaGRAF limit on variables of TMR type, i.e. it is 24 hours. If the specified delay is shorter than 100 cycles, val measured in each cycle is put into FIFO and is output after the delay elapses. However, if this is not the case, max. delayed_val update period is delay/100, otherwise the FIFO through which the input values pass before being output would be too long. Inside one update period, the values of val input in all cycles belonging to it are averaged to produce the value that is eventually put into FIFO and output later. Averaging is correct for up to 10 cycles per update period, but for longer update periods, certain values are weighted with varying weights in order to keep the needed memory space limited.
Call parameters:	val: value to delay (REAL) delay: delay time (TMR)
Return params:	delayed_val: delayed value (REAL)
Prototype:	DELAY_R (value, deltat); d_val := DELAY_R.delayed_val

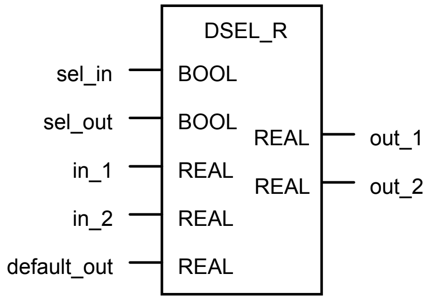
DEMUX_R

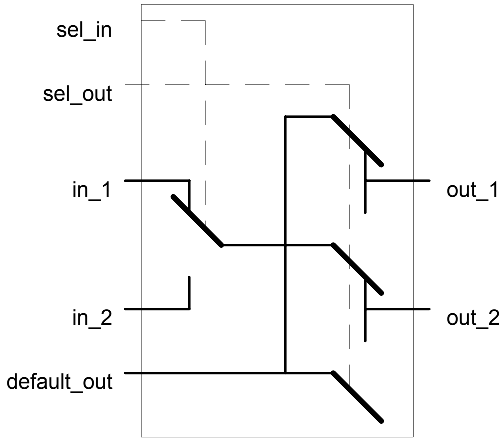


Short description:	Real demultiplexer with memory		
Description:	RESET overrides SET and LOAD inputs. SET overrides LOAD input. If ADDRESS is 0, all outputs are set to 0, just as if RESET input was active. See also DEMUX_A, DEMUX_B and DEMUX_T blocks.		
Call parameters:	set:	if TRUE, new input value is loaded in each cycle	(BOOL)
	load	new input value is loaded on rising edge	(BOOL)
	reset	if TRUE, all outputs are set to 0	(BOOL)
	address:	address of output (range 1 to 16)	(INT)
	input	input value to be demultiplexed	(REAL)
Return params:	out1...out16	outputs	(REAL)
	aerr	address error: set if address <0 or >16	(BOOL)

Prototype:	DEMUX_R (fs, fl, fr, addr, in);
	o1 := DEMUX_R.out1;
	err := DEMUX_R.aerr;		

DSEL_R

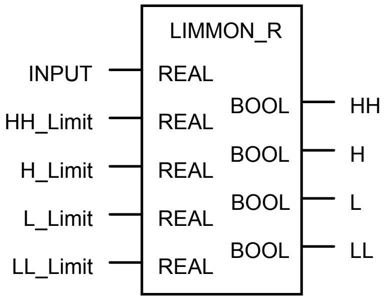




Short description:	Double independently operated real switch with two inputs
Description:	Please refer to the above relay diagram which should be clear enough.
Call parameters:	sel_in: Selects in_1 (FALSE) or in_2 (TRUE) (BOOL) sel_out: Selects out_1 (FALSE) or out_2 (TRUE) (BOOL) in_1: Real input 1 (REAL) in_2: Real input 2 (REAL) default_out: Value to be placed at non-selected output (REAL)
Return params:	out_1: Real output 1 (REAL) out_2: Real output 2 (REAL)

Prototype:	DSEL_R (selin, selout, inval1, inval2, defout);
	outval1 := DSEL_R.out_1;
	outval2 := DSEL_R.out_2;

LIMMON_R



Short description: Extended limit monitor of real value

Description: This function block implements the standard industrial 4-level limit monitor, supporting high alarm (HH), high prealarm (H), low prealarm (L) and low alarm (LL) levels. The 4 outputs indicate in which of the 5 regions the input value currently is:

above HH	HH output TRUE
between HH and H	H output TRUE
between L and H (inside "normal" band)	all outputs FALSE
between LL and L	L output TRUE
below LL	LL output TRUE

At most one of the outputs will be TRUE at any time, except when the limit values are not in increasing order, i.e. when the inequality
LL_Limit <= L_Limit < H_Limit <= HH_Limit
is not satisfied, in which case all 4 outputs will be set to TRUE.

Call parameters:

INPUT	(REAL)
HH_Limit	(REAL)
H_Limit	(REAL)
L_Limit	(REAL)
LL_Limit	(REAL)

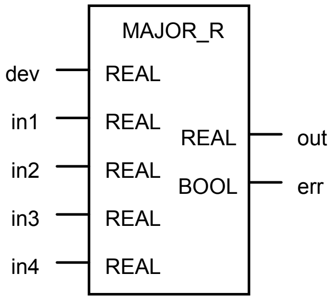
Return params:

HH	(BOOL)
H	(BOOL)
L	(BOOL)
LL	(BOOL)

Prototype:

```
LIMMON_R (in, hh_l, h_l, l_l, ll_l);
hh_alarm := LIMMON_R.HH;
h_alarm := LIMMON_R.H;
l_alarm := LIMMON_R.L;
ll_alarm := LIMMON_R.LL;
```

MAJOR_R



Short description: Majority selector for real inputs

Description: The majority selector calculates the mean value of all inputs. If exactly one input differs from the calculated mean value by more than dev, mean value is calculated once more, but that input is excluded from the calculation.
If more than one input deviates by more than dev, the mean value of all of the input values is calculated and the output err is set.
See also MAJOR_A function block.

