Reference Manual

Sedona Open Control

CONTEMPORARY CONTROLS®

Trademarks

BASautomation, Contemporary Controls and CTRLink are registered trademarks of Contemporary Control Systems, Inc. BACnet is a registered trademark of the American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. Other product names may be trademarks or registered trademarks of their respective companies.

Copyright

Contemporary Control Systems, Inc.	Tel:	+1-630-963-7070
2431 Curtiss Street	Fax:	+1-630-963-0109
Downers Grove, Illinois 60515 USA	E-mail:	info@ccontrols.com
Contemporary Controls Ltd	Tel:	+44 (0)24 7641 3786
14 Bow Court	Fax:	+44 (0)24 7641 3923
Fletchworth Gate	E-mail:	info@ccontrols.co.uk
Coventry CV5 6SP UK		
Contemporary Controls (Suzhou) Co. Ltd	Tel:	+86 512 68095866
19F, Metropolitan Towers	Fax:	+86 512 68093760
No.199 Shishan Road	E-mail:	info@ccontrols.com.cn
Suzhou New District, Suzhou, PR		
China 215009		
19F, Metropolitan Towers		
Contemporary Controls GmbH	Tel:	+49 (0)341 520359 0
Fuggerstraße 1 B	Fax:	+49 (0)341 520359 16
04158 Leipzig Deutschland	E-mail:	info@ccontrols.de

Disclaimer

Contemporary Control Systems, Inc. reserves the right to make changes in the specifications of the product described within this manual at any time without notice and without obligation of Contemporary Control Systems, Inc. to notify any person of such revision or change.

Contents

1 Introduction to Sedona Open Control

Sedona is a component-oriented programming language. Using a Sedona tool, such as Contemporary Controls' Sedona Application Editor, components (function blocks) are assembled onto wiresheets creating applications. This language is ideally suited for graphical representation of control strategies. It has a similar look-and-feel to the popular Niagara Framework™ and it is IP-based. Those with experience with Niagara Framework will have no problem understanding Sedona. For those without Niagara experience, the graphical representation of components linked on a wiresheet to create applications is intuitive and can be easily learned with a minimum of training.

As a Sedona developer, Contemporary Controls has produced several products built on Sedona. In addition to developing custom Sedona components that complement those developed by Tridium (original developer of Sedona), Contemporary Controls has developed a Java-based Sedona tool called Sedona Application Editor (SAE) for those individuals who do not have access to a suitable Sedona tool. All that is needed are the kits and manifests for the connected Sedona device. Contemporary Controls provides the necessary kits and manifests for Tridium's core components and for Contemporary Controls' developed components with the BAScontrol Toolkit download. Also included in the free download are the Sedona Applications Editor, BASbackup for a complete project backup of a Contemporary Controls' Sedona controller, and the BASemulator which is the next best thing to a real Contemporary Controls' Sedona controller.

There are no licenses required for using or developing the Sedona technology or for using the BAScontrol Toolset. Contemporary Controls is committed to keeping Sedona "open." For those interested in obtaining the original Sedona technology, they can download the technology from the Sedona Alliance.

2 Tridium's Sedona 1.2 Core Component Descriptions

Components are typically sorted by function and deployed in kits which are available from Contemporary Controls and other members of the Sedona community. Kits without a company name are from Tridium. Kits with a company name and no product name are from a Sedona community member and these components can be used with other Sedona devices. Kits with both a company name and product name are platform dependent thereby limiting portability. What follows are component descriptions of Tridium-release kits compliant with Sedona release 1.2.28. In prior releases by Tridium, Tridium put all the Sedona components into one kit called the Control kit. With this final release, Tridium organized these components into multiple kits with more meaningful kit names. This organizational structure has been maintained in this document with components listed by the newer kit names.

It is recommended that these kits or components not be modified from their original form to ensure the portability of components and kits among members of the Sedona community. If changes or enhancements need to be made by a Sedona developer, the developer can use what currently exists, modify, or enhance the component, but then save the result as a custom component using the developer's name. The developer would then create a kit name that also includes the developer's name making it clear it is a custom kit.

When studying these components keep the following in mind. Boolean variables are assumed if there is a false/true state indication. Integers (32-bit signed integers) are shown as whole numbers while floats (32-bit floating point) are shown with a decimal point. The Sedona Application Editor (SAE) tool only shows an expanded view of the component thereby showing all the component slots. The SAE view is the one used for the following component images.

The core kits come installed with every Sedona controller.

- Basic Schedule Kit (basicSchedule)
- Date Tine STD Kit (datetimeSTD)
- Function Kit (func)
- HVAC Kit (hvac)
- Logic Kit (logic)
- Math Kit (math)
- Priority Kit (pricomp)
- Timing Kit (timing)
- Types Kit (types)
- Sys Kit (sys)

2.1   Basic Schedule Kit (basicSchedule)

DailySchedule represents a simple daily schedule with up to two active periods. Each active period is defined by a start time and duration. If the duration is zero, the period is disabled. If the periods overlap, then the first period (defined by Start1 and Dur1) takes precedence. If the duration extends past midnight, then the active period will span two separate calendar days. There are two components in the kit — one for Boolean outputs and the other for floats. Both kits rely upon the time being set in the target hardware.

Duration periods — *Dur1 and Dur2* — are configured in minutes from zero to 1439 minutes.

DailySc	
basicSchedule::DailyScheduleBool	
Start1	0:0
Dur1	0:0
Start2	0:0
Dur2	0:0
Val1	false
Val2	false
DefVal	false
Out	false

Daily Schedule Boolean — two-period Boolean scheduler

Configure Def Val to the intended output value if there are no active periods. Configure Val1 and Val2 for the desired output values during period 1 and period 2 respectively.

Out = Def Val if no active periods
Out = Val1 if period 1 is active
Out = Val2 if period 2 is active

DailyS1	
basicSchedule::DailyScheduleFloat	
Start1	0:0
Dur1	0:0
Start2	0:0
Dur2	0:0
Val1	0.0
Val2	0.0
DefVal	0.0
Out	0.0

Daily Schedule Float — two-period float scheduler

Configure Def Val to the intended output value if there are no active periods. Configure Val1 and Val2 for the desired output values during period 1 and period 2 respectively.

Out = Def Val if no active periods
Out = Val1 if period 1 is active
Out = Val2 if period 2 is active

2.2   Date Time STD Kit   (datetimeStd)

DateTim	
datetimeStd::DateTimeServiceStd	
Nanos	540425967000000000
Hour	22
Minute	19
Second	27
Year	2017
Month	2
Day	14
DayOfWeek	2
UtcOffset	0
OsUtcOffset	false
Tz	

The *DateTim* component is the only component in the Date Time STD Kit. This component relies upon a properly functioning real-time clock implemented in hardware. Once date and time are configured, this component can be dragged onto a worksheet allowing individual integer outputs to be wired to logic if so desired. However, it is not necessary to have the component on the wiresheet at all. If the *DailySchedule* components are to be used, they will function properly without the presence of the *DateTim* component. The start and stop times in the *DailySchedule* key on the daily time generated by the *DateTim* component regardless if this component is on the wiresheet or not.

Property	Value
DateTim	
Name	DateTim
Meta	67174401
Nanos	540507745000000...
Hour	16
Minute	2
Second	25
Year	2017
Month	2
Day	15
DayOfWeek	3
UtcOffset	-18000
OsUtcOffset	true
Tz	

Please Note when Using Contemporary Control's Controllers with SAE

When using Contemporary Controls' controllers, make sure that the OsUtcOffset slot is set to *true* in the Property pane. Just click on the DateTim component and its properties will appear on the right side of the screen. To avoid confusing time settings, do not set the time with this component. Set the time using the Set Time web page on the controller which provides more flexibility. You can set time zone, daylight saving time and in some instances Network Time Protocol support using just the web page. These settings will then set this Sedona component properly.

2.3　Function Kit　(func)

Cmpr
func::Cmpr

Xgy	false
Xey	true
Xly	false
X	0.0
Y	0.0

Comparison math — comparison (<=>) of two floats

If X > Y then Xgy is true
If X = Y then Xey is true
If X < Y then Xly is true

Count
func::Count

Out	0
In	false
Preset	0
Dir	up
Enable	false
R	false

Integer counter — up/down counter with integer output

Counts on the false to true transition of *In*. If *Dir = true* the counter counts up to the maximum value of the integer. If *Dir = false* the counter counts down but not below zero. For counting to occur, *Enable* must be true. The counter can be preset. If *R = true* and *Enable = true*, then Out equals the preset value and will not count.

Freq
func::Freq

Pps	0
Ppm	0
In	false

Pulse frequency — calculates the input pulse frequency

Pps = number of pulses per second of In
Ppm = number of pulses per minute of In

Hystere
func::Hysteresis

In	0.0
Out	false
RisingEdge	50.0
FallingEdge	50.0

Hysteresis — setting on/off trip points to an input variable

There are two internal floats called *Rising Edge* and *Falling Edge* which are configurable. If *Rising Edge* is greater than *Falling Edge*, then the following is true.

If In > Rising Edge then Out = true and will remain in that state until
 In < Falling Edge
If Rising Edge is less than Falling Edge, then the action is inverted.

IRamp
func::IRamp

Out	72
Min	0
Max	100
Delta	1
Secs	1

IRamp — generates a repeating triangular wave with an integer output

There are four configurable float parameters — *Min, Max, Delta* and *Secs*. For every scan cycle, the output increments by *Delta* units until the output equals the *Max* value at which time it decrements until *Min* is reached. The result is a triangular wave with limits of *Max* and *Min* and an incremental rate of *Secs* units.

Limiter	
func::Limiter	
Out	0.0
In	0.0
LowLmt	0.0
HighLmt	0.0

Limiter — Restricts output within upper and lower bounds

High Lmt and Low Lmt are configurable floats.
If In > High Lmt then Out = High Lmt
If In < Low Lmt then Out = Low Lmt
If In < High Lmt and > Low Lmt then Out = In

Lineari	
func::Linearize	
Out	null
In	0.0
X0	0.0
Y0	0.0
X1	0.0
Y1	0.0
X2	0.0
Y2	0.0
X3	0.0
Y3	0.0
X4	0.0
Y4	0.0
X5	0.0
Y5	0.0
X6	0.0
Y6	0.0
X7	0.0
Y7	0.0
X8	0.0
Y8	0.0
X9	0.0
Y9	0.0

Linearize — piecewise linearization of a float

For piecewise linearization of a nonlinear input, there are ten pairs of x,y parameters that must be configured into this component. The x,y pairs indicate points along the input curve. For an x value of the input, there should be a corresponding y value of the output. For input values between these points, the component will estimate the output based upon the linear equation:

$$Out = y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0}$$

where y is the value for input value x between coordinates x_0, y_0 and x_1, y_1

Note: The Linearize component could yield inconsistent results when creating negative slope output. Users are encouraged to use generate components—Gen8T, Gen16T, Gen32T—from the General Table in the CControls_Math2 kit.

LP	●
func::LP	
Enable	true
Sp	0.0
Cv	0
Out	0.0
Kp	1
Ki	0
Kd	0
Max	100
Min	0
Bias	0
MaxDelta	0
Direct	true
ExTime	1000

LP — proportional, integral, derivative (PID) loop controller

The LP component is much more complex component requiring an explanation of the numerous configurable parameters. *Sp* is the *setpoint* or the desired outcome. *Cv* is the *controlled variable* which we are trying to make equal to the setpoint. The difference between *Cv* and *Sp* is the *error signal* (e) that drives the *output variable Out* used to manipulate the *controlled variable*. There are three gain factors *Kp, Ki, Kd* — called *tuning parameters* — for each of the three modes of the controller: *proportional, integral, and derivative*. Setting a gain factor to zero effectively disables that mode. Setting Kp to zero would completely disable the controller. Typical controller operation is either:

Proportional-only (P)
Proportional plus reset (integral) (PI)
Proportional plus reset plus rate (PID)

In HVAC applications, P and PI are the most common. PID is seldom used.

Enable must be set true if loop action is to occur. If *Enable* is set to false, control action ceases and the output will remain at its last state. However, if *Ki* or *Kd* are non-zero, internal calculations will continue.

Note: If this action is not acceptable, users are encouraged to use the EnhPID component in the CControls_HVAC kit which provides a more definitive output control when *Enable* is false.

If *Direct* is equal to *true*, then the output will increase if the *Cv* becomes greater than *Sp*. If this was a temperature loop, this would be considered being in *cooling mode*. If *Direct* is equal to *false*, then the output will decrease if the *Cv* becomes greater than the *Sp*. If this was a temperature loop, this would be considered being in *heating mode*. Notice that by entering negative gain factors, the action of the controller is reversed.

Max and *Min* are limits on the output's swing and are considered the absolute boundaries to the controller's throttling range (proportional control range). Basically, the *LP* component includes *Limiter* functionality.

Bias sets the output's offset. Sometimes *bias* is called manual reset to correct an output error with a large proportional band. It is usually only used with proportional-only control. The amount of bias is not influenced by the proportional gain *Kp*. Bias is also used on split-range control systems that will be discussed shortly.

Ex Time is the amount of time in milliseconds that the control loop is solved. Typical times are from 100–1000 ms with a default of 1000. Most HVAC loops are slow acting and therefore solving loops faster brings no benefit.

In the following discussion on setting the gain factors, assume we need a temperature controller enabled for direct action and that the output can swing from –50% to +50%. When the output ranges from 0 to 50%, a proportional cooling valve is modulated. When the output ranges from 0 to –50%, a proportional heating valve is modulated. At 0% output no valve is open. This is called a split range control system. *Max* and *Min* are set to –50 and +50 respectively. When we force the controller output from maximum heat to maximum cooling (100% output change), we notice that we can effect a change in our process temperature of 20°. This becomes our throttling range. In the real world, conducting this test might be difficult.

Now we need to set the three tuning parameters. We first begin by setting *Ki* and *Kd* to zero, thereby creating a proportional-only controller. The controller equation therefore becomes:

Out = Kp(e) + Bias where e = Cv − Sp and Bias equals zero

Our first guess at *Kp* is 5 because we know that a 100% change in output yielded a 20° change in process temperature. This assumes that we can cool with the same efficiency as we can heat which may not be the case. By having a *Kp* of 5, the output will remain linear over this wide range. Notice that if there is no error signal (*Cv-Sp* is equal to zero), the output will then equal the *bias*, but in this case the bias is zero. The value 5 is entered into *Kp* and a disturbance is introduced into the process such as a step change in the setpoint. If the process continues to oscillate between heating and cooling and never settles down, then we must reduce our proportional gain *Kp* which increases our proportional band (1/*Kp* times 100% is the proportional band). Assume we achieve a stable system with *Kp* at 5 (proportional band at 20%) but based on the load on the system we notice that the output reached 70%. Our setpoint is at 70°, but our controlled temperature is 74°. Temperature is stable, but we have a 4° offset. This is the inherent difficulty with proportional-only control, there is an offset depending upon the value of the output. We have two choices. We can increase the proportional gain to 10 because we do not need a 20° range in input, but we risk oscillation. The second approach is to "reset the output manually" by increasing the bias. Approach one will never solve the problem but will minimize it, and there is a better method to approach two and that is called *automatic reset* — or adding reset action by adding a *Ki* term. The new controller equation becomes:

Out = Kp(e + Ki∫e dt)   (Bias is disabled when Ki is non-zero.)

If there remains an error signal (e ≠ 0), then the integral of the error over time will continue to drive the output until the error is driven to zero. The amount of action is determined by the *Ki* term. Notice that the integral term in the equation is also multiplied by the proportional gain before being applied to the output. The *Ki* coefficient is defined in units of repeats per minute. Too large a value can cause overshoot while too small a value will make the control system sluggish. The final setting *Kp* and *Ki* is done in the field based upon system response.

The third parameter is the rate parameter *Kd* which acts upon the rate of change of the error signal. Adding this term changes the controller equation as follows:

Out = Kp(e + Ki∫e dt +Kd de/dt)

For processes with extremely long reaction times, derivative control could be helpful in reducing overshoot. *Kd* is entered in seconds. As mentioned before, it is seldom used because tuning a control loop with three parameters can be challenging.

There is one more parameter called *Max Delta*. This value limits the output slew rate by restricting the output change each time the control loop is recalculated by the amount entered. This parameter will dramatically reduce the response time of the control loop.

Ramp — generates a repeating triangular or saw tooth wave with a float output

Ramp	
func::Ramp	
Out	77.54
Min	0.0
Max	100.0
Period	10
RampType	triangle

There are four configurable float parameters — *Min, Max, Period* and *Ramp Type*. For every scan cycle, the output increments by one unit until the output equals the *Max* value at which time it decrements until *Min* is reached. The result is a triangular wave with limits of *Max* and *Min* if *Ramp Type* is set for triangle. If *Ramp Type* is set for saw tooth, then the output will immediately drop to *Min* when *Max* is reached. The *Period* of the ramp is adjustable.

Set/Reset Latch — single-bit edge-triggered data storage

SRLatch	
func::SRLatch	
Out	false
S	false
R	false

The following logic applies on the false-to-true transition of S or R:

If S goes true and R does not change, then Out = true and remains true.
If R goes true and S does not change, then Out = false and remains false.
If both S and R go true on the same scan, then Out = false and remains false.

Ticking clock — an astable oscillator used as a time base

TickToc	
func::TickTock	
Out	true
TicksPerSec	1

There is one configurable float parameter — *Ticks Per Sec* — which can range from a low of 1 to a high of 10 pulses per sec.

Out = a periodic wave between 1 and 10 Hz

Float counter — up/down counter with float output

UpDn	
func::UpDn	
Out	0.0
Ovr	false
In	false
Rst	false
CDwn	false
Limit	0.0
HoldAtLimit	false

The counter range is between zero and a value that can be set with configurable parameter *Limit*. To cease counting at the limit set the configurable parameter Hold at *Limit* to true. To count down instead of up, set C *Dwn* to true. To reset the counter to zero set *Rst* to true. *Ovr* is the overflow indicator. *In* is the Boolean count input.

Out = the current count

If Out ≥ Limit, then Ovr is true

2.4   HVAC Kit   (hvac)

LSeq	
hvac::LSeq	
In	0.0
InMin	0.0
InMax	100.0
NumOuts	16
Delta	5.88
DOn	0
Out1	false
Out2	false
Out3	false
Out4	false
Out5	false
Out6	false
Out7	false
Out8	false
Out9	false
Out10	false
Out11	false
Out12	false
Out13	false
Out14	false
Out15	false
Out16	false
Ovfl	false

Linear Sequencer — bar graph representation of input value

There are two internally configurable floats called In *Min* and In *Max* that set the range of input values. An internal configurable integer — called *Num Outs* — specifies the intended number of active outputs. By dividing the input range by one more than the number of active outputs, the *Delta* between outputs is determined. Outputs will turn on sequentially from *Out1* to *Out16* within the input range as a function of increasing input value.

For example: In *Min* is set to 0, In *Max* to 100, and *Num Outs* is set to 9. This would give a *Delta* of 10. The following is true for increasing values of the input:

If In = 9 then Out1–16 are false and D On is zero.
If In = 70 then Out1–7 are true and Out8–16 are false. D On is 7.
If In = 101 then Out1–9 are true and Out10–16 are false. D On is 9 and Ovfl is true.

Note that for decreasing values of the input, the outputs will change by a value of Delta/2 below the input values stated above.

ReheatS	
hvac::ReheatSeq	
Out1	false
Out2	false
Out3	false
Out4	false
In	0.0
Enable	false
DOn	0
Hysteresis	0.0
Threshold1	0.0
Threshold2	0.0
Threshold3	0.0
Threshold4	0.0

Reheat Sequence — linear sequence up to four outputs

There are four configurable threshold points — Threshold1 through Threshold4 — that determine when a corresponding output will become true as follows:

Out1 = true when In ≥ Threshold1
Out2 = true when In ≥ Threshold2
Out3 = true when In ≥ Threshold3
Out4 = true when In ≥ Threshold4

These outputs will remain true until the input value falls below the corresponding threshold value by an amount greater than the configurable parameter *Hysteresis*. Output signal *D On* indicates how many outputs are true. Configurable parameter *Enable* must be true otherwise all outputs will be false.

Reset	
hvac::Reset	
Out	0.0
In	0.0
InMin	0.0
InMax	4095.0
OutMin	0.0
OutMax	100.0

Reset — output scales an input range between two limits

There are four configurable float parameters — In *Max, In Min, Out Max* and *Out Min* — which determine the input and output ranges respectively of the input and output. The output of this component will scale linearly with the value of the input if the input is within the input range. The input range (IR) is determined by *In Max-In Min* while the output range (OR) is determined by *Out Max-Out Min*. If the input is within the input range, then the following is true:

Out = (In + In Min)(OR/IR) + Out Min
If the input exceeds In *Max* then *Out = Out Max.*
If the input is less than In *Min* then *Out = Out Min*

Tstat	
hvac::Tstat	
Diff	0.0
IsHeating	false
Sp	1.0
Cv	0.0
Out	false
Raise	true
Lower	false

Thermostat — on/off temperature controller

The configurable float parameter — *Diff* — provides hysteresis and deadband. Another configurable parameter — *Is Heating* — indicates a heating application. *Sp* is the *setpoint* input and Cv is the *controlled* variable input. *Raise* and *lower* are outputs.

If *Cv > (Sp + Diff/2)* then *Lower* is true and will remain true until *Cv < Sp*
If *Cv < (Sp – Diff/2)* then *Raise* is true and will remain true until *Cv > Sp*

If *Is* Heating is false, then Out = Lower
If *Is* Heating is true, then Out = Raise

2.5   Logic Kit   (logic)

ADemux2	
logic::ADemux2	
Out1	0.0
Out2	0.0
In	0.0
S1	false

Analog Demux — Single-input, two-output analog de-multiplexer

If S1 is false then Out1 = In while Out2 = the last value of In just before S1 changed.

If S1 is true then Out2 = In while Out1 = the last value of In just before S1 changed.

And2	
logic::And2	
Out	false
In1	false
In2	false

Two-input Boolean product — two-input AND gate

Out = In1 • In2

And4	
logic::And4	
Out	false
In1	false
In2	false
In3	false
In4	false

Four-input Boolean product — four-input AND gate

Out = In1 • In2 • In3 • In4

ASW
logic::ASW

Out	0.0
In1	0.0
In2	0.0
S1	false

Analog switch — selection between two float variables

If S1 is false, then Out = In1

If S1 is true, then Out = In2

ASW4
logic::ASW4

Out	0.0
In1	0.0
In2	0.0
In3	0.0
In4	0.0
StartsAt	0
Sel	0

Analog switch — selection between four floats

Configurable integer parameter Starts At sets the base selection.

If integer Sel <= Starts At then Out = In1
If integer Sel = Starts At + 1 then Out = In2
If integer Sel = Starts At + 2 then Out = In3
If integer Sel = Starts At + 3 then Out = In4
For all values of Sel that are 4 greater than Starts At then Out = In4

B2P
logic::B2P

Out	false
In	false

Binary to pulse — simple mono-stable oscillator (single-shot)

Out = true for one scan on the raising edge of In

BSW
logic::BSW

Out	false
In1	false
In2	false
S1	false

Boolean Switch — selection between two Boolean variables

If S1 is false, then Out = In1
If S1 is true, then Out = In2

DemuxI2
logic::DemuxI2B4

In	0
Out1	true
Out2	false
Out3	false
Out4	false
StartsAt	0

Four-output Demux — integer to Boolean de-multiplexer

If In = StartAt + 0 then Out1 is true, else false
If In = StartAt + 1 then Out2 is true, else false
If In = StartAt + 2 then Out3 is true, else false
If In = StartAt + 3 then Out4 is true, else false

ISW
logic::ISW

Out	0
In1	0
In2	0
S1	false

Integer switch — selection between two integer variables

If S1 is false, then Out = In1
If S1 is true, then Out = In2

Not	!
logic::Not	
Out	true
In	false

Not — inverts the state of a Boolean

$Out = \overline{In}$

Or2	II
logic::Or2	
Out	false
In1	false
In2	false

Two-input Boolean sum — two-input OR gate

$Out = In1 \mid In2$

Or4	II
logic::Or4	
Out	false
In1	false
In2	false
In3	false
In4	false

Four-input Boolean sum — four-input OR gate

$Out = In1 \mid In2 \mid In3 \mid In4$

Xor	⊕
logic::Xor	
Out	false
In1	false
In2	false

Two-input exclusive Boolean sum — two-input XOR gate

$Out = In1 \oplus In2 = \overline{In1} \bullet In2 \mid In1 \bullet \overline{In2}$

2.6   Math Kit   (math)

Add2	+
math::Add2	
Out	0.0
In1	0.0
In2	0.0

Two-input addition — results in the addition of two floats

$Out = In1 + In2$

Add4	+
math::Add4	
Out	0.0
In1	0.0
In2	0.0
In3	0.0
In4	0.0

Four-input addition — results in the addition of four floats

$Out = In1 + In2 + In3 + In4$

Avg10	⋀
math::Avg10	
Out	null
In	0.0
MaxTime	0

Average of 10 — sums the last ten floats while dividing by ten thereby providing a running average

Out = (sum of the last ten values)/ten

The float input *In* is sampled once every scan and stored. If the input does not change in value on the next scan, it is not sampled again — unless sufficient time passes that exceeds the internal integer *Max Time* with units of milliseconds. In this instance the input is sampled and treated as another value. Once ten readings occur, the average reading is outputted.

AvgN	⋀
math::AvgN	
Out	0.0
In	0.0
NumSamplesToAvg	5
Reset	false

Average of N — sums the last N floats while dividing by N thereby providing a running average

Out = (sum of the last N values)/N

The float input *In* is sampled once every scan and stored regardless whether or not the value changes. Once *Num Samples* to *Avg* readings occur, the average reading is outputted.

Div2	✛
math::Div2	
Out	0.0
In1	0.0
In2	0.0
Div0	true

Divide two — results in the division of two floats

Out = In1/In2
Div0 = true if In2 is equal to zero

FloatOf	✛
math::FloatOffset	
Out	0.0
In	0.0
Offset	0.0

Float offset — float shifted by a fixed amount

Out = In + Offset

Offset is a configurable float.

Max	✎
math::Max	
Out	0.0
In1	0.0
In2	0.0

Maximum selector — selects the greater of two inputs

Out = Max [In1, In2] where Out, In1 and In2 are floats

Min	✎
math::Min	
Out	0.0
In1	0.0
In2	0.0

Minimum selector — selects the lesser of two inputs

Out = Min [In1, In2] where Out, In1 and In2 are floats

MinMax
math::MinMax

MinOut	0.0
MaxOut	0.0
In	0.0
R	false

Min/Max detector — records both the maximum and minimum values of a float

Min Out = Max [In] if R is false
Max Out = Min [In] if R is false
If R is true, then Min Out and Max Out = In
Both *Min Out* and *Max Out* are floats — as is *In*.
It may be necessary to reset the component after connecting links to the component.

Mul2
math::Mul2

Out	0.0
In1	0.0
In2	0.0

Multiply two — results in the multiplication of two floats

Out = In1 ∗ In2

Mul4
math::Mul4

Out	0.0
In1	0.0
In2	0.0
In3	0.0
In4	0.0

Multiply four — results in the multiplication of four floats

Out = In1 ∗ In2 ∗ In3 ∗ In4

Not
logic::Not

Out	true
In	false

Negate — changes the sign of a float

Out = − In

Round
math::Round

Out	0
In	0
DecimalPlaces	0

Round — rounds a float to the nearest N places

For N = -1, Out = In rounded to the nearest tens
For N = 0, Out = In rounded to the nearest units
For N = 1, Out = In rounded to the nearest tenth's
For N = 2, Out = In rounded to the nearest hundredths
For N = 3, Out = In rounded to the nearest thousandths

For positive input values, the output will round up (more positive). For negative input values, the output will round down (more negative).

Sub2
math::Sub2

Out	0.0
In1	0.0
In2	0.0

Subtract two — results in the subtraction of two floats

Out = In1 − In2

Sub4	▬
math::Sub4	
Out	0.0
In1	0.0
In2	0.0
In3	0.0
In4	0.0

Subtract four — results in the subtraction of four floats

Out = In1 – In2 – In3 – In4

TimeAvg	⏧
math::TimeAvg	
Out	0.0
In	0.0
Time	10000

Time Average — the average of a float over a determined time

Out = Avg[In] over the integer time in milliseconds.

2.7   Priority Kit   (pricomp)

Priorit	☐
pricomp::PrioritizedBool	
SourceLevel	fallback
OverrideExpTime	0
In1	null
In2	null
In3	null
In4	null
In5	null
In6	null
In7	null
In8	null
In9	null
In10	null
In11	null
In12	null
In13	null
In14	null
In15	null
In16	null
Fallback	null
Out	null
MinActiveTime	0
MinInactiveTime	0

Priority array (Priorit) components exist for Boolean, float and integer variables. Up to 16 levels of priority from In1 to In16 can be assigned. In1 has the highest priority and In16 the lowest. With few exceptions, all can be pinned out. If a priority level is not assigned, it is marked as a Null and therefore ignored. If a Null is inputted to the priority array, the priority array will ignore it and choose the next input in line. The Boolean version of the array has two timer settings — one for minimum active time and one for minimum inactive time. If the highest priority device changes from false to true and then back to false, the priority component will maintain the event for the configured times.

There is a Fallback setting in each array that can be specified. If no valid priority input exists, the Fallback value is transferred to the output.

Priori1	■
pricomp::PrioritizedFloat	
SourceLevel	fallback
OverrideExpTime	0
In1	null
In2	null
In3	null
In4	null
In5	null
In6	null
In7	null
In8	null
In9	null
In10	null
In11	null
In12	null
In13	null
In14	null
In15	null
In16	null
Fallback	null
Out	null

Priori2	■
pricomp::PrioritizedInt	
SourceLevel	fallback
OverrideExpTime	0
In1	min
In2	min
In3	min
In4	min
In5	min
In6	min
In7	min
In8	min
In9	min
In10	min
In11	min
In12	min
In13	min
In14	min
In15	min
In16	min
Fallback	min
Out	min

2.8   Timing Kit   (timing)

DlyOff	⬁
timing::DlyOff	
Out	false
In	false
DelayTime	0.0
Hold	0

Off delay timer — time delay from a true to false transition of the input

For input transitions from false to true, Out = true.

For input transitions from true to false that exceed the Delay Time, Out = false after the delay time.

Hold is a read-only integer that counts down the time. Delay time is in seconds.

DlyOn	⬁
timing::DlyOn	
Out	false
In	false
DelayTime	0.0
Hold	0

On delay timer — time delay from a false to true transition of the input

For input transitions from true to false, Out = false.

For input transitions from false to true that exceed the Delay Time, Out = true after the delay time.

Hold is a read-only integer that counts down the time. Delay Time is in seconds.

OneShot	⬁
timing::OneShot	
Out	false
In	false
PulseWidth	0.0
CanRetrig	false

Single Shot — provides an adjustable pulse width to an input transition

Upon the input transitioning to true, the output will pulse true for the amount of time set in the configurable parameter *Pulse Width*. Time is in seconds. If the configurable parameter *Can Retrig* is set to true, the component will repeat its action on every positive transition of the input. For example, in retrigger mode, a one-second *TickToc* connected to a *OneShot* with a 2 second pulse width setting will have the *OneShot* output in a continuous true state due to constant retriggering at a rate faster than the *OneShot* pulse width.

Timer	⊓
timing::Timer	
Out	false
Run	stop
Time	0
Left	0

Timed pulse — predefined pulse output

Out becomes true for a predetermined time when Run transitions from false to true. If Run returns to false, then Out becomes false.

Time determines the amount of time the output will be on in seconds.

2.9   Types Kit   (types)

B2F	⊡
types::B2F	
Out	0.0
Count	0.0
In1	false
In2	false
In3	false
In4	false
In5	false
In6	false
In7	false
In8	false
In9	false
In10	false
In11	false
In12	false
In13	false
In14	false
In15	false
In16	false

Binary to float encoder — 16-bit binary to float conversion

Out = encoded value of binary input with In16 being the MSB and In1 being the LSB

Count = sum of the number of active inputs

ConstBo	●
types::ConstBool	
Out	false

Boolean Constant — a predefined Boolean value

Out = a Boolean value that is internally configurable

ConstFl	●
types::ConstFloat	
Out	0.0

Float Constant — a predefined float value

Out = a float value that is internally configurable

ConstIn	●
types::ConstInt	
Out	0

Integer Constant — a predefined integer value

Out = an integer value that is internally configurable

F2B	⤳
types::F2B	
In	0.0
Out1	false
Out2	false
Out3	false
Out4	false
Out5	false
Out6	false
Out7	false
Out8	false
Out9	false
Out10	false
Out11	false
Out12	false
Out13	false
Out14	false
Out15	false
Out16	false
Ovrf	false

Float to binary decoder — float to 16-bit binary conversion

Out1 to Out16 = the 16-bit decoded value of In — with Out16 representing the MSB and Out1 representing the LSB

Ovrf = true when In > 65535

Although the input requires a float, fractional amounts are ignored during the conversion.

F2I	⤳
types::F2I	
In	0.0
Out	0

Float to integer — float to integer conversion

Out = In except that the output will be a whole number

The fractional amount of the float input will be truncated at the output.

I2F	⤳
types::I2F	
In	0
Out	0.0

Integer to Float — integer to float conversion

Out = In except that the output will become a float

L2F	⇌
types::L2F	
In	0
Out	0.0

Long to Float — 64-bit signed integer to float conversion

Out = In except that the output will become a float from a 64-bit signed integer

WriteBo
types::WriteBool

In	false
Out	false

Write Boolean — setting a writable Boolean value

Out = In

Unlike ConstBo, this component has an input. Could be helpful when transferring a variable between two wiresheets.

WriteFl
types::WriteFloat

In	0.0
Out	0.0

Write Float — setting a writable float value

Out = In

Unlike ConstFl, this component has an input. Could be helpful when transferring a variable between two wiresheets.

WriteIn
types::WriteInt

In	0
Out	0

Write Integer — setting an integer value

Out = In

Unlike ConstIn, this component has an input. Could be helpful when transferring a variable between two wiresheets.

2.10   Sys Kit   (sys)

Folder
sys::Folder

Folder — gateway to a wiresheet

Folders allow for the segregation of wiresheet logic. To create an additional wiresheet from the main wiresheet, a folder must be placed on the main wiresheet. If cascading wiresheets are desired, then a folder must be placed on the wiresheet from every preceding wiresheet. By segregating wiresheets, Sedona tags can be reused as long as there are no duplicates on the same wiresheet.

RateFol
sys::RateFolder

AppCyclesToSkip	0

Rate Folder — gateway to a wiresheet with delayed execution

The Rate Folder has the same attributes of a Folder except in terms of execution. With a Folder, the logic within the folder is executed once per scan cycle just like any other logic in the application. However, with a Rate Folder the execution of the Rate Folder can by skipped by configuring the AppCycleToSkip slot. For example:

If AppCycleToSkip = 1, then the execution of the logic within the Rate Folder will only occur every other scan cycle.

3   Sedona Core Components

This tutorial assists in the understanding of the Sedona core components provided in Tridium's Sedona-1.2.28 release. New with the 1.2 release is that the Sedona components, previously concentrated in one Control kit, are now organized in smaller kits under a functional name. Components discussed in this document can be found in the following kits:

- basicSchedule
- datetimeSTD
- func
- hvac
- logic
- math
- pricomp
- timing
- types
- sys

Sedona components are deployed in kits of which there are three types – core components, custom platform-independent, and custom platform-dependent. This section addresses the core components which can be found in any Sedona application. By core components we are referring to Tridium release 1.2.28 components. Custom components deployed in custom kits are provided by developers from the Sedona community. It is up to the developer to explain how their custom components function. They are not explained here. However, the techniques found in this tutorial can be applied to any Sedona project.

The intent of this tutorial is not to teach how to create HVAC applications using Sedona because Sedona can be used anywhere where control of processes is required. It is only intended to introduce the basic Sedona components which can be assembled onto a wiresheet by integrators thereby creating applications.

This tutorial was conducted using Sedona Application Editor as the Sedona tool, so the appearance of the components and the configuration of parameters are unique to this tool. Other Sedona tools create a somewhat different appearance of the components and use different methods for configuration. However, all the examples in the tutorial can be implemented with any Sedona tool.