Call parameters:

dev:	Max. permissible deviation between any input and the calculated mean value	(REAL)
in1:	Input 1	(REAL)
...		
in4:	Input 4	(REAL)

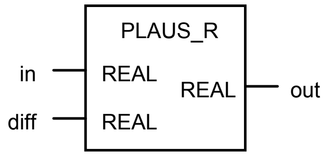
Return params:

out:	Mean value of inputs not deviating by more than dev from itself	(REAL)
err:	set when majority selection is impossible	(BOOL)

Prototype:

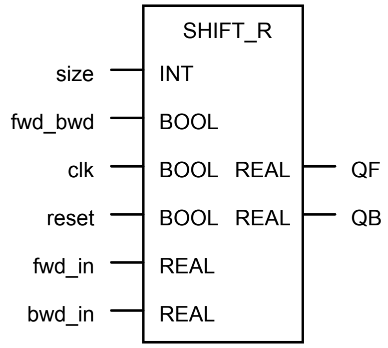
```
MAJOR_R  (deviation, i1, i2, i3, i4);
error := MAJOR_R.err;
mean := MAJOR_R.out;
```

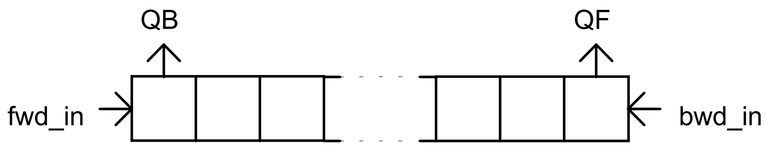
PLAUS_R



Short description:	Plausibility checking block for real input		
Description:	The block compares the difference between two succesive values sampled on the "in" input with the value on the "diff" input. If the difference of successive samples is less than "diff", the actual "in" value is forwarded to the output. If the difference exceeds "diff", the value to be output is calculated as the mean value of "in" samples in the 3 preceding cycles. In the cycle following this one, "in" is compared not to the preceding value, but with the last plausible value, i.e. one before it. If the difference in the cycle following the cycle in which the mean value was output is still above "diff", this is taken as the proof that both this and previous "in" values are plausible and the "in" value is normally forwarded to the output.		
Call parameters:	in:	input	(REAL)
	diff:	allowed difference	(REAL)
Return params:	out:	output	(REAL)
Prototype:	PLAUS_R (input, difference); o := PLAUS_R.out;		

SHIFT_R





Short description: Bidirectional real shift register of programmable length

Description: At each end of the shift register, there is one input and one output. When a forward shift is executed, the value applied to the fwd_in input appears immediately at the QB output. Likewise, when a backward shift is executed, the value applied to the bwd_in input appears immediately at the QF output. Initially (after power-up) and during reset, the whole register contains only zeros.
If a number less than 2 is applied to the size input, the shift register will have the length of 2. If a number greater than 256 is applied to the size input, the shift register will have the length of 256. The length of the register cannot be changed dynamically; value applied to the size input is read only in first cycle after power-up or reset.
See also SHIFT_A, SHIFT_B and SHIFT_T function blocks.

Call parameters:

	size:	register length (range 2...256)	(INT)
	fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
	clk:	shifts one place on rising edge	(BOOL)
	reset:	when TRUE, clears register to 0	(BOOL)
	fwd_in:	forward shift data input	(REAL)
	bwd_in:	backward shift data input	(REAL)

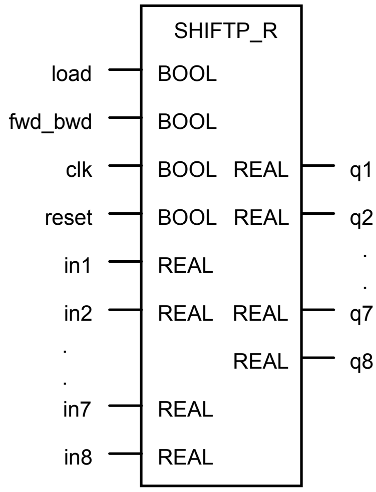
Return params:

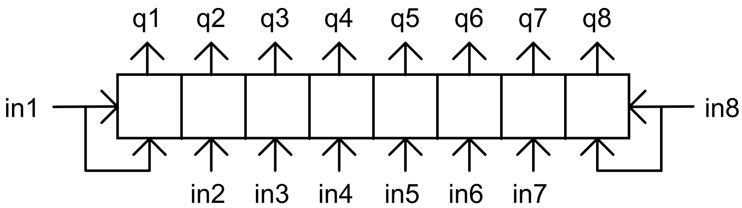
	QF:	forward shift data output	(REAL)
	QB:	backward shift data output	(REAL)

Prototype:

```
SHIFT_R (100, TRUE, FALSE, FALSE, 123.55, 47.2);
outfwd := SHIFT_R.QF;
outbwd := SHIFT_R.QB;
```

SHIFTP_R





Short description:	Bidirectional real shift register with 8 parallel inputs and outputs
Description:	Except that it is of fixed length and has parallel inputs and outputs, the functioning of this block is similar to that of SHIFT_R block. Initially (after power-up) and during reset, the whole register contains only zeros. Inputs in2...in7 are parallel inputs only, while inputs in1 and in8 are both parallel and serial inputs. See also SHIFTP_A, SHIFTP_B and SHIFTP_T function blocks.

Call parameters:

load:	on rising edge, register is loaded from parallel inputs	(BOOL)
fwd_bwd:	shift direction: forwards (TRUE)/backwards	(BOOL)
clk:	shifts one place on rising edge	(BOOL)
reset:	when TRUE, clears register to 0	(BOOL)
in1:	parallel input 1 and forward shift data input	(REAL)
in2:	parallel input 2	(REAL)
...		
in7:	parallel input 7	(REAL)
in8:	parallel input 8 and backward shift data input	(REAL)

Return params:

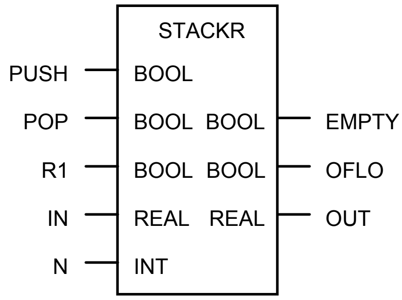
q1:	output 1	(REAL)
...		
q8:	output 8	(REAL)

Prototype:

```
SHIFTP_R (FALSE, TRUE, TRUE, FALSE, 2.1, 15., -4.11, 0., 100., 1.5,
5.2, -.7);
o1 := SHIFTP_R.q1;
```

```
...
o8 := SHIFTP_R.q8;
```

STACKR

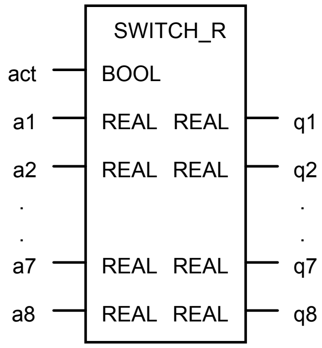


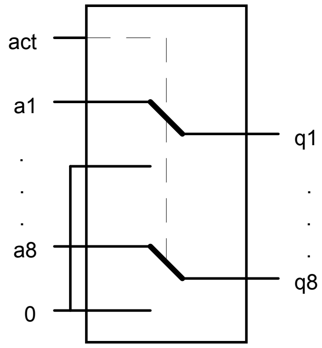
Short description:	Stack of real values	
Description:	-	
Call parameters:	PUSH: pushes IN value on rising edge	(BOOL)
	POP: pops value on rising edge	(BOOL)
	R1: TRUE resets stack to the "Empty" state	(BOOL)
	IN: value to be pushed	(REAL)
	N: maximum stack depth	(INT)
Return params:	EMPTY: TRUE indicates that the stack is empty	(BOOL)
	OFLO: TRUE indicates stack overflow	(BOOL)
	OUT: value at the top of stack	(REAL)

Prototype:

```
STACKR ( push_cmd, pop_cmd, reset_cmd, push_value, max_stack );
stackempty := STACKR.EMPTY;
overflow := STACKR.OFLO;
top_value := STACKR.OUT;
```

SWITCH_R





Short description: 8 single switches for real data

Description: -

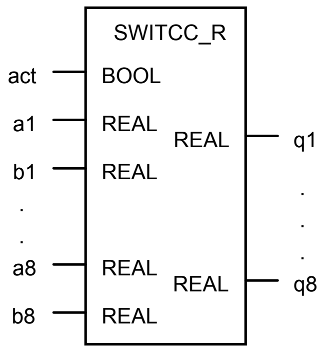
Call parameters:	act:	TRUE: inputs connected to outputs FALSE: zero output on all outputs	(BOOL)
	a1:	input to switch 1	(REAL)
	...		
	a8:	input to switch 8	(REAL)
Return params:	q1:	output of switch 1	(REAL)
	...		
	q8:	output of switch 8	(REAL)

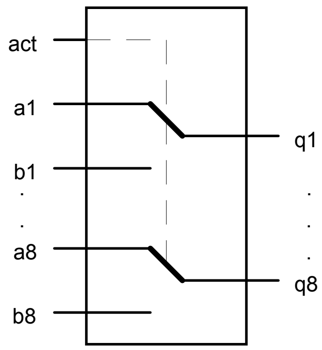
Prototype:

```
SWITCH_R (TRUE, 1., 22.2, -17.11, 4., 512., -93.745, 100., 0.);
out1 := SWITCH_R.q1;
...
out8 := SWITCH_R.q8;
```

Remarks: a) See also SWITCH_A, SWITCH_B and SWITCH_T function blocks.

SWITCC_R





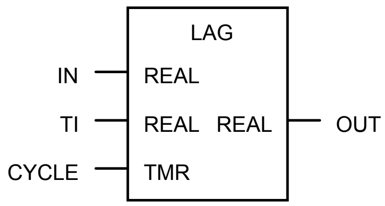
Short description:	8 changeover switches for real data		
Description:	-		
Call parameters:	act:	TRUE: A inputs connected to outputs FALSE: B inputs connected to outputs	(BOOL)
	a1:	switch 1, input A	(REAL)
	b1:	switch 1, input B	(REAL)
	...		
	a8:	switch 8, input A	(REAL)
	b8:	switch 8, input B	(REAL)
Return params:	q1:	output of switch 1	(REAL)
	...		
	q8:	output of switch 8	(REAL)

Prototype:

```
SWITCC_R (TRUE, 1., 22.6, -17.11, 4., ...  100., 0.52);
out1 := SWITCC_R.q1;
...
out8 := SWITCC_R.q8;
```

Remarks: a) See also SWITCH_A, SWITCH_B and SWITCH_T function blocks.

LAG



Short description: First order filter

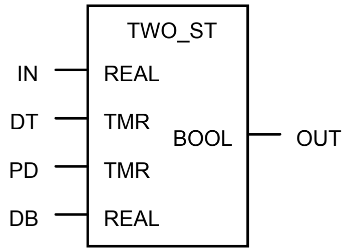
Description: The transfer function implemented by this block is: 1/(TI∗p + 1), where TI has the dimension of time, and p is the Laplace transform (p-plane) operator.
In the time domain, output OUT will follow the value of input IN with a certain lag. The rate of output change is proportional to the difference IN - OUT.
If a step function is applied to the input, an exponential waveform will be output, according to the following formula: OUT = IN∗(1 - exp(-t/TI)).
In this case, but also in any other case in which the IN value is stable for a reasonable amount of time, for practical purposes, the output can be considered to be equal to the input after an interval of 3..5∗TI.
Many natural phenomena (for example common heating processes) can be simulated using the LAG block.

Call parameters:	IN:	value to be filtered	(REAL)
	TI:	integral time (in units of 0,01s)	(REAL)
	CYCLE:	input sampling interval	(TMR)
Return parameter:	OUT:	filtered value	(REAL)

Prototype:
```
LAG (in, 10., 1s);
filtered = LAG.OUT;
```

Remarks: a) TI input is of type REAL although it is used for inputting a quantity that has a dimension of time, so type TMR would be appropriate. The reason for this awkward typing is in keeping the block compatible with similar standard blocks, delivered by CJ International. This may change with the following release of ISaGRAF!
ALL inputs/outputs in ALL function blocks with dimension of time and type REAL use a **unit of 10ms**. For example, to denote a time interval of 1s, value 100. must be applied to such an input.