3.1   Understanding the Wiresheet Structure
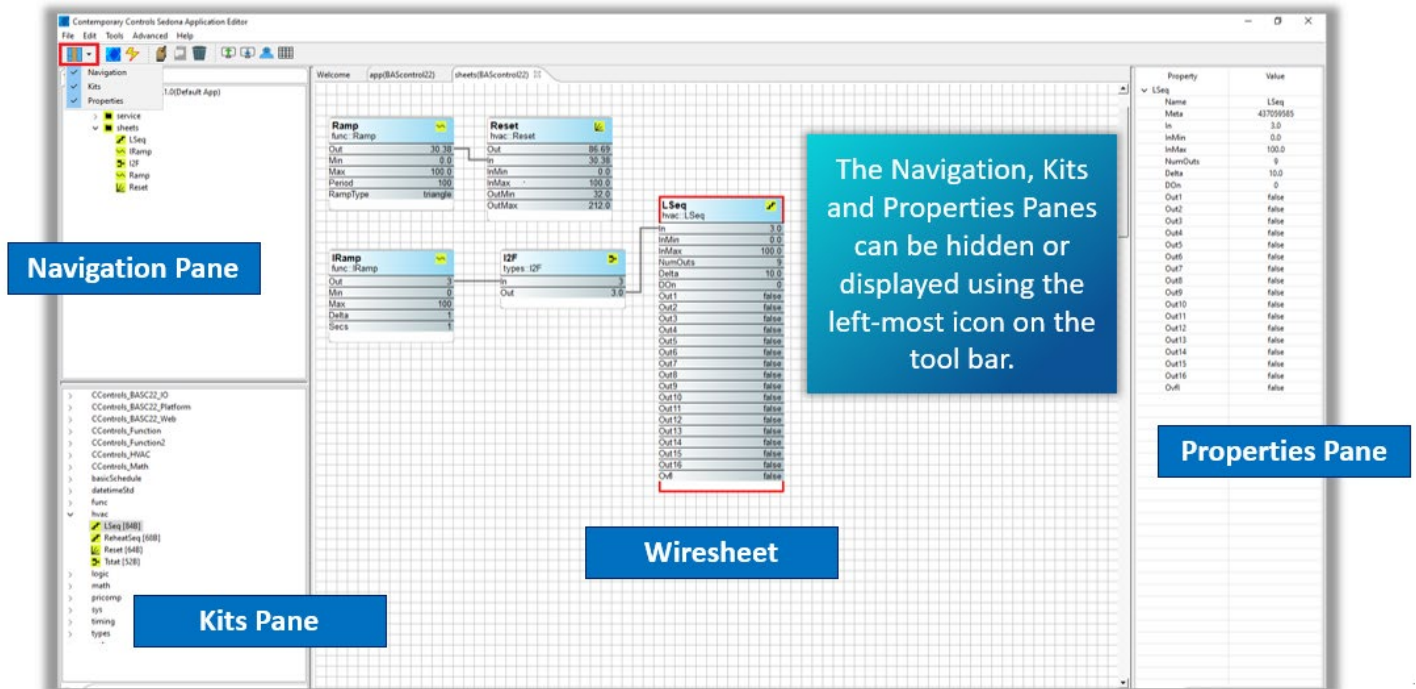
3.2   Adding a Component to the Wiresheet

To add a component, expand the kit, "hvac" in this example. Select the component from the kit. Then, drag and drop the component onto the wiresheet.
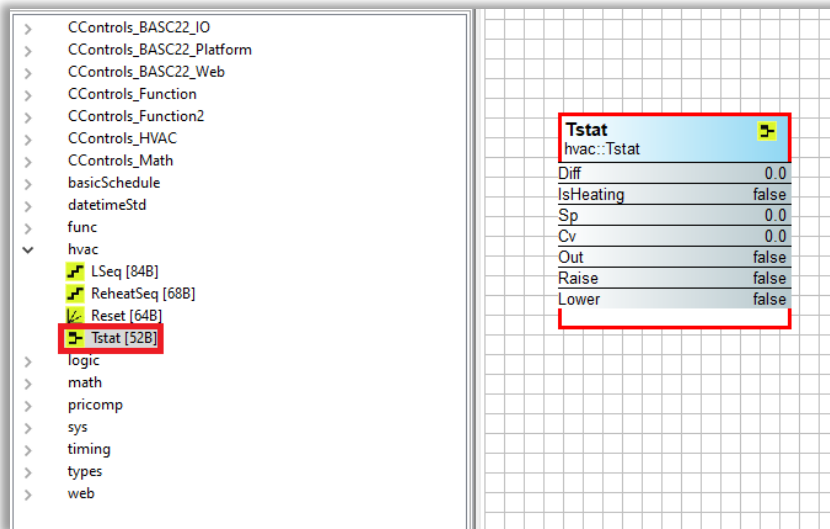
Property	Value
∨ Tstat	
Name	Tstat
Meta	52101121
Diff	0.0
IsHeating	false
Sp	0.0
Cv	0.0
Out	false
Raise	false
Lower	false

Properties / Links | Slots

Click the component (it will highlight with a red border), to view and update its Properties, Links and Slots in the Properties Pane. The Properties tab shows you the Properties and Values for that component. The Links tab shows you links to other components. The Slots tab shows you the variable Type and Facets for each slot.

3.3   Renaming a Component

To rename a component, select the component. Click the Name field in the Properties Pane, and type in the new name, "Heat1" in this example.

Tstat
hvac::Tstat

Diff	0.0
IsHeating	false
Sp	0.0
Cv	0.0
Out	false
Raise	false
Lower	false

Property	Value
∨ Tstat	
Name	Heat1
Meta	788660225
Diff	0.0
IsHeating	false
Sp	0.0
Cv	0.0
Out	false
Raise	false
Lower	false

Note: SAE prevents duplicate names, and names can only contain seven English characters a-z (upper and lower case) and numbers 0-9. The name cannot begin with a number or use special characters.

Press return, and the name changes.
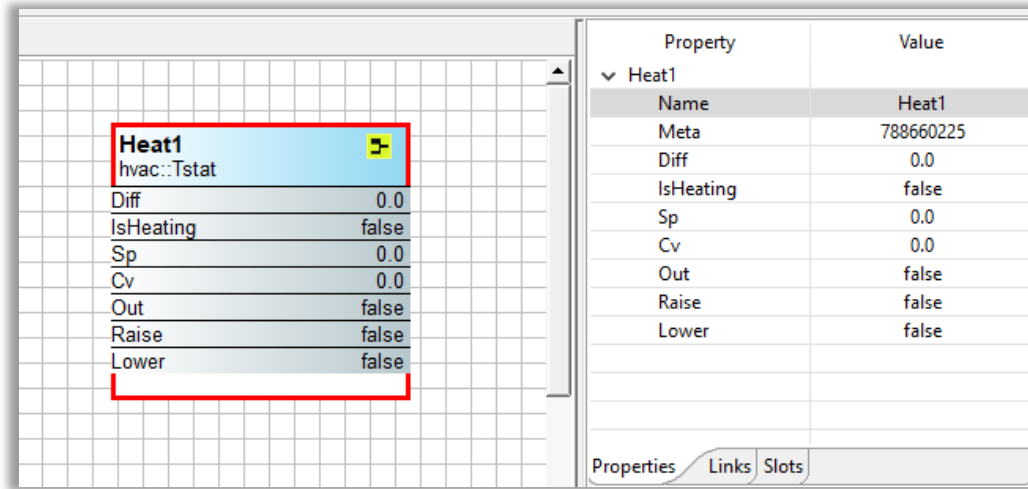
3.4 Creating a Blank Wiresheet

Within the sys kit, the Folder component allows for the segregation of wiresheet logic, allowing you to create an additional wiresheet from the main wiresheet. To create a blank wiresheet, drag the Folder components onto the main wiresheet.
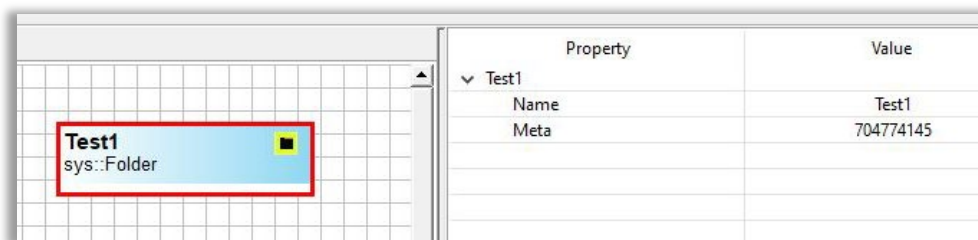
Rename the component in the Properties Pane to "Test1," and press return.

Double-click the component, and a new blank wiresheet appears which shows the contents of your new folder. At the top of the wiresheet is a new tab with the name of the "Test1" folder.
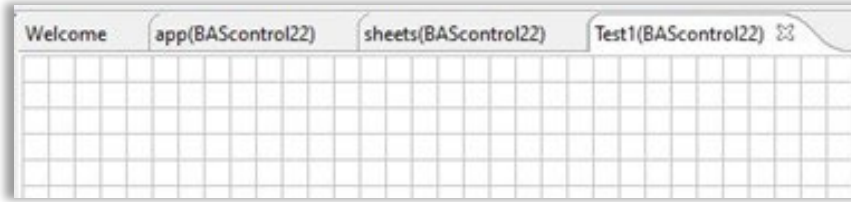
The Rate Folder has the same attributes of a Folder except in terms of execution. With a Folder, the logic within the folder is executed once per scan cycle just like any other logic in the application. However, with a Rate Folder the execution of the Rate Folder can by skipped by configuring the AppCycleToSkip slot.
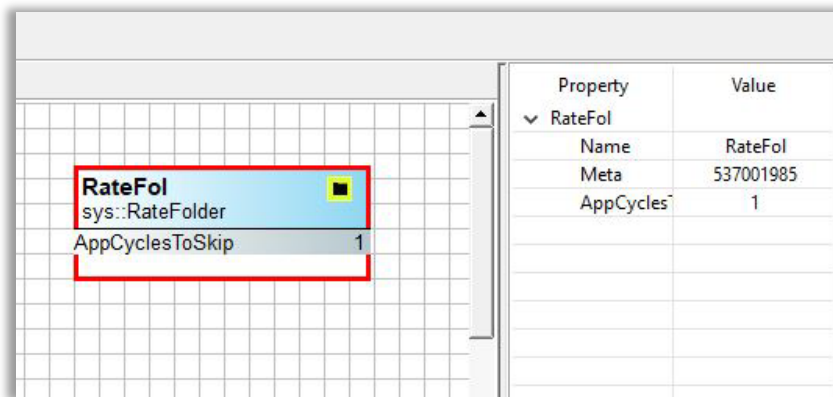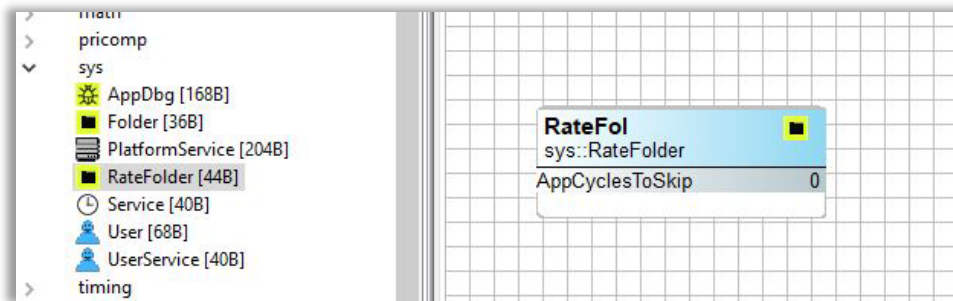
For example:

If AppCycleToSkip = 1, then the execution of the logic within the Rate Folder will only occur every other scan cycle.

3.5   Updating a Configurable Component Value

To update a component value, click the Slots tab and review the Facets column. Slots with the Facet [config] can be configured from the default value. Ignore the meta slot.

Return to the Properties tab. Set your new value. In this example, Diff = 5. Save your application before closing your program, otherwise the Value will return to its default state.

3.6   Saving Your Wiresheet

To save your application, click the "Save to Controller" icon on the toolbar. Then, click "OK."
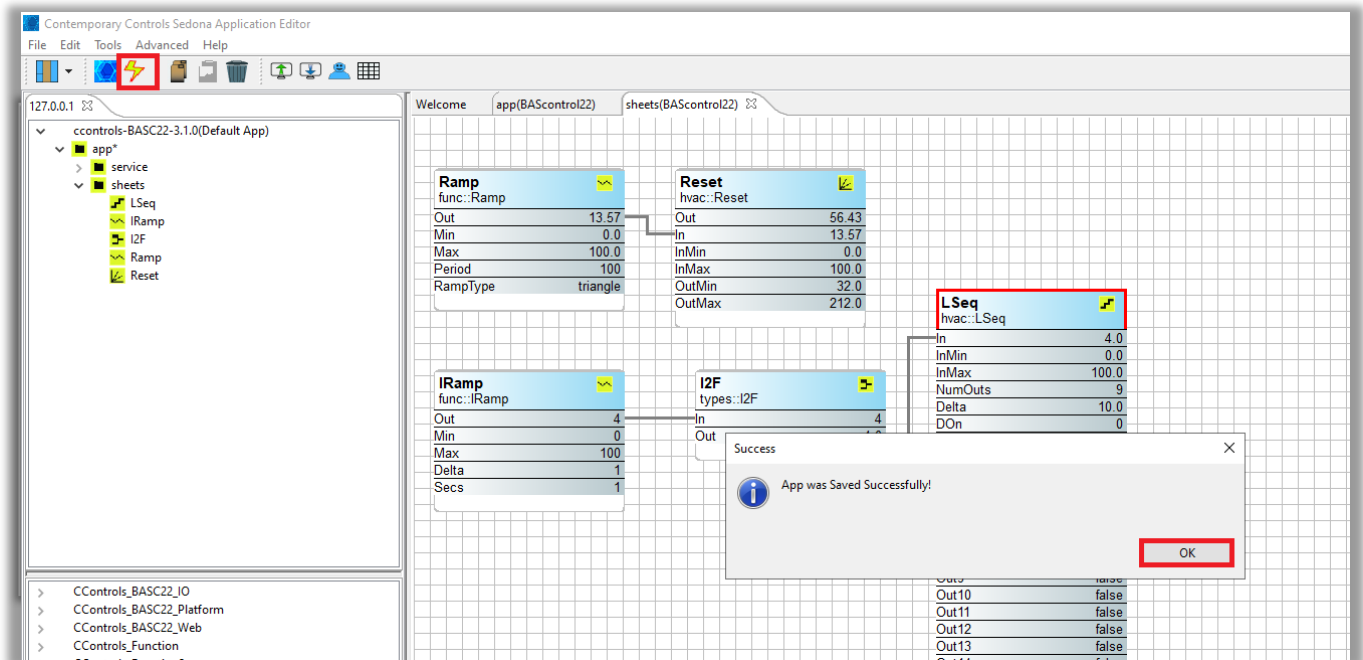
The saved wiresheet will be stored in the emulator, and your program will be there the next time you launch it. Without saving, your work will be lost.

You can also save your wiresheet to your PC. You can use the BAScontrol Project Utility program (BASbackup) to save a Sedona wiresheet and device configuration together as a single BAScontrol project file and restore it to a real or emulated controller. To summarize:

 Save to Controller from the SAE: Saves a Sedona binary application file (app.sab.target) to an emulator or a real controller. A SAB file is only a machine-readable executable file.

Save to PC/Load from PC from the SAE: Saves a Sedona source application file (app.sax) to your PC. A SAX file (also referred to as an application or App file) is human readable. When saving, you are required to provide a name for your file. Similarly, "Load from PC" uploads a SAX file from your PC into SAE.

 Backup/Restore from BASbackup: Saves configuration files specific to the BAScontrol, BASpi, BASioT, or RTU controller used, including all the non-Sedona configuration data, such as web page settings and IP address settings, to a single BAScontrol project file. When saving, you are required to provide a name for your file. The "Restore" function allows you to copy (clone) the project to a real or emulated controller.

3.7   Understanding Variable Types

Although there are several variable types used by Sedona, three are the most interesting — Boolean, Float, and Integer. You can define constants for each type and use converting components to change the data representation to a different type.
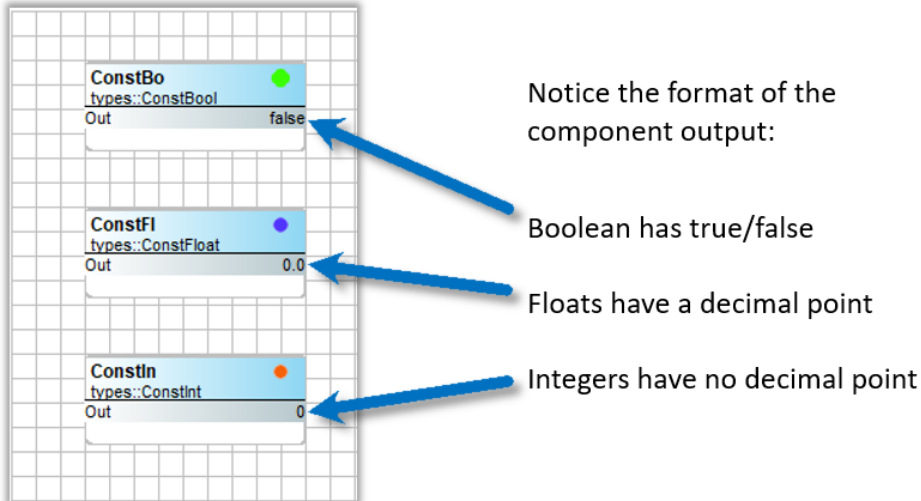
Notice the format of the component output:

Boolean has true/false

Floats have a decimal point

Integers have no decimal point

These are constant components that can be configured. However, they must be saved, or the settings will be lost.

3.8   Configuring Constants
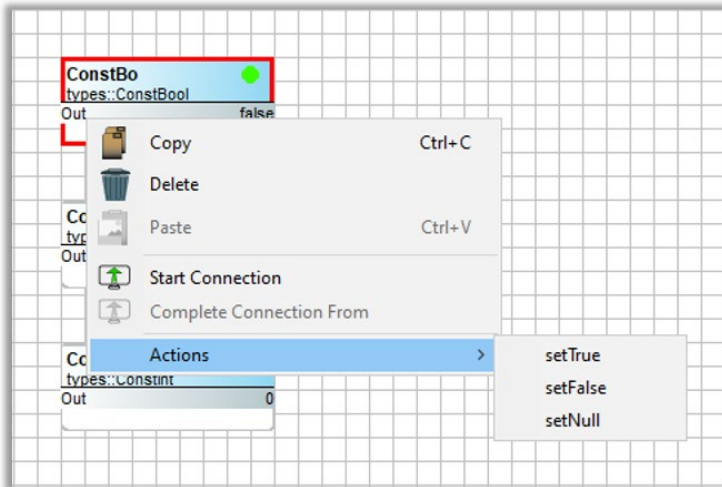
You can set the value of the constant by right-clicking on the component and the selecting Actions. For the ConstBool components, your choices are True, False or Null. Null is seldom used.

3.9   Using Write Components

WriteBo
types::WriteBool
In   false
Out   false

WriteFl
types::WriteFloat
In   0.0
Out   0.0

WriteIn
types::WriteInt
In   0
Out   0
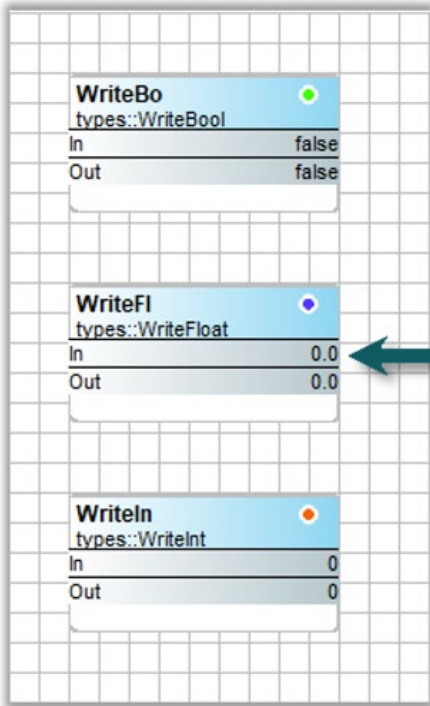
In a similar manner, there are write components for each variable type. Unlike the constant components, these write components have an input slot. The value of the input will be saved if the application program is saved. Other than the input slot difference, the constant components and the write components function the same.

3.10 Converting Between Component Types
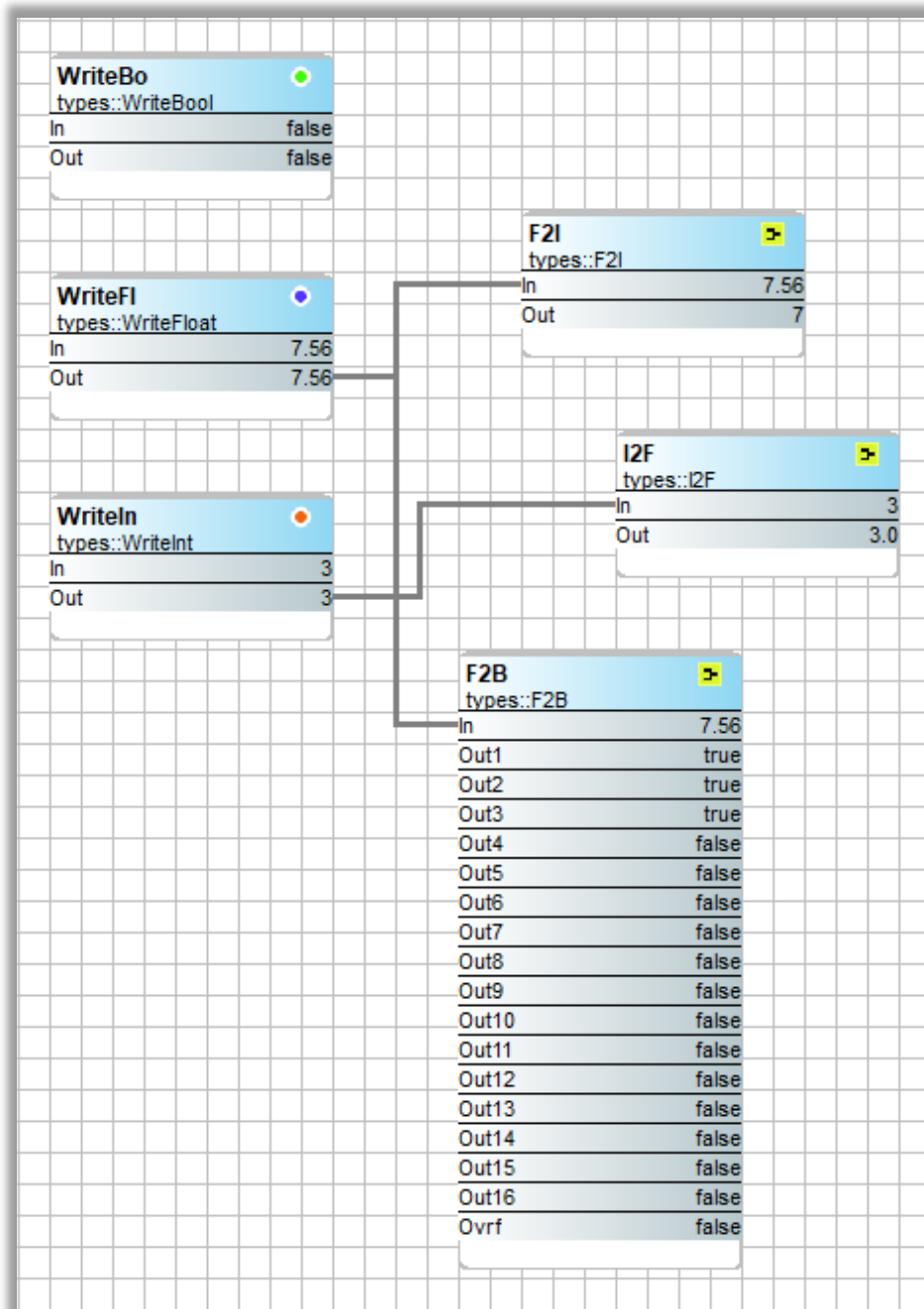
Float-to-Integer and Integer-to-Float components exist. Notice that when we converted from a float to an integer, the Float-to-Integer component truncated the original value during conversion.

Although it appears that an Integer-to-Float conversion created a much higher accuracy of the original value, this is not the case. The ability to convert variable types is necessary because not all Sedona components exist for each variable type.

You can also convert a float to a binary using the Float-to-Binary component. However, notice that the resulting 0000 0000 0000 0111 binary representation is actually a decimal 7 and again the original float value was truncated.

There are no Integer-to-Binary components, but this could be accommodated by using an Integer-to-Float ahead of the F2B component.

3.11 Converting from Float-to-Boolean and Boolean-to-Float
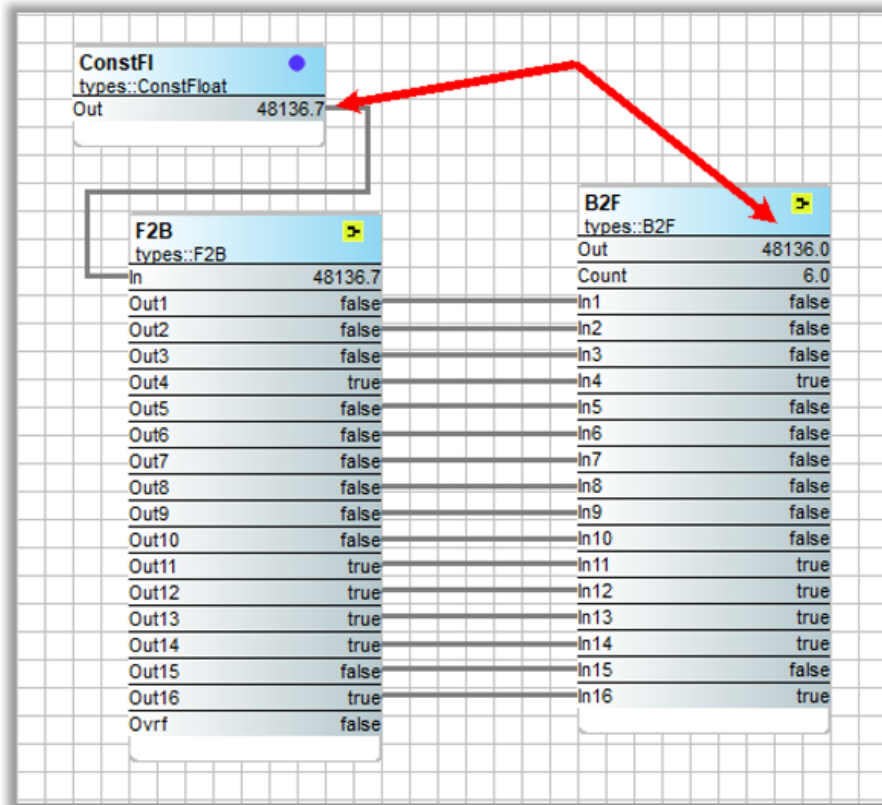
In this example, we will begin with a float with a value of 48138.7 and convert it to binary using a Float-to-Binary component, and then immediately convert it back into a float using a Binary-to-Float component.

Notice the recovered float values are truncated from their original value.
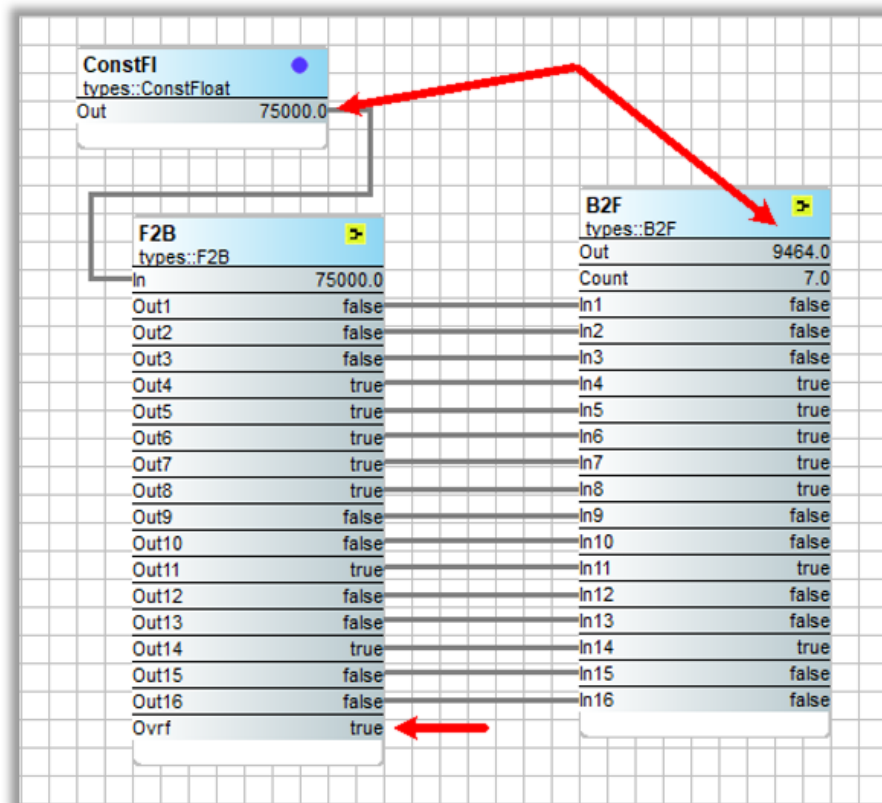
We increased the float value to 75000, but this time we have different answers. Because we are only doing a 16-bit conversion, we can only count up to 65535. Notice that the Ovrf pin is true, meaning there was a counter overflow. The Ovrf pin means that a value of 65536 or higher was detected. The remaining counter value, called the residue, represents a Modulo-65536 result of 9464. If you subtract 65536 from 75000, you will get 9464. It is important to monitor the Ovrf flag when doing float to binary conversion.

3.12 Negating a Boolean Variable — Inverting Your Logic

A Boolean can have either of two states – true or false. A true can be referred to as a logic 1 and a false as a logic 0.
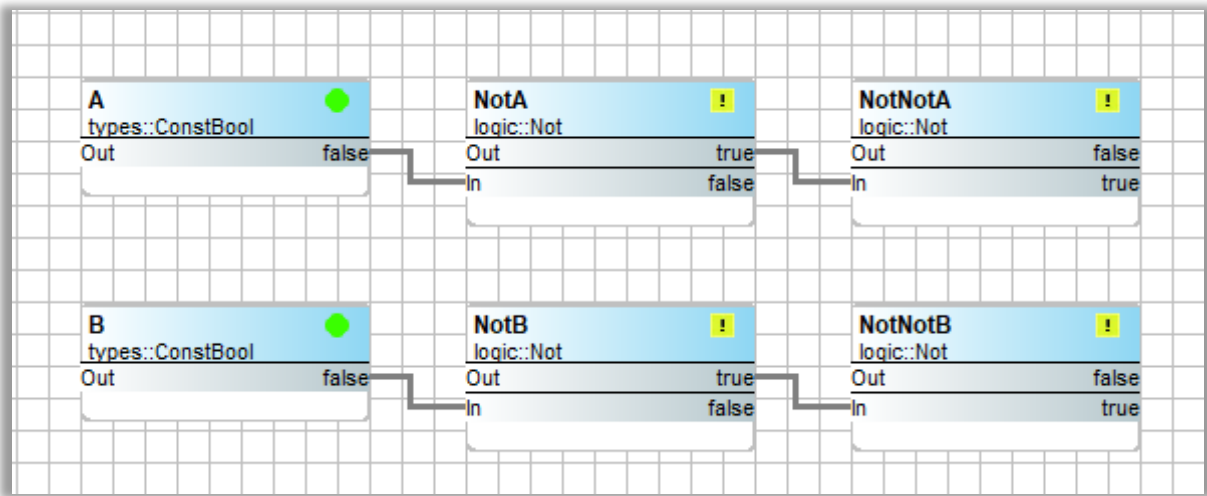
There are two Boolean variables A and B which are set to be false. Both feed a Not component that is usually called an inverter because it changes the initial variable to the opposite state, which it true. Going into another inverter changes the state back to the original states of A and B.
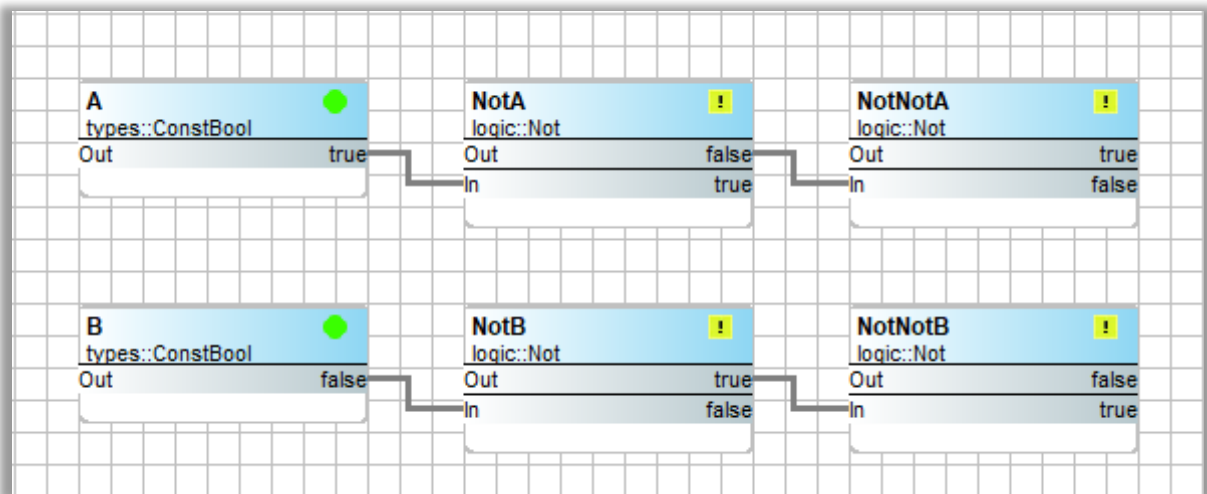
Variable A is now set to be true. Notice in the second panel the output of the first inverter changes the value of A to a false, while the second inverter restores the state of A back to true.

3.13 Boolean Product — "ANDing" Boolean Variables
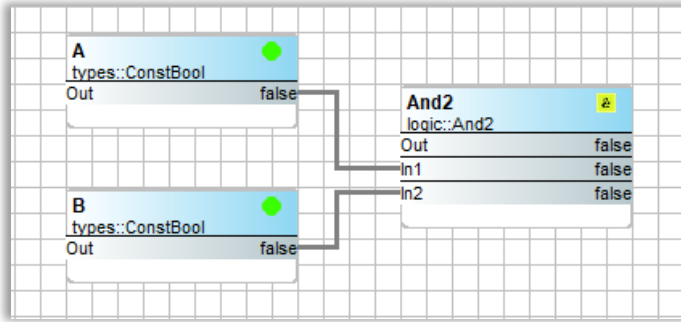
The AND component if frequently called an AND gate. If A is false and B is false, then the output is false.
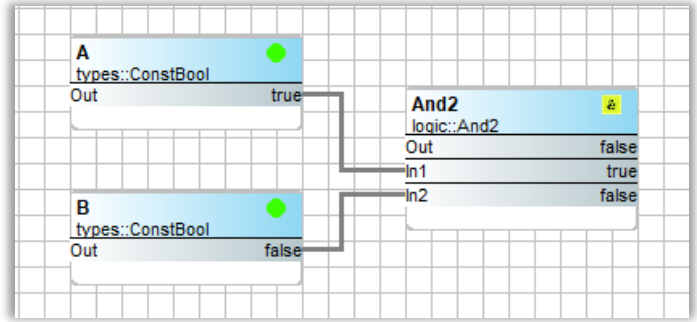
For an AND gate, If A is true and B is false, then the output is false.
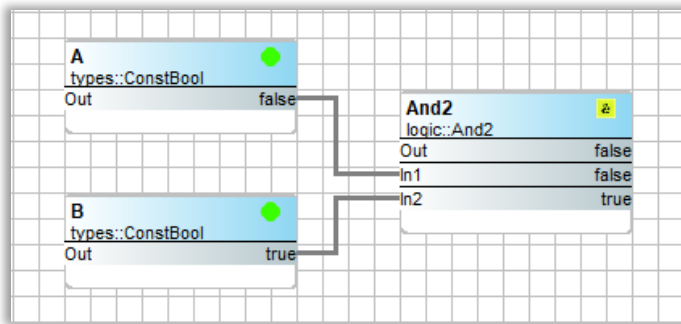
For an AND gate, If A is false and B is true, then the output is false.
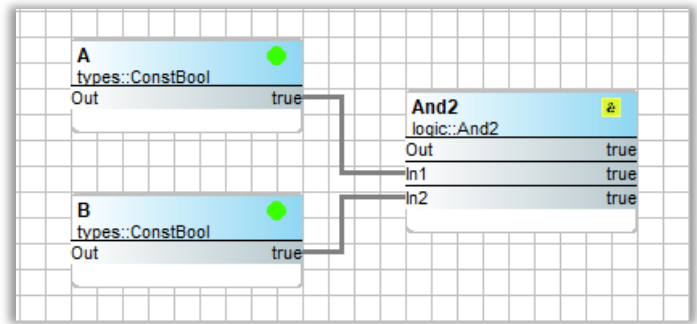
For an AND gate, If A is true and B is true, then the output is true.