TWO_ST



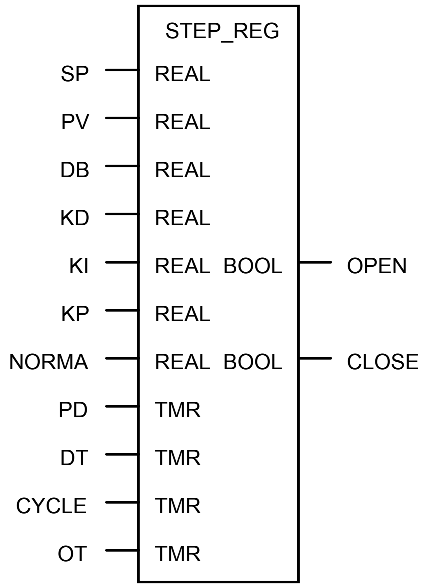
Short description:	Two state regulator
Description:	This block is used for switching the output load ON and OFF according to the value of the input signal. Two mechanisms for protecting both the contactor and the load from too frequent switching are provided: - deadband - minimum times for ON and OFF output states. If IN exceeds DB and if the time elapsed since last ON-to-OFF transition of OUT exceeds DT, OUT is set to ON. If IN falls below the negative value of DB and if the time elapsed since last OFF-to-ON transition of OUT exceeds PD, OUT is set to OFF.

Call parameters:	IN:	input value	(REAL)
	DT:	min. output non-activation time (dead time)	(TMR)
	PD:	minimum output ON pulse duration	(TMR)
	DB:	dead band	(REAL)
Return params:	OUT:	ON (TRUE)/OFF output	(BOOL)

Prototype:

```
TWO_ST (in, 5s, 20s, 1.5);
on_off := TWO_ST.OUT;
```

STEP_REG



Short description: 3-state controller with PID velocity algorithm

Description: Internally, the block can be conceived as being composed ot two sub-blocks: PID and 3-state Controller, connected in series.
Input to the PID sub-block is the deviation value : SP - PV, and its output is the internal variable Y.
The difference of PID output values in the current and previous calculation cycles, Y[k] - Y[k-1], is fed to the input of the 3-state Controller sub-block. If this value exceeds the positive value of the dead band, output OPEN is set to TRUE (ON) and output CLOSE is set to FALSE (OFF). If it exceeds the negative value of dead band, OPEN is set to FALSE and CLOSE is set to TRUE, providing that minimum output ON and OFF times have elapsed.
The minimum duration of TRUE (ON) state on any output is equal to PD and the minimum duration of FALSE (OFF) state is equal to DT.
Input sampling period (CYCLE) depends on the process that is controlled. Since PID parameters also depend on the process, a good rule of the thumb is to choose CYCLE to be approximately equal to Tmin/10, where Tmin is defined as min(KD,KP).

Call parameters:	SP:	setpoint	(REAL)
	PV:	process variable	(REAL)
	DB:	dead band	(REAL)
	KD:	derivative time	(REAL)
	KI:	integral time	(REAL)
	KP:	proportional gain	(REAL)
	NORMA:	maximum setpoint or process variable value	(REAL)
	PD:	minimum output ON time	(TMR)
	DT:	minimum output OFF time (dead time)	(TMR)
	CYCLE:	inputs sampling period	(TMR)
	OT:	time needed for fully opening/closing the valve	(TMR)
Return params:	OPEN: when TRUE (ON), valve is opening		(BOOL)
	CLOSE:	when TRUE (ON), valve is closing	(BOOL)

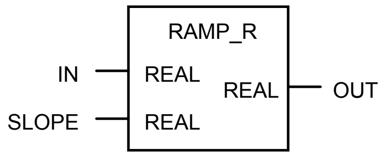
Prototype:

```
STEP_REG (in_real, db_real, kd_real, ki_real, kp_real, norma_real,
pd_tmr, dt_tmr, cycle_tmr, ot_tmr);
op_out := STEP_REG.OPEN;
cl_out := STEP_REG.CLOSE;
```

Remarks:

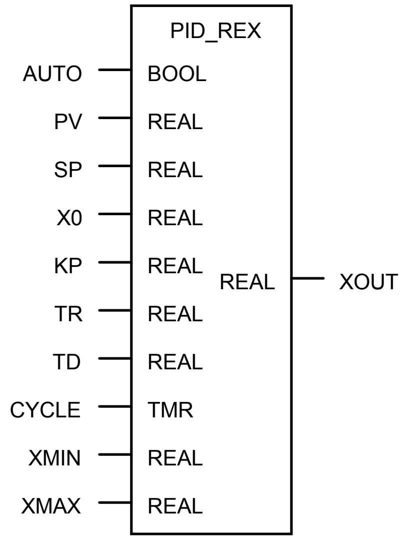
or an external automatic control specialist
b) KD and KI inputs are of type REAL although they are used for inputting quantites that have a dimension of time, so type TMR would be appropriate. The reason for this awkward typing is in keeping the block compatible with similar standard blocks, delivered by CJ International. This may change with the following release of ISaGRAF!
ALL inputs/outputs in ALL function blocks with dimension of time and type REAL use a **unit of 10ms**. For example, to denote a time interval of 1s, value 100. must be applied to such an input.

RAMP_R



Short description:	Ramp limiter for real signals		
Description:	Output (OUT) follows the input signal (IN) as long as the absolute value of its rate of change is below the value applied to the SLOPE input. When the absolute value of rate of change of input exceeds SLOPE, the rate of change of output il limited to +SLOPE or -SLOPE until the moment when OUT again becomes equal to IN. At that moment, tracking continues. SLOPE is expressed in units of 1/10ms, i.e. the numerical value applied to this input represents the maximum allowed change of the IN signal in the interval of 10ms. This makes the block compatible with blocks delivered by CJ International (e.g. derivator).		
Call parameters:	IN:	input	(REAL)
	SLOPE:	allowed input change	(REAL)
Return params:	OUT:	output	(REAL)
Prototype:	RAMP_R (inp, slope); outp := RAMP_R.out;		

PID_REX



Short description:	PID controller with real I/O, EXOR version
Description:	This PID Controller block implements the same algorithm as the standard PID_CJ function block delivered by CJI with the following differences: - current time is not read from the static variable LAST_DATE, since this doesn't work; a call to sys_readtim() function is done instead - when calculated output is outside the XMIN-XMAX interval, the appropriate limit value is output just as in PID_CJ; but, integral term is not reset to zero - it is recalculated so that the equation PTERM + ITERM + DTERM = XOUT remains satisfied.

Call parameters:

Call parameters:	AUTO:	Auto (TRUE)/Manual (FALSE) mode	(BOOL)
	PV:	Process variable (X)	(REAL)
	SP:	Setpoint (W)	(REAL)
	X0:	Value to be output in Manual mode	(REAL)
	KP:	Proportional gain	(REAL)
	TR:	Integral time	(REAL)
	TD:	Derivative time	(REAL)
	CYCLE :	Calculation and output updating period	(TMR)
	XMIN:	Min. value of output quantity (Y)	(REAL)
	XMAX:	Max. value of output quantity (Y)	(REAL)
Return params:	XOUT:	Output quantity (Y)	(REAL)

Prototype:

```
PID_REX (TRUE, temp_5, 120.5, manual_temp, kp, tr, td, 0s40, 0., 1000.);
heater := PID_REX.XOUT;
```

Remarks: Algorithm implemented in this block is the so-called "independent" PID algorithm. Kp multiplies all three terms (proportional, integral and derivative) in the following way:

error = SP - PV
XOUT = KP * (error + (1/TR)*integral(error) + TD*derivative(error))

For this type of algorithm, optimum KP, TR, TD parameters according to the Ziegler-Nichols method are:

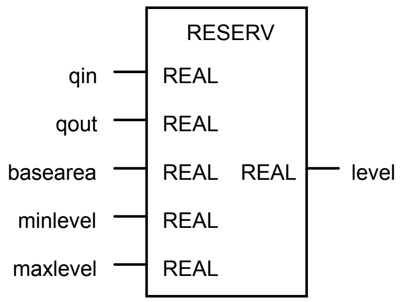
for P controller: KP = 0.5 *KPosc
for PI controller: KP = 0.45*KPosc TR = 0.83*Tosc
for PID controller: KP = 0.6 *KPosc TR = 0.5 *Tosc TD = 0.125*Tosc

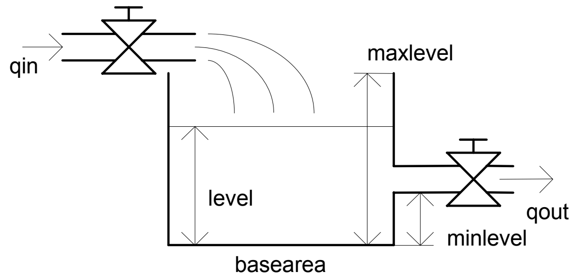
where KPosc is that KP which causes constant-amplitude closed-loop oscillations with only P-action enabled and Tosc is the period of these oscillations.

IMPORTANT! Integral time (TR) and derivative time (TD) MUST be input in units of 10ms, e.g. 100 should be applied to TR to indicate that integral time is 1 second.

See also PID_A function block.

RESERV





Short description: Reservoir (integrator) with input and output flow

Description: This block simulates a prismatic (cylindrical) reservoir with input and output pipe and upper (full) and lower (empty) limit level as depicted above.

Call parameters:

qin:	flow in input pipe	(REAL)
qout:	flow in output pipe	(REAL)
basearea:	area of the prismatic reservoir base	(REAL)
minlevel:	minimum level (reservoir empty)	(REAL)
maxlevel:	maximum level (reservoir spilling over)	(REAL)

Return params: level: current level in the reservoir (REAL)

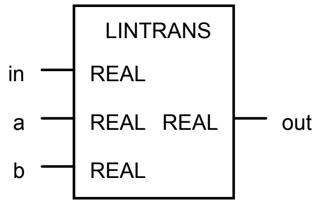
Prototype:

```
RESERV (0.5, 0.4, 1., 0., 5.);
lvl = RESERV.level;
```

Remarks: Reasonable care about units must be taken: if one wants level to be expressed in U units, then minlevel and maxlevel must also be expressed in U units, basearea must be expressed in U^2 units and qin and qout must be expressed in U^3/s units.

Example: For level in meters, minlevel could be 0, maxlevel 5m, basearea $1m^2$, qin could be $0,5m^3/s$ and qout $0,4m^3/s$. In this case, level would steadily rise at the rate of 0,1m/s until it reaches 5m, then it would remain at that value until qout becomes greater then qin.

LINTRANS



Short description: Linear transformation of real input

Description: out = a * in + b

Call parameters:	in:	input signal	(REAL)
	a:	multiplication factor	(REAL)
	b:	offset	(REAL)
Return params:	out:	output signal	(REAL)

Prototype:

```
LINTRANS (input, 10., 2.);
out := LINTRANS.out;
```

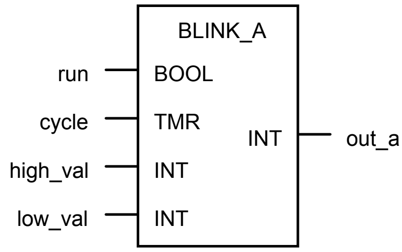
3.6 Signal Generation FBs

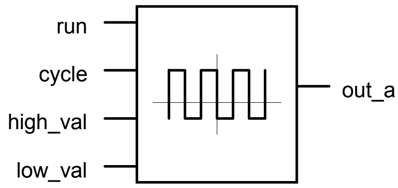
Standard Signal Generation Function Blocks delivered by CJ International are not described in this document. For their full description, please refer to the ISaGRAF User's Manual.