3.14   Boolean Sum — "Oring" Boolean Variables
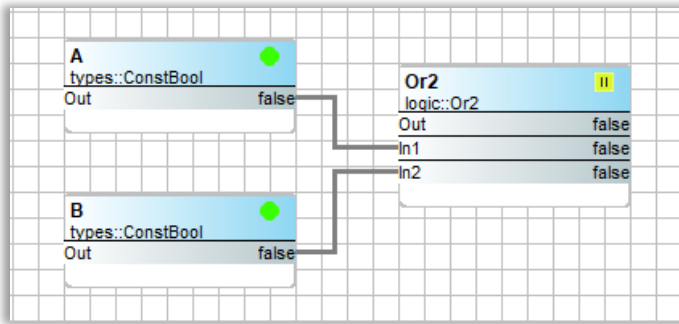
The OR component if frequently called an OR gate. If A is false and B is false, then the output is false.
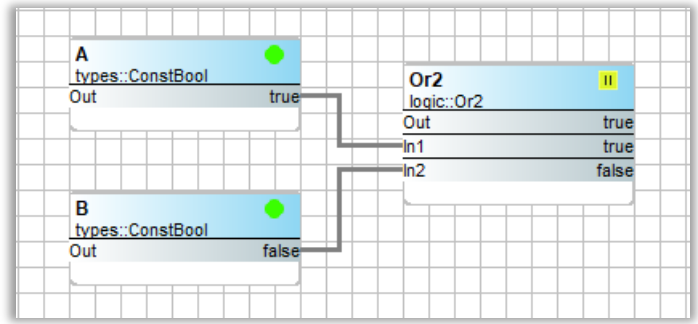
For an OR gate, If A is true and B is false, then the output is true.
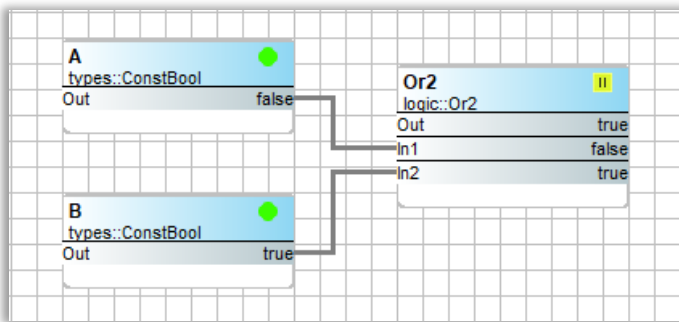
For an OR gate, If A is false and B is true, then the output is true.
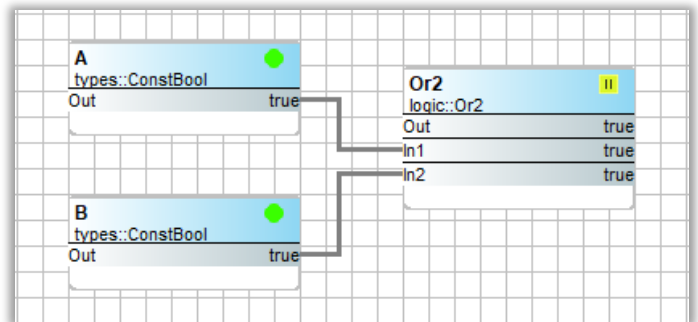
For an OR gate, If A is true and B is true, then the output is true.

3.15   Creating an Exclusive OR — A OR B, but Not Both
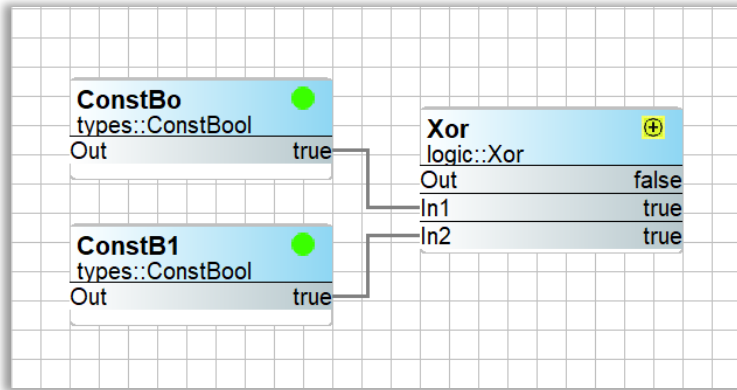
An Exclusive OR is very similar to an OR except for the condition when both inputs are true. In this case, the output is false. An XOR solves the problem of A or B, but not both.

3.16   Cascading Logic Blocks and Unused Inputs
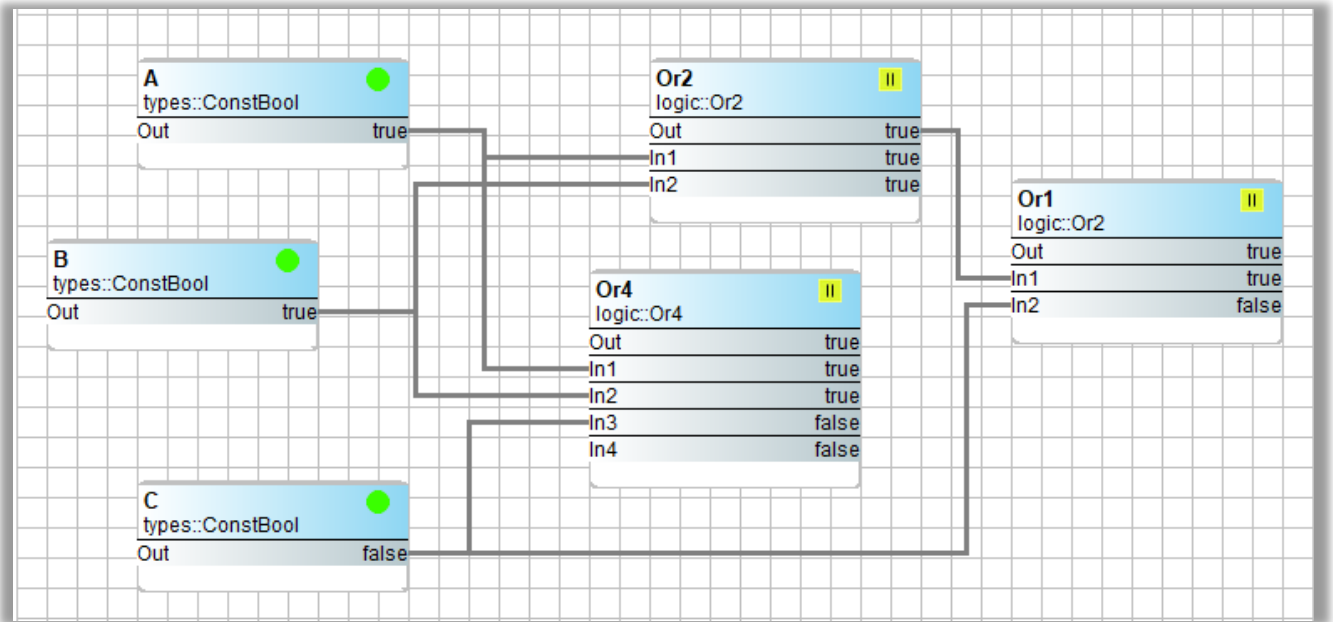
Four-input OR gates operate the same allowing more variables to be added to the logic. With OR gates, unused inputs can be left unconnected because unused inputs default to false.

Notice that two-input OR gates can be used with three variables by cascading two-input OR gates.
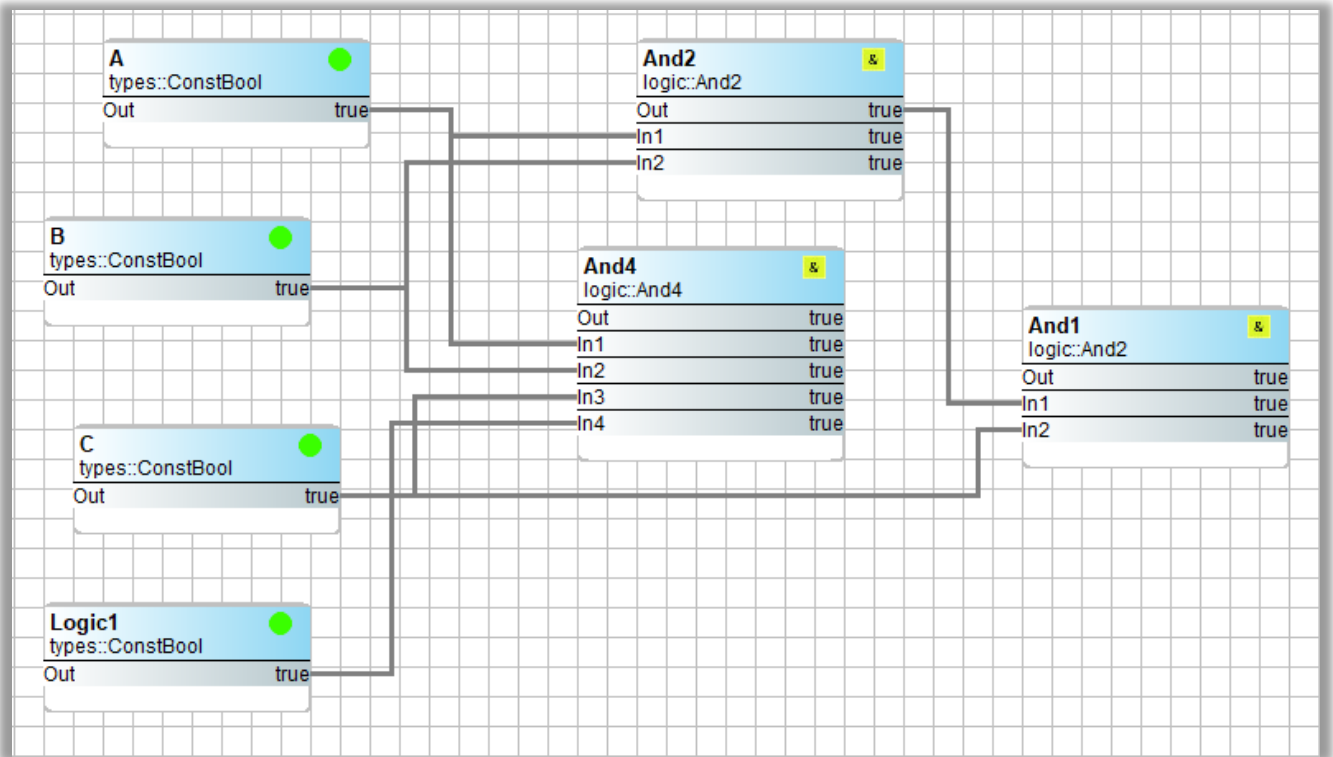
Four-input AND gates exists, but unused inputs must be accommodated by creating a Logic1 constant that is tied to all unused AND gate inputs, otherwise the AND gate outputs would be permanently disabled.

Cascading AND gates are also a possibility when using more than two variables.
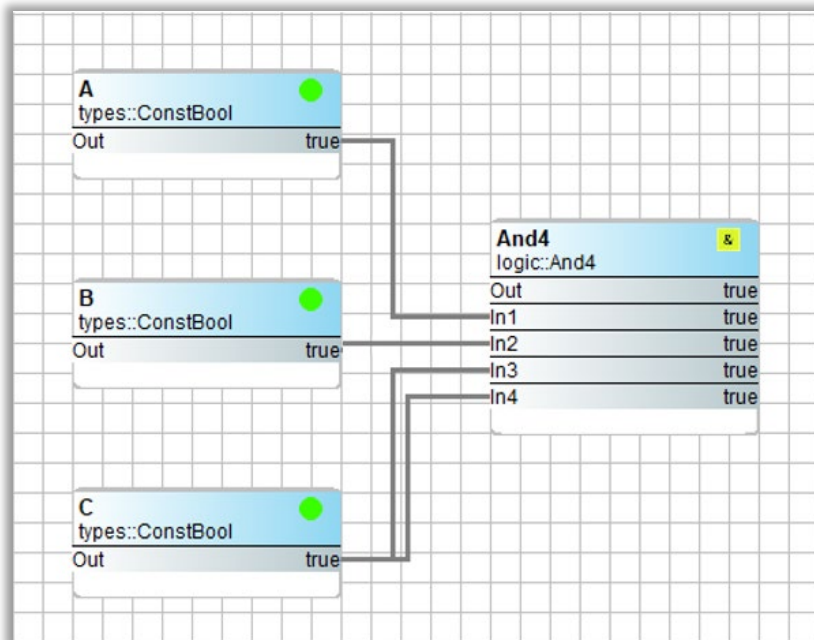
Here is another way of handling an unused input when three variables are connected to a four-input AND gate. Attach the unused input to one of the variables. It does not matter which one is used.

3.17   Selecting Boolean, Float or Integer
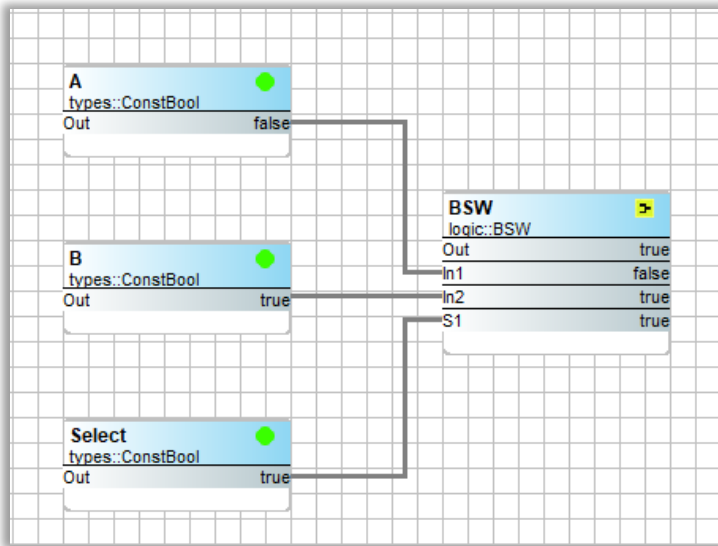
A two-binary selector switch is used to enable one variable over another. If the S1 slot is true, then the value at In2 is passed to the output of the switch. If S1 is false, then In1 is passed to the output.
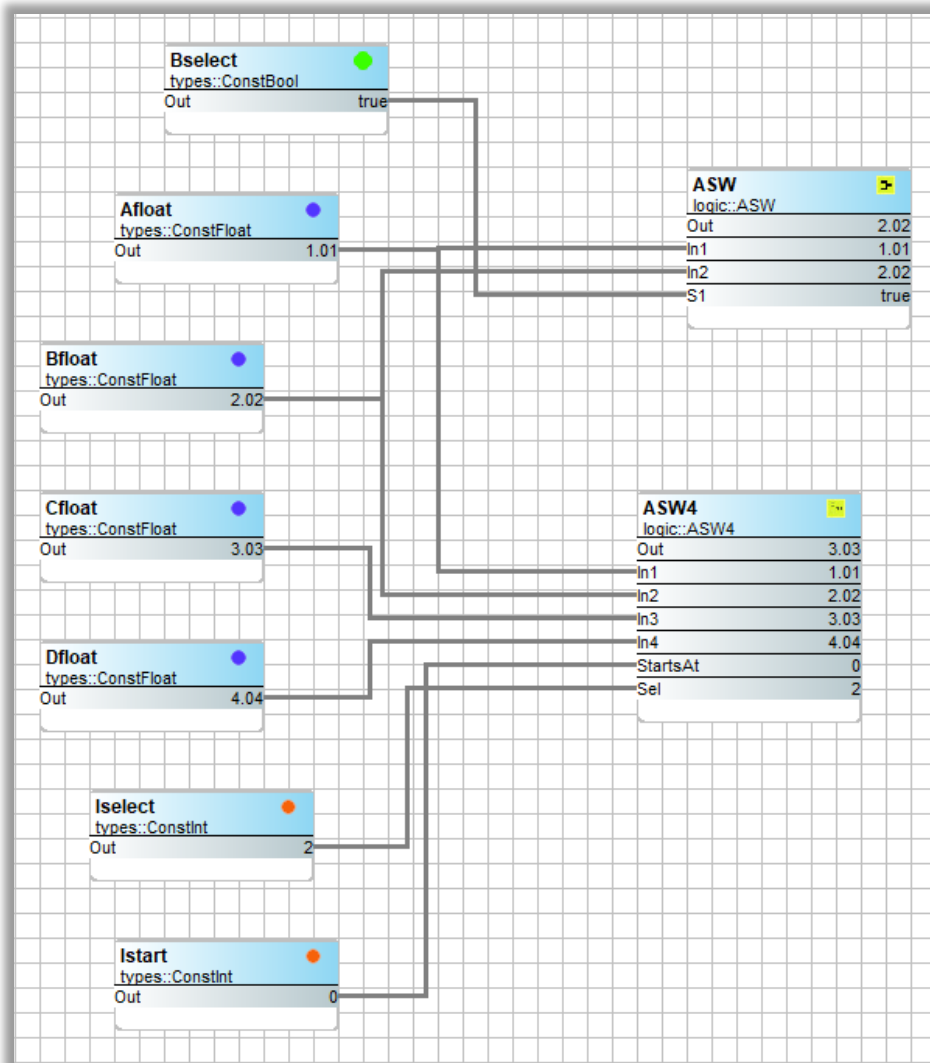
Similar in operation to the binary switch (BSW) is the analog switch (ASW). Instead of Boolean variables, the inputs are floats, but the selector (S1) is a Boolean variable. With S1 set to true, the ASW output selects input 2 (In2), otherwise Input 1 (In1) is selected. Therefore, the ASW selects one of two input float options.

The four-input analog switch (ASW4) is slightly different. There are four float inputs instead of two as with the ASW. The selector slot (Sel) is actually an integer and not a Boolean, thereby allowing the selection of up to four float inputs. Also, the selection process begins at a particular integer value (StartsAt).

In this example, the StartsAt slot has a value of 0, meaning that input 1 (In1) is selected if Sel is 0. Since Sel is 2, the third input (In3) is selected.
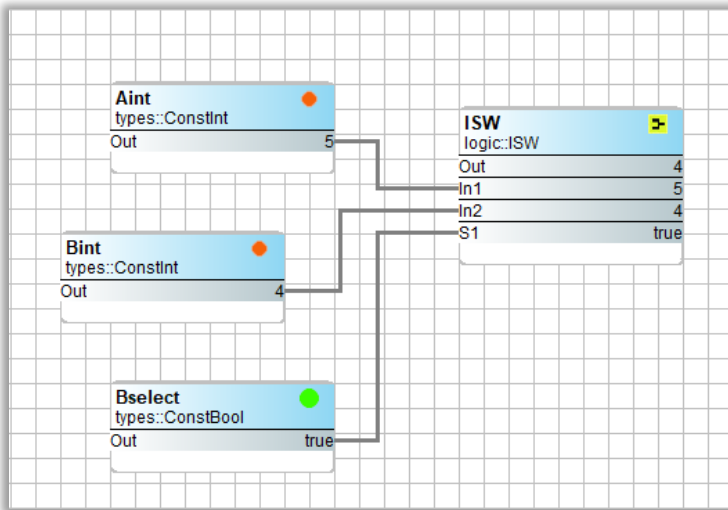
The integer switch (ISW) is much like the BSW and ASW. The selector is a Boolean, but the inputs are integers. The output remains an integer. The logic is the same. If the selector (S1) is true, then the output follows In2, otherwise it follows In1.

3.18   De-Multiplexing
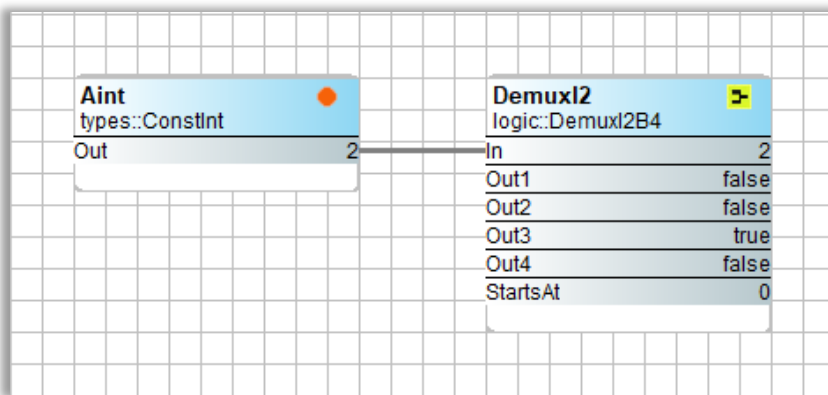
The de-multiplexer (DemuxI2) operates on integer values and provides a linear selection of the outputs based upon the value of the input. If the input is 0, the first output (Out1) is set to true. If the input is 2, the third output (Out3) is set to true.
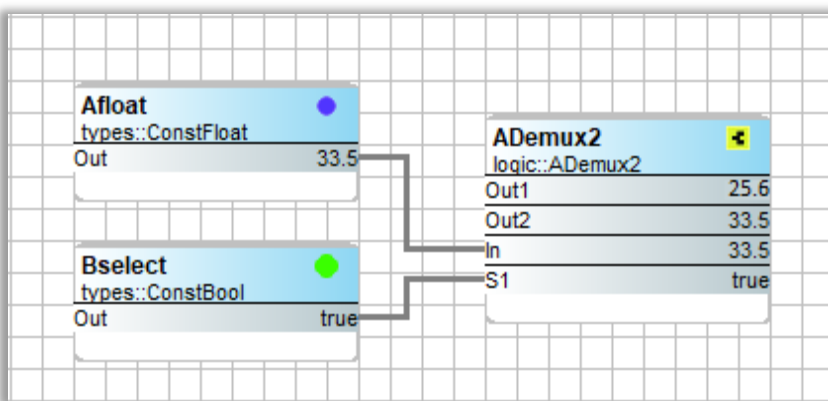
The analog de-multiplexer has only one input (In) one selector (S1), and two outputs. When the S1 is false, Out1 reflects the input. When S1 is true, Out1 will reflect the input just before the selector changed state, and Out2 will reflect the instantaneous value of the input. This component can be treated as a sample-and-hold detector.

3.19   Creating Float Addition
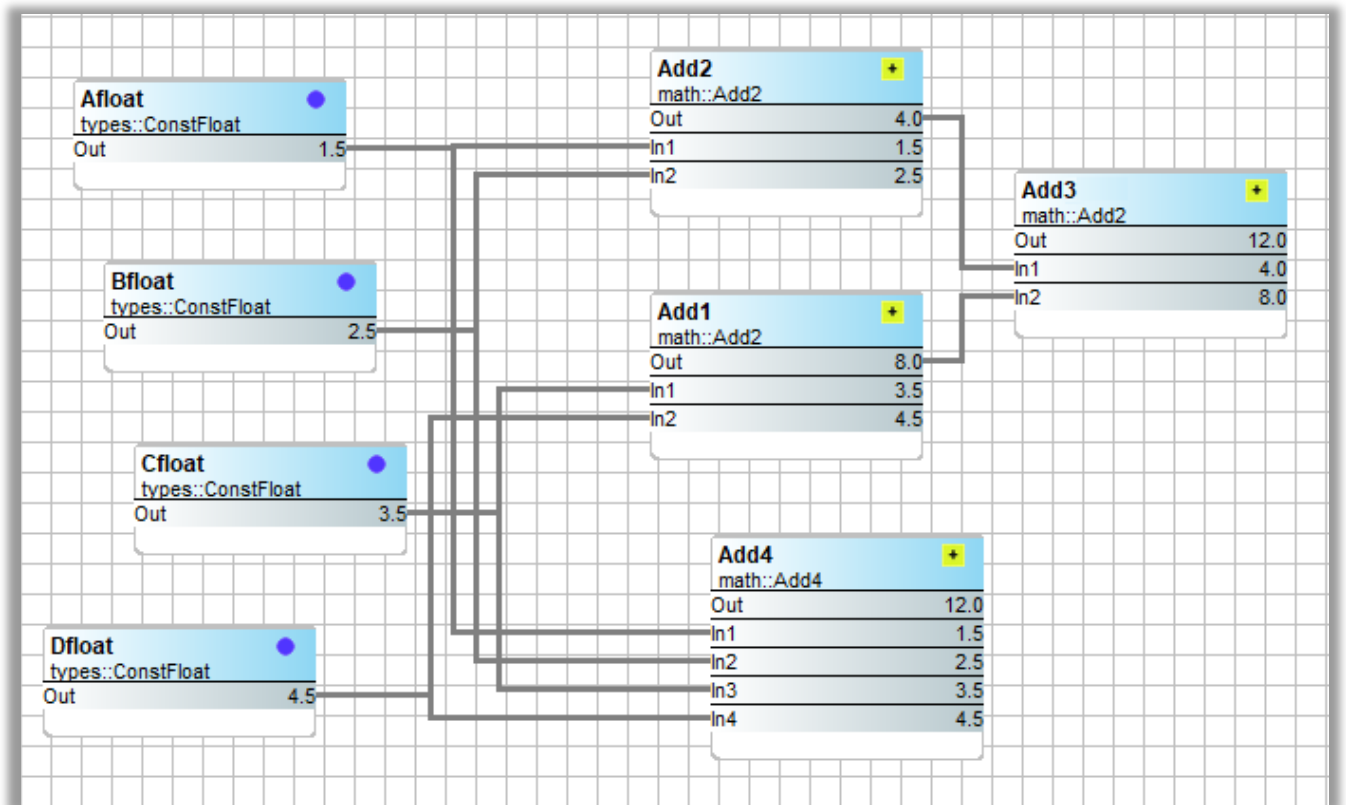
The addition (Add) components are straight forward. You can cascade two-input Add components, or you can use a four-input Add component. All inputs and outputs are floats.

3.20 Creating Float Subtraction
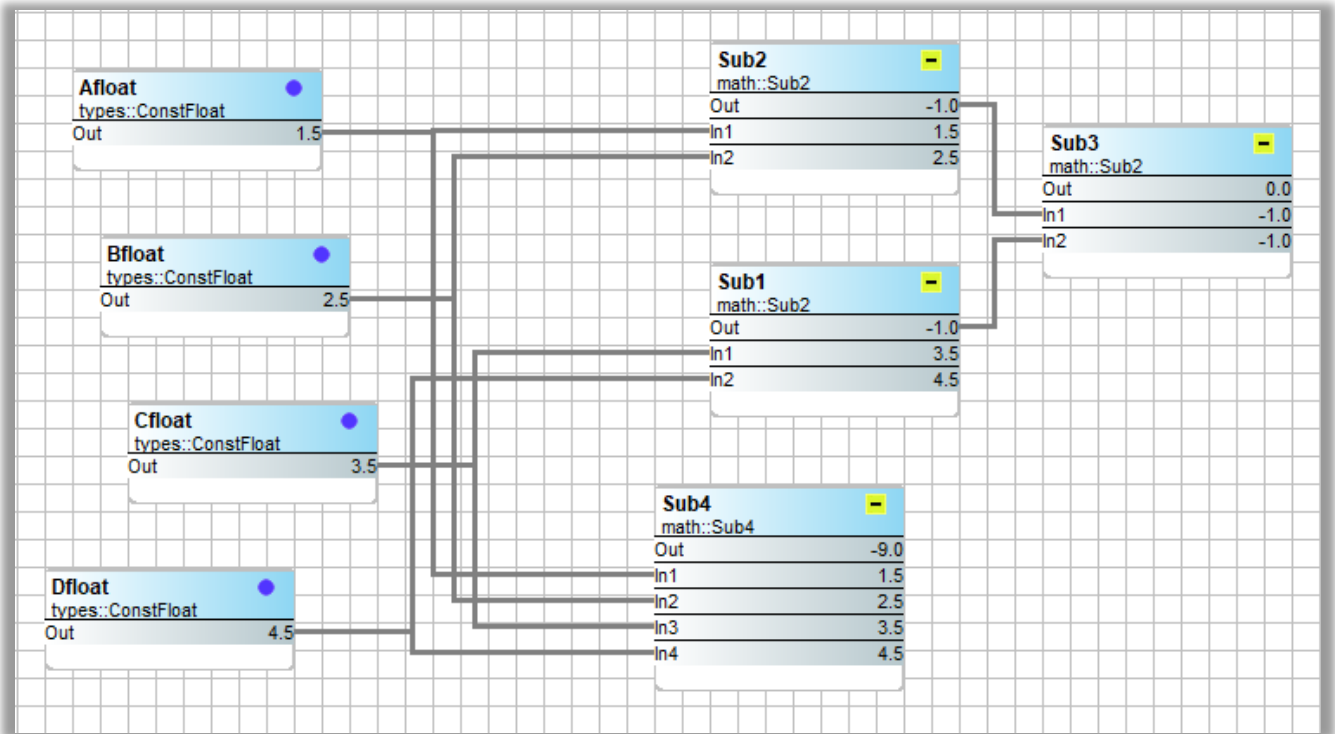
The subtract (Sub) components are also easy to work with but notice that cascading components do not yield the same results. The first input (In1) is the minuend, and all other inputs (In2, In3, In4) are subtrahends leading to outputs which represent the difference.

3.21   Creating Float Multiplication
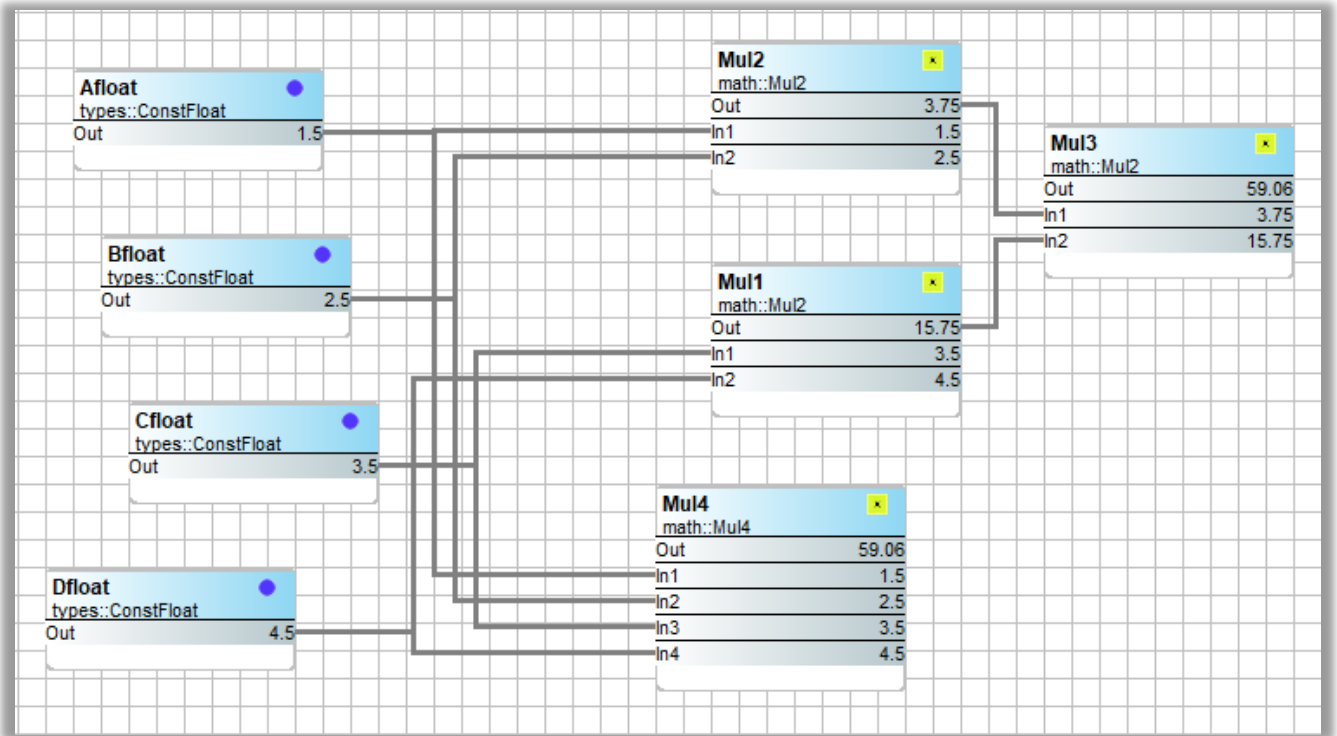
Similar to addition, float variables can be multiplied either by cascading multiply (Mul) components or by using a single larger multiplier component. All inputs and outputs are floats.

3.22   Creating Float Division
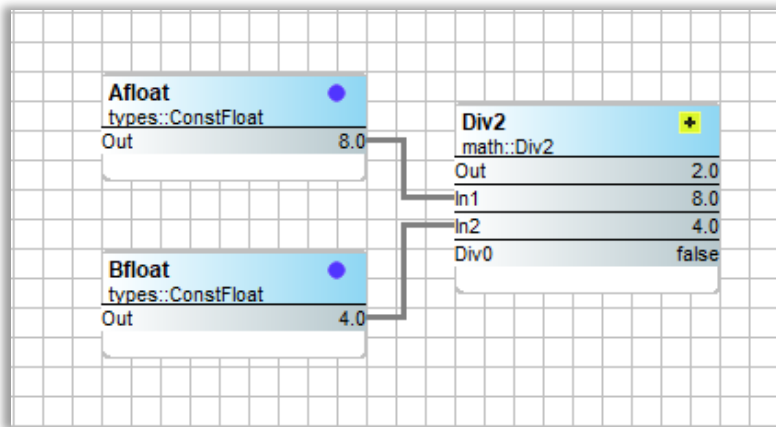
Division is also straight forward. Input 1 (In1) is the dividend, input 2 (In2) is the divisor, and the output (Out) is the quotient. Dividing by zero will result in the pin Div0 being set to true.

3.23   Finding Minimums and Maximums
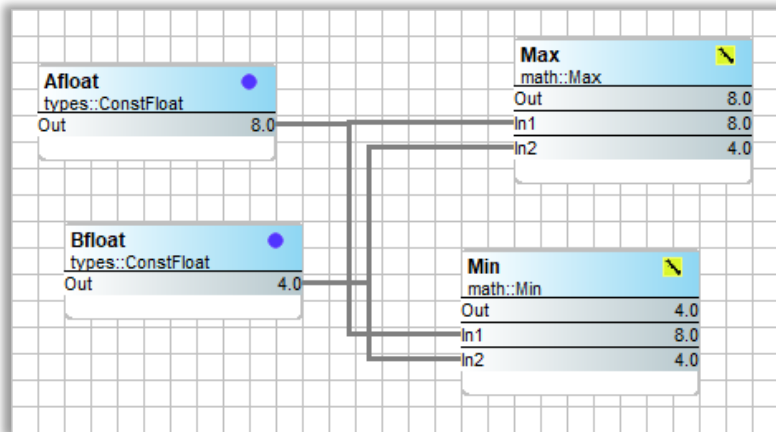
The Max component output (Out) reflects the maximum values of the two input floats (In1, In2), while the Min component reflects the minimum value of the two inputs.
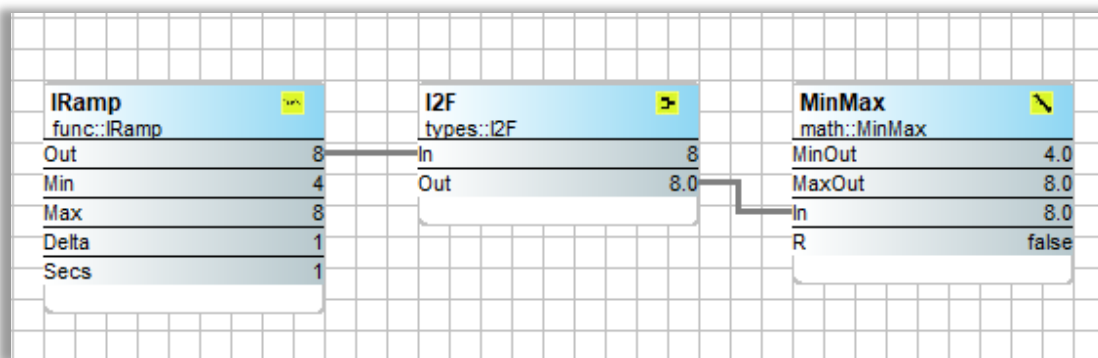
To demonstrate this operation, an IRamp was configured to generate a triangle wave with a minimum value of 4 and a maximum value of 8. The MinMax component captured the limits. Notice the need for an Integer-to-Float converter. The MinMax component is slightly more complex. There is only one input and two outputs. If R is held in the true state, the two outputs simply reflect the input state. If R if false, the MinOut captures the lowest value of the input, while MaxOut captures the maximum of the input. When connecting the component for the first time you should reset the component.