For quick reference, here is just a brief listing of these function blocks, containing the function block name and short description::

BLINK	Blinking BOOLEAN signal

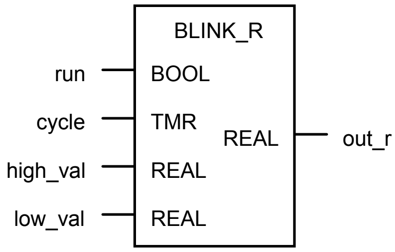
BLINK_A

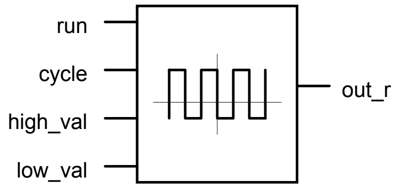




Short description:	Alternating analog (integer) signal generation		
Description:	Based on standard boolean BLINK function block. Once enabled, the output will toggle continuously between high_val and low_val values with the period equivalent to cycle. When disabled, low_val will be output.		
Call parameters:	run:	Bink enable	(BOOL)
	cycle:	Blinking period	(TMR)
	high_val:	1st level to be output	(INT)
	low_val:	2nd level to be output	(INT)
Return params:	out_a:	Output signal	(INT)
Prototype:	BLINK_A (enab, period, hilev, lolev); signal = BLINK_A.out_a;		

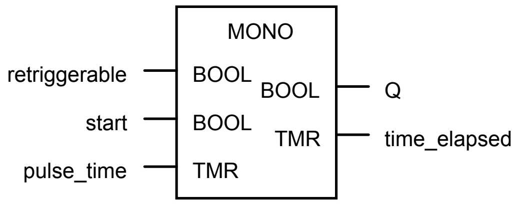
BLINK_R





Short description:	Alternating real signal generation
Description:	Based on standard boolean BLINK function block. Once enabled, the output will toggle continuously between high_val and low_val values with the period equivalent to cycle. When disabled, low_val will be output.
Call parameters:	run: Bink enable (BOOL) cycle: Blinking period (TMR) high_val: 1st level to be output (REAL) low_val: 2nd level to be output (REAL)
Return params:	out_r: Output signal (REAL)
Prototype:	BLINK_A (enab, period, hilev, lolev); signal = BLINK_A.out_r;

MONO

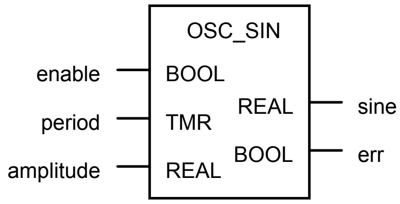


Short description:	Monostable element		
Description:	-		
Call parameters:	retriggerable:	if TRUE, monostable can be retriggered	(BOOL)
	start:	positive edge triggers monostable	(BOOL)
	pulse_time:	duration of monostable pulse	(TMR)
Return params:	Q:	monostable output	(BOOL)
	time_elapsed:	time elapsed from last pos. edge of start	(TMR)

Prototype:

```
MONO (TRUE, TRUE, 5s);
out := MONO.Q;
rest := MONO.time_elapsed;
```

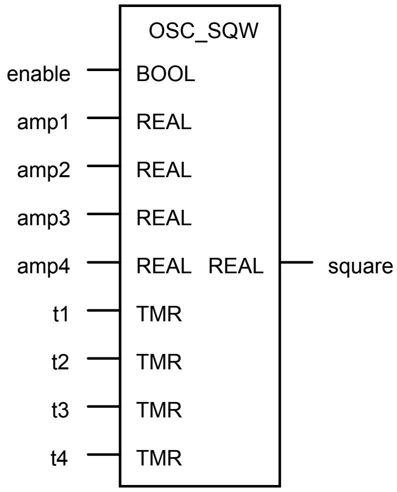
OSC_SIN



Short description:	Sine wave oscillator		
Description:	The sine wave is created from a table containing 256 amplitude values per period. If enable is FALSE, output is set to 0. The period should be minimally 20 times longer than the duration of one PLC cycle, otherwise the err output is set and zero is output on the sine output.		
Call parameters:	enable:	oscillator enable	(BOOL)
	period:	period of oscillations	(TMR)
	amplitude:	amplitude of sine wave	(REAL)
Return params:	sine:	sine wave	(REAL)
	err:	set to TRUE if period is too short	(BOOL)

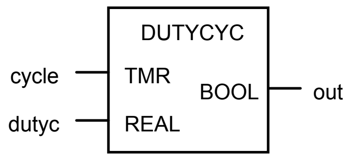
Prototype:	OSC_SIN (TRUE, per, amp);
	osc := OSC_SIN.sine;
	error := OSC_SIN.err;		

OSC_SQW



Short description:	Four-level square wave oscillator		
Description:	One full period of the square wave is composed of 4 parts with potentially different duration and with the output amplitude selectable for each sub-period independently. If enable is FALSE, output is set to 0.		
Call parameters:	enable:	oscillator enable	(BOOL)
	amp1:	amplitude during time T1	(REAL)
	amp2:	amplitude during time T2	(REAL)
	amp3:	amplitude during time T3	(REAL)
	amp4:	amplitude during time T4	(REAL)
	t1:	time T1	(TMR)
	t2:	time T2	(TMR)
	t3:	time T3	(TMR)
	t4:	time T4	(TMR)
Return params:	square:	square wave output	(REAL)
Prototype:	OSC_SQW (start, 0., 1., 0., -1., 1s, 1s, 1s, 1s); sqw := OSC_SQW.square;		

DUTYCYC



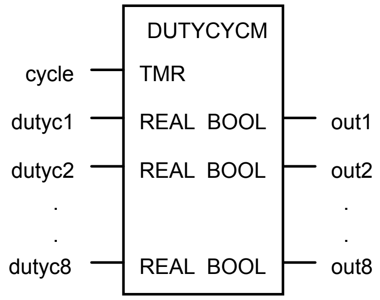
Short description:	Digital oscillator with variable duty-cycle
Description:	Within each cycle, the out signal will be TRUE for dutyc percent of the cycle duration and FALSE for (100 - dutyc) percent of the cycle duration. The first cycle after power-up will begin with out set to TRUE, except if dutyc is less than or equal to zero. If dutyc is less than or equal to zero, a steady FALSE signal will be output. If dutyc is greater than or equal to 100%, a steady TRUE signal will be output.
REMARK:	One should take into account the limited time resolution of the PC-based target: 55ms (one BIOS tick). No regular pulse can be shorter than this interval. Therefore, some lower limit on the CYCLE input variable to DUTYCYC and DUTYCYCM blocks must be respected. As a reasonable value, 10 seconds for this limit is proposed. (Since the unit for CYCLE is 10ms, 10 seconds are represented by number 1000 applied to CYCLE input of the block.) In this case, 55ms will be approximately 0,5% of the cycle and this amount of error introduced by the finite resolution can be accepted in most applications. If CYCLE is shorter than this, one must be aware of the increased influence of the resolution-related error, rising with the decreasing CYCLE value.