3.24   Rounding Off Floats
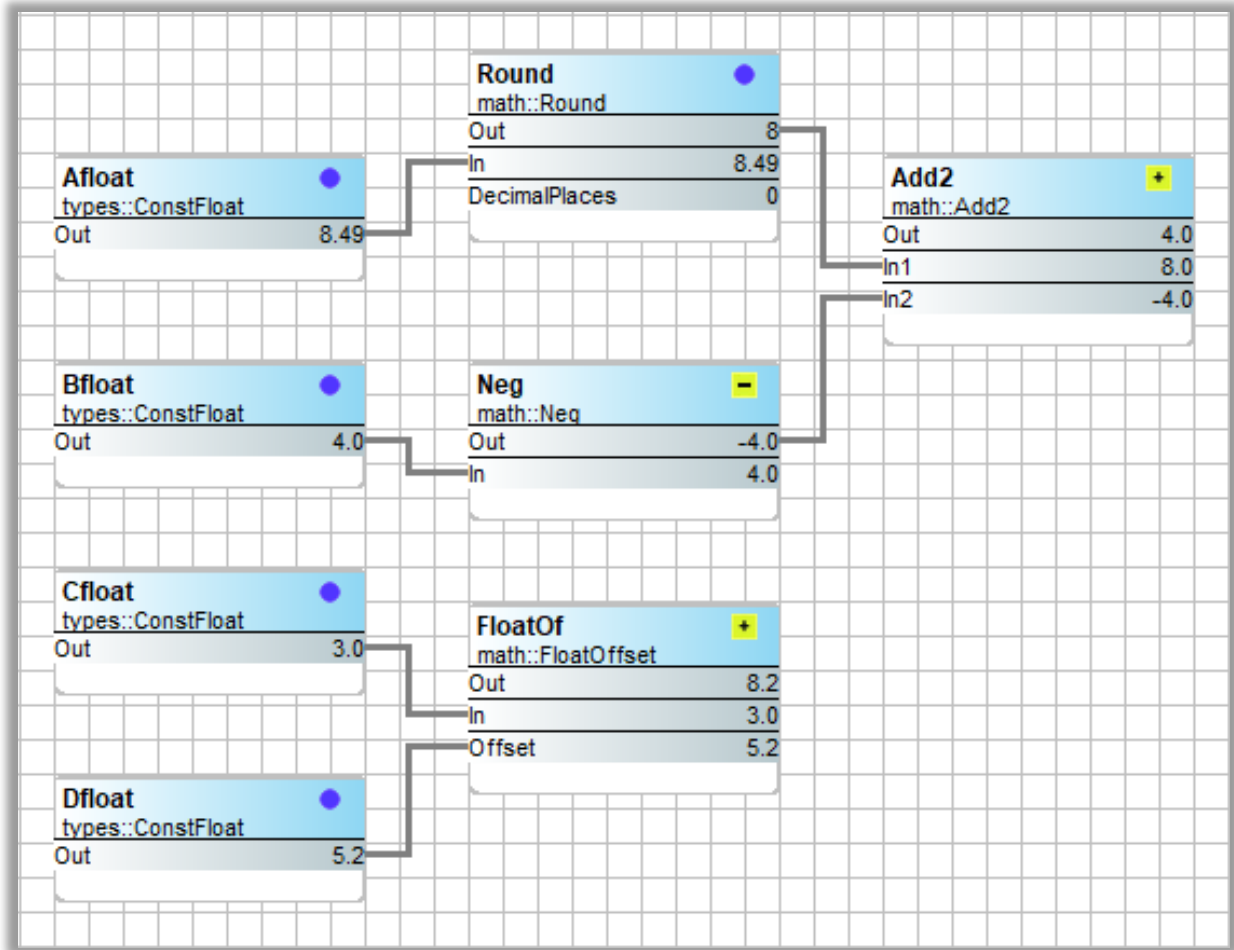
Using the Round component, you can round-off the value of a float to the closest integer value, but the output will remain a float. Using the Neg component, you can append a minus sign to a float with the output remaining a float.

The FloatOf component appends an offset value to the output. It is not necessary to use a constant component for establishing the offset amount. The offset amount can be configured within the FloatOf component.

3.25   Averaging Successive Readings
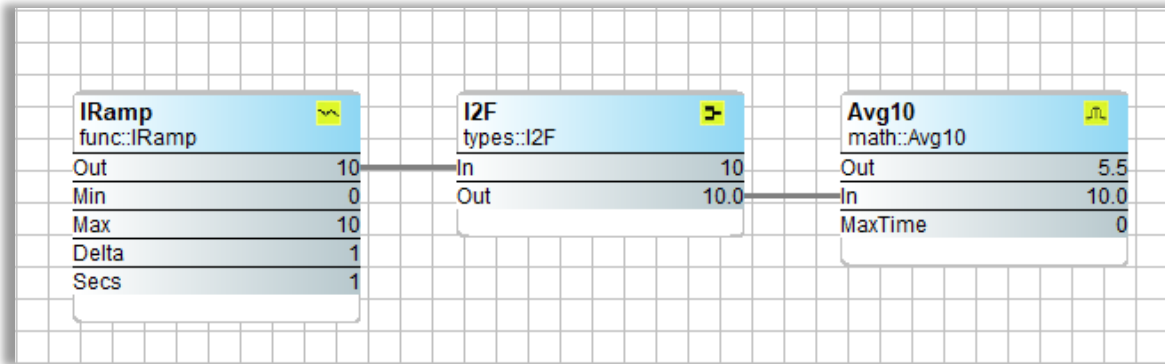
The Avg10 components averages the last ten input values and sends the result to the output. To demonstrate this, an IRamp is configured to create a triangle wave with a minimum value of 0 and a maximum value of 10. Increments are set to 1. When the IRamp reaches 10, the Avg10 component would have averaged 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 for a value of 5, which appears in the output.
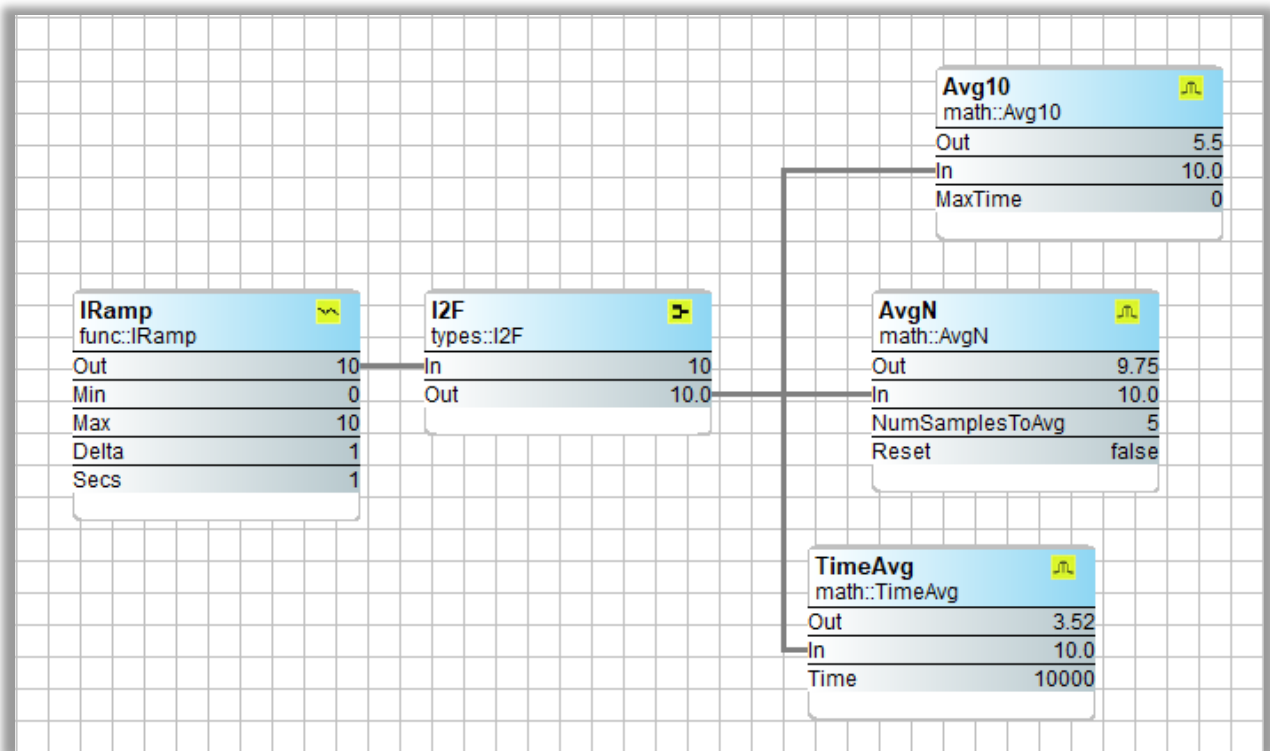
In this example, three averaging components are compared. The Avg10 averages over ten samples, but the data must change to trigger a new sample. The AvgN component can be configured for the number of samples, but it samples every scan and not just on a change in value. The TimeAvg averages over a fixed period of time which is configurable. The output does not change until all samples are obtained.

3.26   Creating On-Delays and Off-Delays
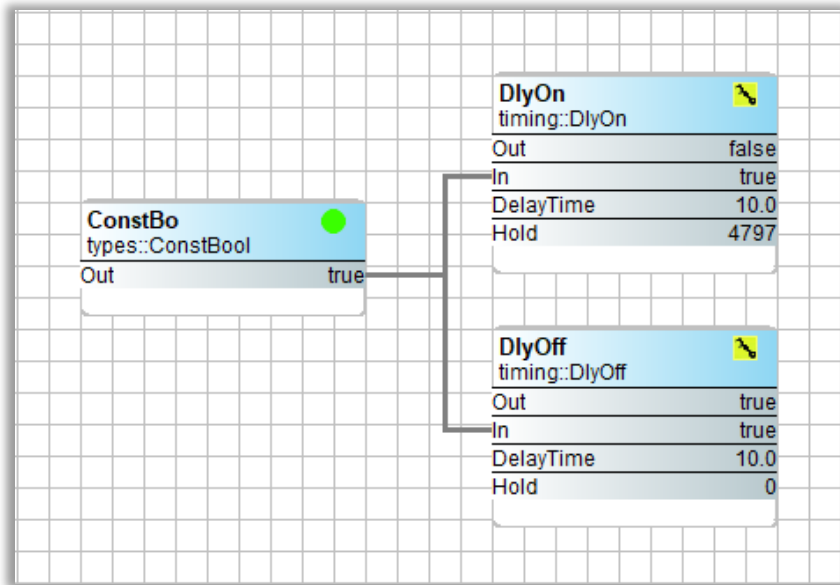
The DlyOn component is an on-delay timer which begins timing on the false to true transition of the input. Once the time (as shown is the Hold slot) goes to 0, the output will become true. This delay time is configurable. In this example, the delay timer is still timing after the input when true 4.8 seconds ago.

The Dlyoff component operates the same except it is triggered by a true to false transition of the input.
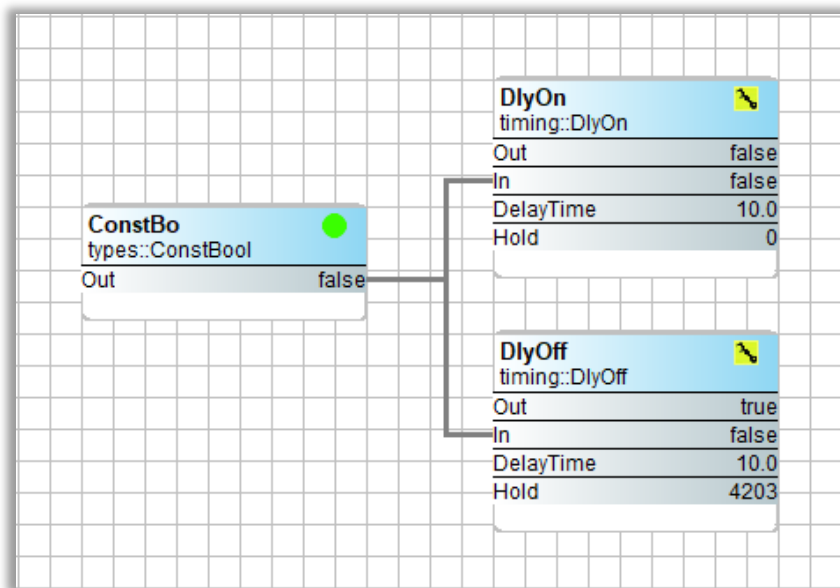
In this example, the input to the two timers made a true to false transition six seconds ago. The DlyOn components had immediately transitioned from true to false with the input, but the DlyOff timer is still timing. In another four seconds its output will become false.

3.27   Using the Timer
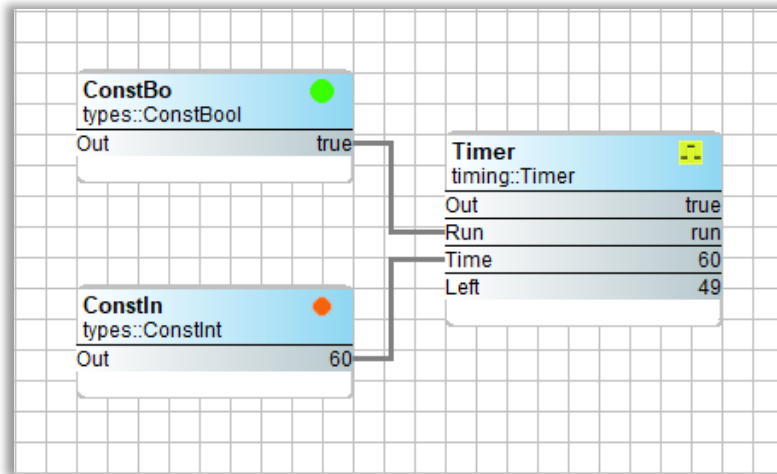
The timer component will count down from a predetermined amount when the Run input is true. A constant integer component was used to set the time, although the Timer component can be internally configured. The output will remain true during timing and transition false upon completion or if the Run input goes false. To begin a new timing period, the Run input must be cycled.

3.28   Using One-Shots — Mono-Stable Multivibrators
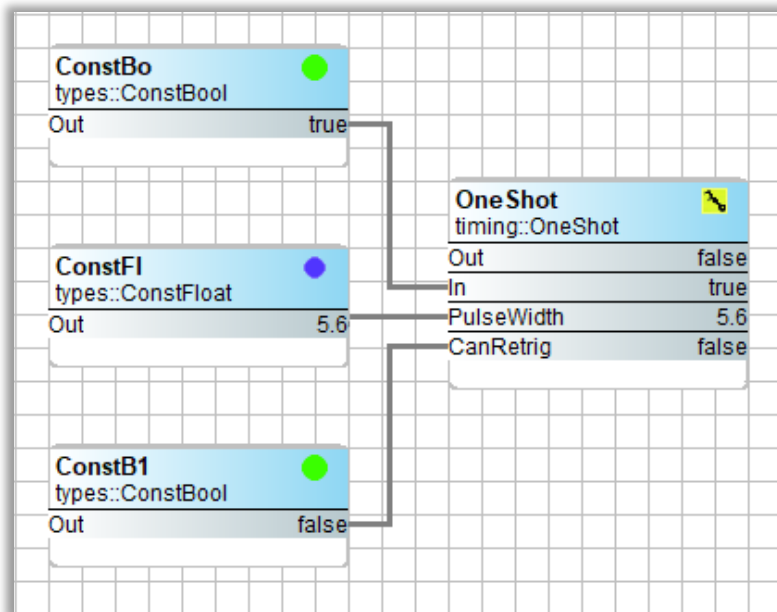
The OneShot provides a single pulse of determined value upon the false to true transition of the input signal. The output immediately goes true on its input's false to true transition, and then returns to false when timing is complete. The PulseWidth can be configured or externally programmed as shown. A retriggerable one-shot will renew timing if the input transitions from true to false to true during the timing period. A non-retriggerable will not. Notice that the PulseWidth is a float.
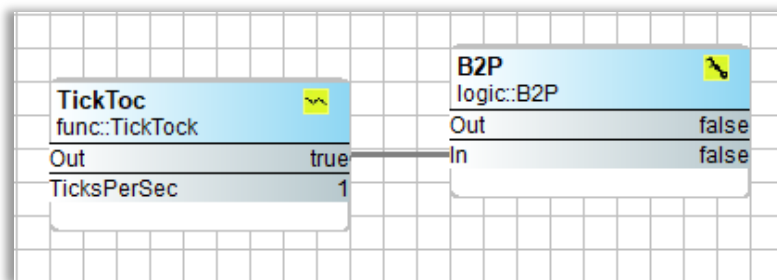
The Boolean-to-Pulse (B2P) converter is actually a very simple single-shot in that it outputs a true for only one scan time when its input goes from false to true. There are no time settings. It is used when a pulse is required after detection of an event instead of a logic level.