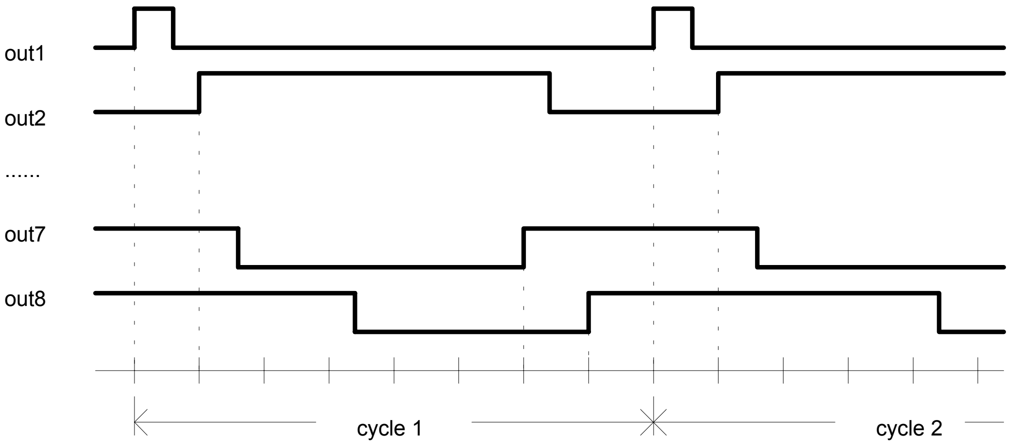
Call parameters:	cycle:	Cycle time	(TMR)
	dutyc:	Duty-cycle percentage (0...100%)	(REAL)
Return params:	out:	Output waveform	(BOOL)

Prototype:

```
DUTYCYC (1m30s, 20.5);
wave := DUTYCYC.out;
```

DUTYCYCM





Short description: Multiple digital oscillator with variable duty-cycle

Description: Within each cycle, on each channel the outX signal will be TRUE for dutycX percent of the cycle duration and FALSE for (100 - dutycX) percent of cycle duration.
The FALSE-to-TRUE transition on each two successive channels will be apart by 0.125*cycle. In this way, FALSE-to-TRUE (OFF-to-ON) transitions will be uniformly spaced over the cycle duration.
This is very important since this function block is typically used as a pulse-width modulator whose outputs drive electrical heaters, known to create current surges in a short interval following the switch-on. By spacing the OFF-to-ON transitions uniformly over the cycle duration, the power supply can be designed for just $Isurge_{max}$ instead of $8*Isurge_{max}$, which is the worst case for non-synchronized channels.
If dutyc is less than or equal to zero for some channel, a steady FALSE signal will be output on that channel's output.
If dutyc is greater than or equal to 100% for some channel, a steady TRUE signal will be output on that channel's output.

REMARK: One should take into account the limited time resolution of the PC-based target: 55ms (one BIOS tick). No regular pulse can be shorter than this interval. Therefore, some lower limit on the CYCLE input variable to DUTYCYC and DUTYCYCM blocks must be respected.

As a reasonable value, 10 seconds for this limit is proposed. (Since the unit for CYCLE is 10ms, 10 seconds are represented by number 1000 applied to CYCLE input of the block.) In this case, 55ms will be approximately 0,5% of the cycle and this amount of error introduced by the finite resolution can be accepted in most applications.
If CYCLE is shorter than this, one must be aware of the increased influence of the resolution-related error, rising with the decreasing CYCLE value.

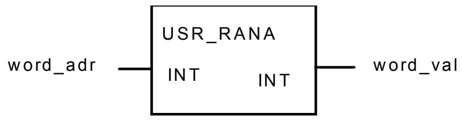
Call parameters:	cycle:	Cycle time	(TMR)
	dutyc1:	Duty-cycle percentage, channel 1 (0...100%)	(REAL)
	dutyc2:	Duty-cycle percentage, channel 2 (0...100%)	(REAL)
	...		
	dutyc8:	Duty-cycle percentage, channel 8 (0...100%)	(REAL)
Return params:	out1:	Output waveform, channel 1	(BOOL)
	out2:	Output waveform, channel 2	(BOOL)
	...		
	out8:	Output waveform, channel 8	(BOOL)

Prototype:

```
DUTYCYCM (1m30s, 10.7, 22.5, 30., 49.2, 55.5, 65., 78., 83.1);
wave1 := DUTYCYCM.out1;
...
wave8 := DUTYCYCM.out8;
```

3.7 Variable Access FBs

USR_RANA



Short description: Read one analog variable

Description: Read the analog value of a variable using the Network Address to select it.
If the variable is not found (Network Address is not defined) the returned value is 0
The returned value is always analog even if the variable is of different type.

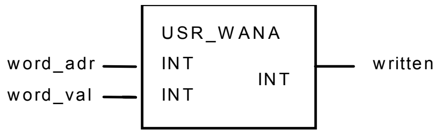
Call parameters: WORD_ADR: network address (INT)

Return params: WORD_VAL: analog output (INT)

Prototype:

```
USR_RANA (16#1000);
value := USR_RANA.WORD_VAL;
```

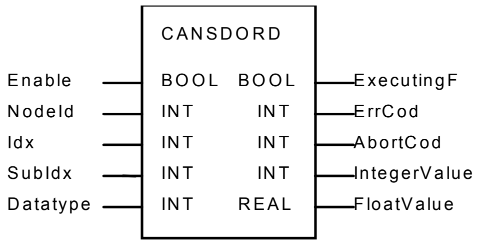
USR_WANA



Short description:	Write one analog variable
Description:	Write a analog value to a variable using the Network Address to select it. If the variable is not found (Network Address is not defined) the returned value is FALSE The value is always written as analog even if the variable is of different type.
Call parameters:	WORD_ADR: network address (INT) WORD_VAL: analog output (INT)
Return params:	WRITTEN: TRUE if successful (BOO)
Prototype:	USR_WANA (16#1000,1234); isok := USR_WANA.WRITTEN;

3.8 Hardware Specific FBs

CANSDORD



Short description:	Read one variable in CANopen node using SDO protocol
Description:	Read an element of remote database using SDO protocol, as defined in CANopen standard document DS301.

Call parameters:

ENABLE:	start the read operation, must be reset by user	(BOO)
NODEID:	node number od the remote node	(INT)
IDX:	index in remote database	(INT)
SUBIDX:	subindex in database	(INT)
DATATYPE	type of data to be read (see table below)	(INT)

Return params:

EXECUTINGF: TRUE if operation is still in progress	(BOO)
ERRCOD:state of the last execution (see table below)	(BOO)
ABORTCOD: code answered by remote node (see table below)	(BOO)
INTEGERVALUE: value read represented as integer number	(INT)
FLOATVALUE: value read represented as float numbe	(REAL)

Prototype:

```
CANSDORD(TRUE, node, index, subindex, INT32);
isrunning := CANSDORD.EXECUTINGF;
error := CANSDORD.ERRCOD;
answer:= CANSDORD.ABORTCOD;
Value := CANSDORD.INTEGERVALUE;
```

Remarks: several SDO operations cannot be executed on the same node at the same time.

Data Type Table

1	BOOL
2	INT8
3	INT16
4	INT32
5	UINT8
6	UINT16
7	UINT32
8	FLOAT

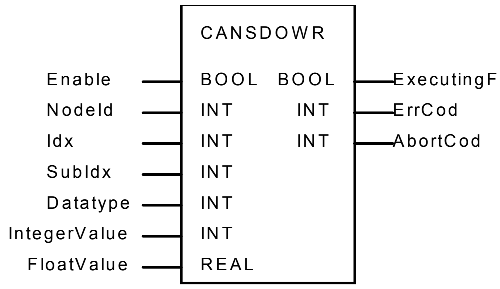
Error Codes Table

12	Too many SDO/PDO

13 Invalid parameter/s
14 Invalid reply from remote SDO server
15 Returned size differs from requested size
16 No reply timeout

Abort code Description
0503 0000h Toggle bit not alternated.
0504 0000h SDO protocol timed out.
0504 0001h Client/server command specifier not valid or unknown.
0504 0002h Invalid block size (block mode only).
0504 0003h Invalid sequence number (block mode only).
0504 0004h CRC error (block mode only).
0504 0005h Out of memory.
0601 0000h Unsupported access to an object.
0601 0001h Attempt to read a write only object.
0601 0002h Attempt to write a read only object.
0602 0000h Object does not exist in the object dictionary.
0604 0041h Object cannot be mapped to the PDO.
0604 0042h The number and length of the objects to be mapped would exceed PDO length.
0604 0043h General parameter incompatibility reason.
0604 0047h General internal incompatibility in the device.
0606 0000h Access failed due to an hardware error.
0607 0010h Data type does not match, length of service parameter does not match
0607 0012h Data type does not match, length of service parameter too high
0607 0013h Data type does not match, length of service parameter too low
0609 0011h Sub-index does not exist.
0609 0030h Value range of parameter exceeded (only for write access).
0609 0031h Value of parameter written too high.
0609 0032h Value of parameter written too low.
0609 0036h Maximum value is less than minimum value.
0800 0000h general error
0800 0020h Data cannot be transferred or stored to the application.
0800 0021h Data cannot be transferred or stored to the application because of local control.
0800 0022h Data cannot be transferred or stored to the application because of the present device state.
0800 0023h Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

CANSDOWR



Short description:	Write one variable in CANopen node using SDO protocol		
Description:	Write an element of remote database using SDO protocol, as defined in CANopen standard document DS301.		
Call parameters:	ENABLE:	start the read operation, must be reset by user	(BOO)
	NODEID:	node number od the remote node	(INT)
	IDX:	index in remote database	(INT)
	SUBIDX:	subindex in database	(INT)
	DATATYPE	type of data to be read (see table below)	(INT)
	INTEGERVALUE:	value read represented as integer number	(INT)
	FLOATVALUE:	value read represented as float numbe	(REAL)
Return params:	EXECUTINGF:	TRUE if operation is still in progress	(BOO)
	ERRCOD:	state of the last execution (see table below)	(BOO)
	ABORTCOD:	code answered by remote node (see table below)	(BOO)

Prototype:

```
CANSDORD(TRUE, node, index, subindex, INT32, 1234, 0.0);
isrunning := CANSDORD.EXECUTINGF;
error := CANSDORD.ERRCOD;
answer:= CANSDORD.ABORTCOD;
```

Remarks: several SDO operations cannot be executed on the same node at the same time.

Data Type Table

1	BOOL
2	INT8
3	INT16
4	INT32
5	UINT8
6	UINT16
7	UINT32
8	FLOAT

Error Codes Table

12	Too many SDO/PDO
13	Invalid parameter/s
14	Invalid reply from remote SDO server
15	Returned size differs from requested size

16 No reply timeout

Abort code Description
0503 0000h Toggle bit not alternated.
0504 0000h SDO protocol timed out.
0504 0001h Client/server command specifier not valid or unknown.
0504 0002h Invalid block size (block mode only).
0504 0003h Invalid sequence number (block mode only).
0504 0004h CRC error (block mode only).
0504 0005h Out of memory.
0601 0000h Unsupported access to an object.
0601 0001h Attempt to read a write only object.
0601 0002h Attempt to write a read only object.
0602 0000h Object does not exist in the object dictionary.
0604 0041h Object cannot be mapped to the PDO.
0604 0042h The number and length of the objects to be mapped would exceed PDO length.
0604 0043h General parameter incompatibility reason.
0604 0047h General internal incompatibility in the device.
0606 0000h Access failed due to an hardware error.
0607 0010h Data type does not match, length of service parameter does not match
0607 0012h Data type does not match, length of service parameter too high
0607 0013h Data type does not match, length of service parameter too low
0609 0011h Sub-index does not exist.
0609 0030h Value range of parameter exceeded (only for write access).
0609 0031h Value of parameter written too high.
0609 0032h Value of parameter written too low.
0609 0036h Maximum value is less than minimum value.
0800 0000h general error
0800 0020h Data cannot be transferred or stored to the application.
0800 0021h Data cannot be transferred or stored to the application because of local control.
0800 0022h Data cannot be transferred or stored to the application because of the present device state.
0800 0023h Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

4. Index