3.29   Creating Ramps — A-Stable Multivibrators
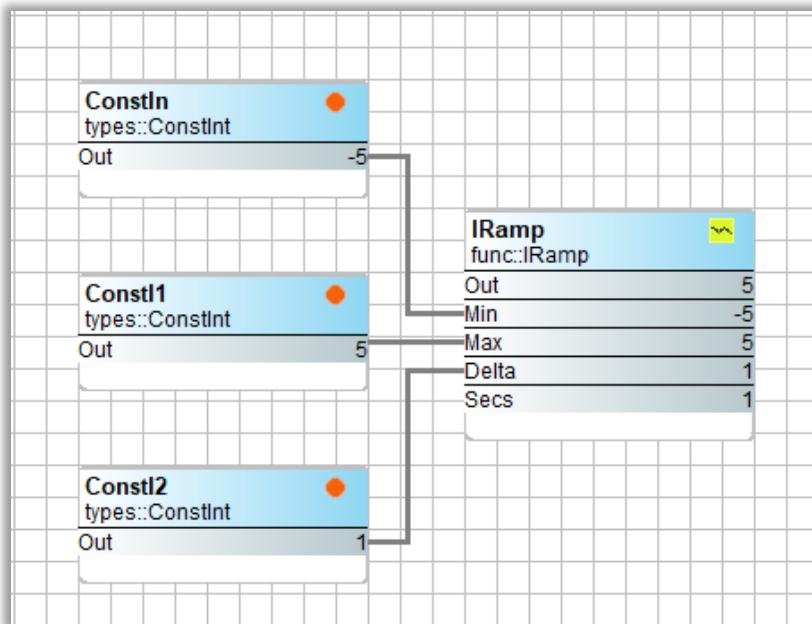
The IRamp provides a triangular output ranging from Min to Max with increments of Delta. These paraments can be configured or programmed as shown. The time increment must be configured having the units of seconds. Notice that all the configurable settings are integers, as is the output.
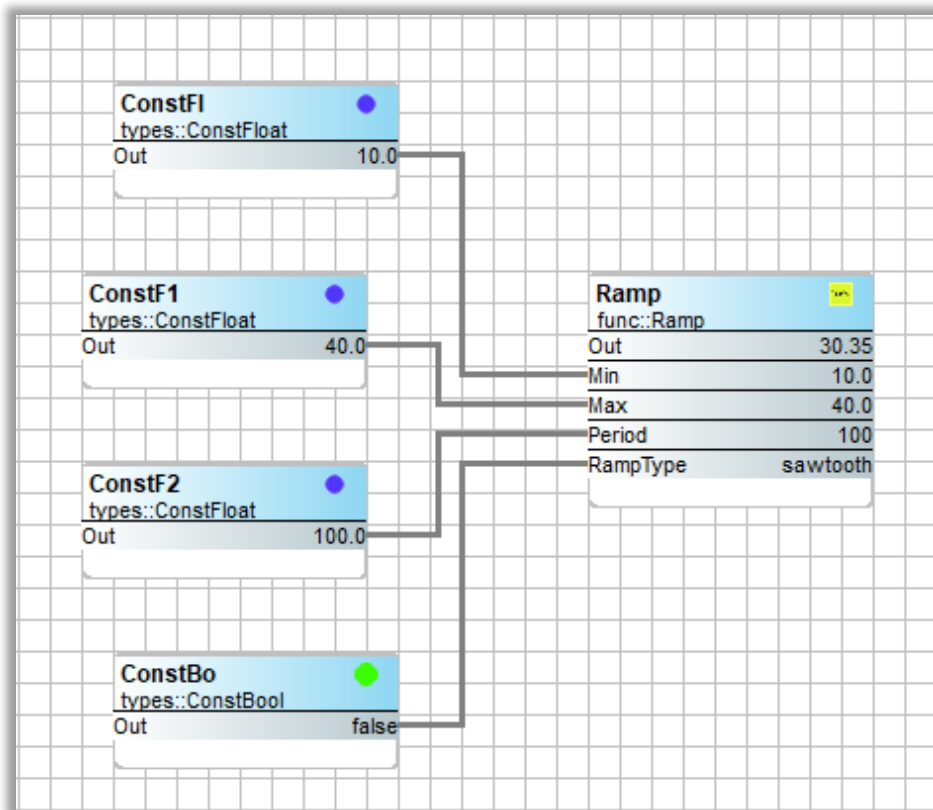
The Ramp is similar to the IRamp, but the Ramp has mostly float settings and a float output. The output of the Ramp can be configured or programmed for either a sawtooth or triangle wave. Increasing the period slows down the speed of the Ramp within the limits of Min and Max. Notice that although the Period is a float, the input to Period will be rounded to the nearest integer.

3.30   Comparing Two Floats
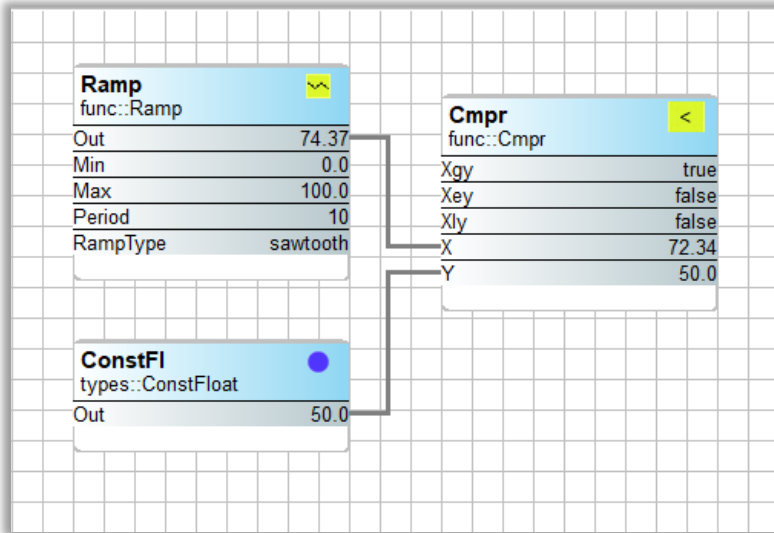
The Comparator component (Cmpr) compares the X input to that of the Y input. If X is less that Y, then the Xly output is true. If X equals Y, then Xey is true. If X is greater than Y, then Xgy is true. Both inputs are floats, and the outputs are Booleans. In this example, the output of the Ramp is compared to that of a constant. Using the default values of the Ramp, the input X varies as a triangle between 0 and 100 every 10 seconds. You can watch how the comparator outputs change over this range.

3.31   Creating a Simple Clock — the TickToc
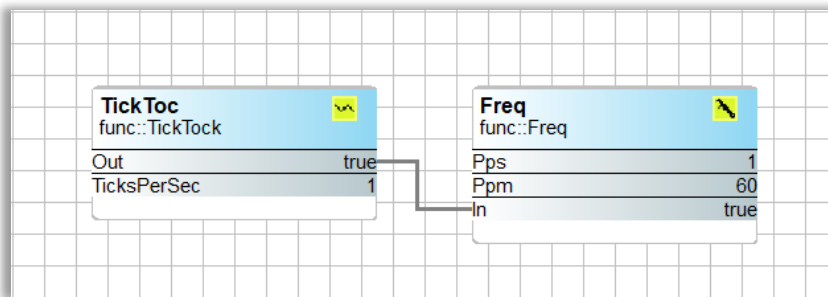
The TickToc component provides a convenient clock from 1 to 10 pulses per second. However, because the controller scan time and other processing overhead, it is recommended to use its default value of 1 second. More accurate timing is available from a real-time clock.

The Freq components can provide output values in pulses-per-second (Pps) or pulses-per-minute (Ppm). Because of the low-speed nature of these two components, the Ppm calculation will probably be the most useful.

3.32   Introducing Counters
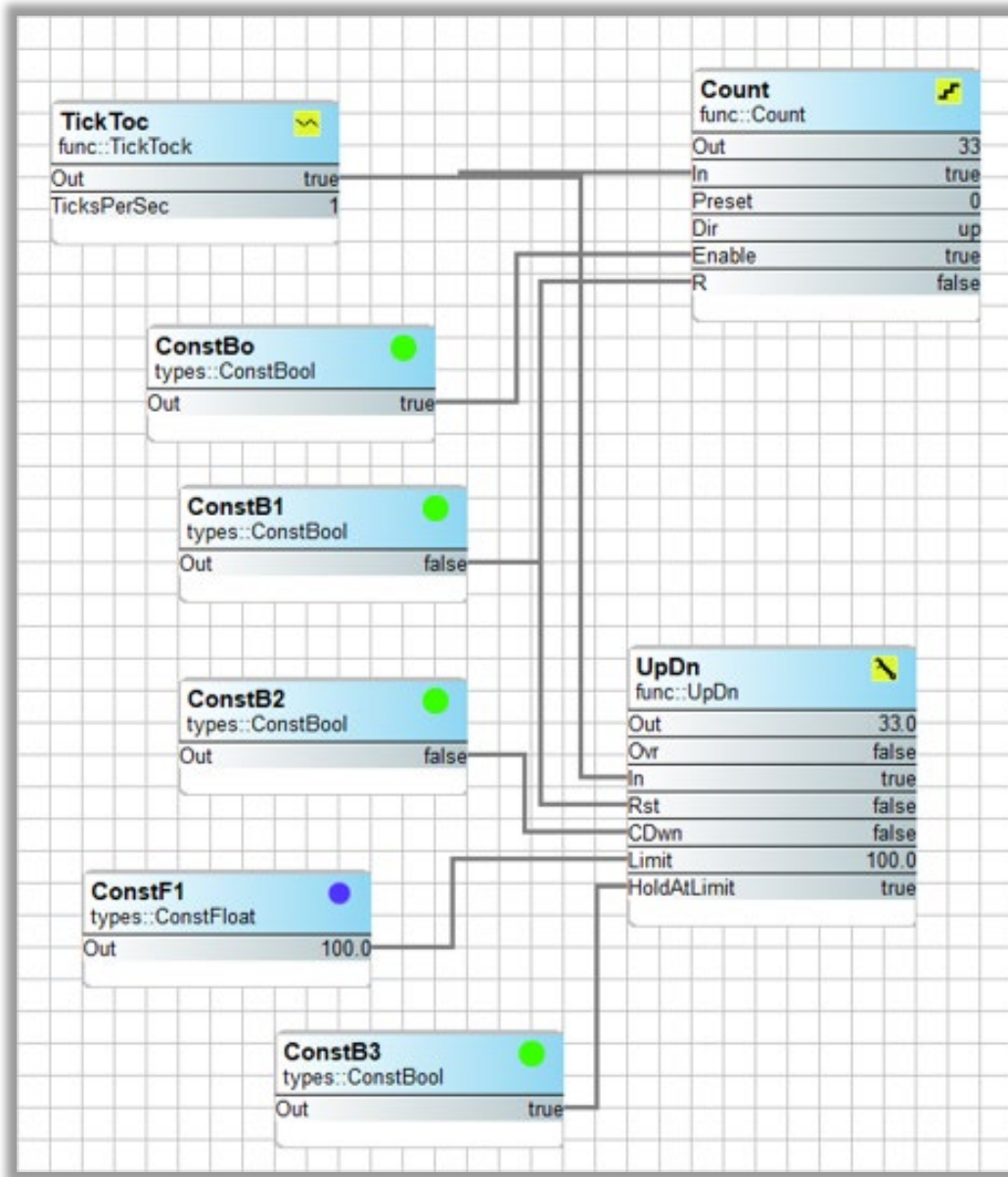
There are two counters. Count is an up/down counter with an integer output. It must be enabled to count. It can be reset to 0 or preset.

UpDn is an up/down counter with a programable direction input (C Dwn) which can also be configured. Although counters are inherently integer devices, the output of this component is a float. In this example, a limit of 100 has been programmed. Once the limit is hit, the overflow bit (Ovr) will be set. If HoldAtLimit is true, the counter will not go past 100. If it is false, the counter will continue to count past the limit, but the overflow bit will remain set. Resetting the counter returns the component to the start position while clearing the counter and overflow bit.

3.33 Operating on Real-World Signals — Hysteresis and Limiting
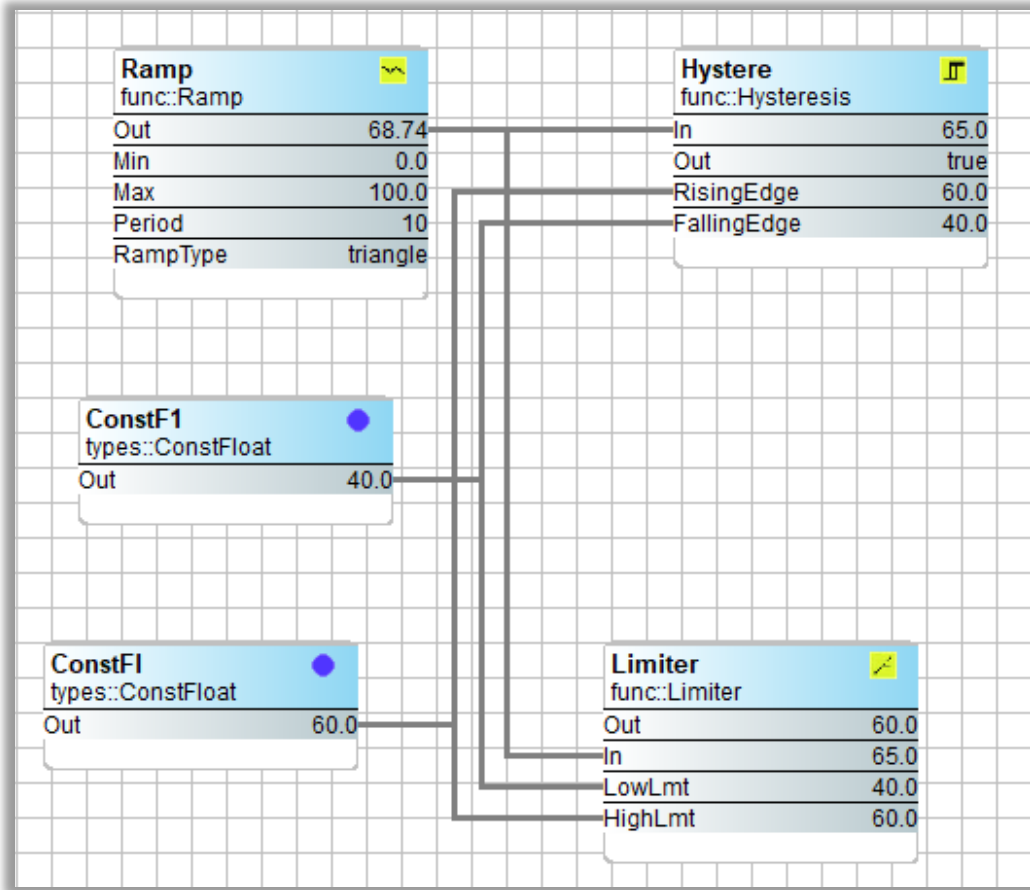
The hysteresis component (Hystere) has separate rising-edge and falling-edge trip points when setting a trigger on a float variable. It is ideal for creating a digital event from a real-world analog input. Its output is Boolean.

The Limiter component restricts the range of a float variable by outputting a float that does not exceed the configurable low-limit (LowLmt) or high-limit (HighLmt). The Limiter only limits the range of its output and does not scale the input float.

3.34   Handling Non-Linear Signals

The

IRamp			I2F			Lineari		
func::IRamp			types::I2F			func::Linearize		
Out		9	In		9	Out		81.0
Min		0	Out		9.0	In		9.0
Max		100				X0		0.0
Delta		1				Y0		0.0
Secs		1				X1		1.0
						Y1		1.0
						X2		2.0
						Y2		4.0
						X3		3.0
						Y3		9.0
						X4		4.0
						Y4		16.0
						X5		5.0
						Y5		25.0
						X6		6.0
						Y6		36.0
						X7		7.0
						Y7		49.0
						X8		8.0
						Y8		65.0
						X9		9.0
						Y9		81.0
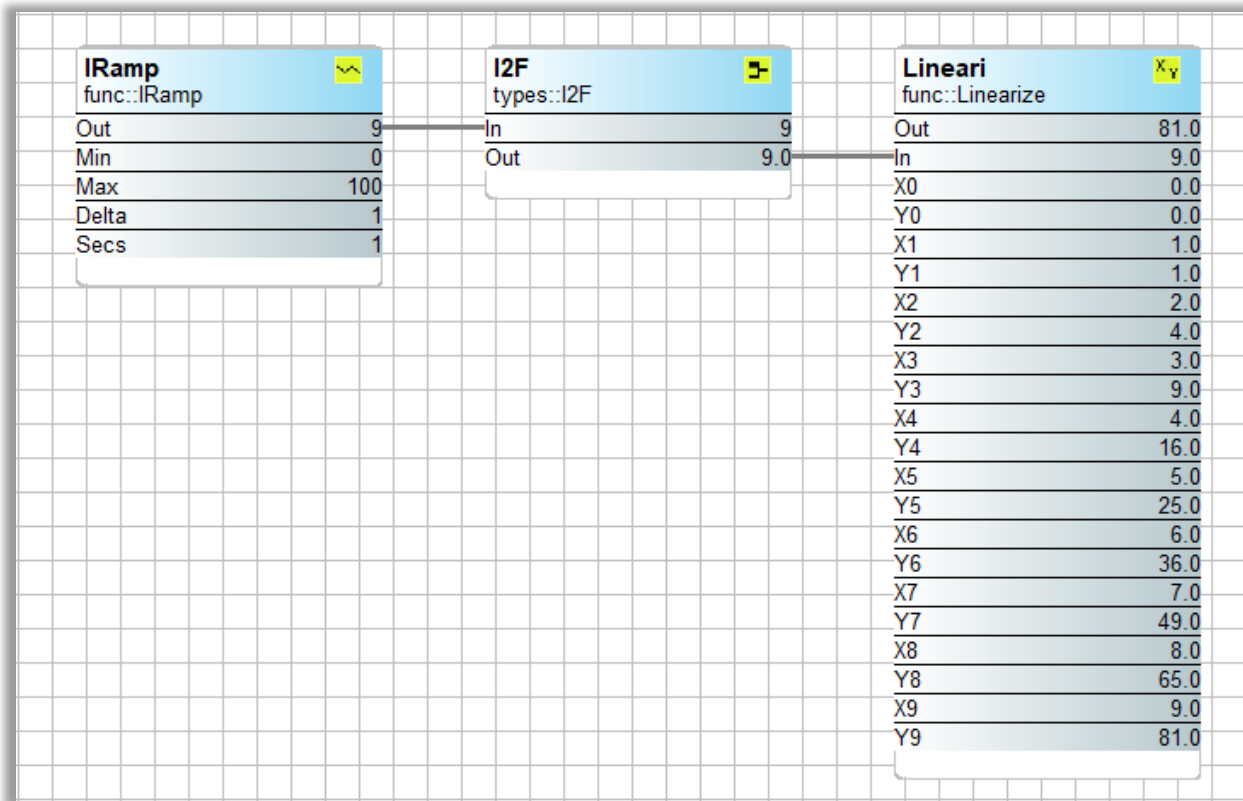
Property	Value
⌄ Lineari	
Name	Lineari
Meta	504168449
Out	57.0
In	7.5
X0	0.0
Y0	0.0
X1	1.0
Y1	1.0
X2	2.0
Y2	4.0
X3	3.0
Y3	9.0
X4	4.0
Y4	16.0
X5	5.0
Y5	25.0
X6	6.0
Y6	36.0
X7	7.0
Y7	49.0
X8	8.0
Y8	64.0
X9	9.0
Y9	81.0

Linearize component (Lineari) operates on a float input and creates a piece-wise linear representation of a non-linear input (such as a thermistor), or it can create a non-linear piece-wise representation of a linear input. There is complete flexibility in the defining the ten X,Y coordinates along the output curve.

The component determines the approximate output between the ten coordinates using linear interpolation.

In this example, we will do the reverse of what is commonly done. We will use a linear input and create a non-linear output that approximates the equation Y=X*X over the range of X values from 0 to 9. We need to input corresponding values of Y that obey the desired equation. To make it easy we will use integer values, but this is not a restriction. For example, the square of 4 is 16, and the square of 5 is 25. We enter the X values as an independent variable and then the Y values as the dependent variable. We need to be careful that the input does not exceed 9 in this example because we do not define a corresponding value for Y above 9.

You can test the interpolation by entering a value for X in the In slot, assuming not link is connected to the Linearize component. This is done here. Notice that the result is 56.5 for an input value of 7.5. The correct value would have been 56.25, which is very close.

3.35   Creating a Simple Set-Reset Flip Flop — Bi-Stable Multivibrator
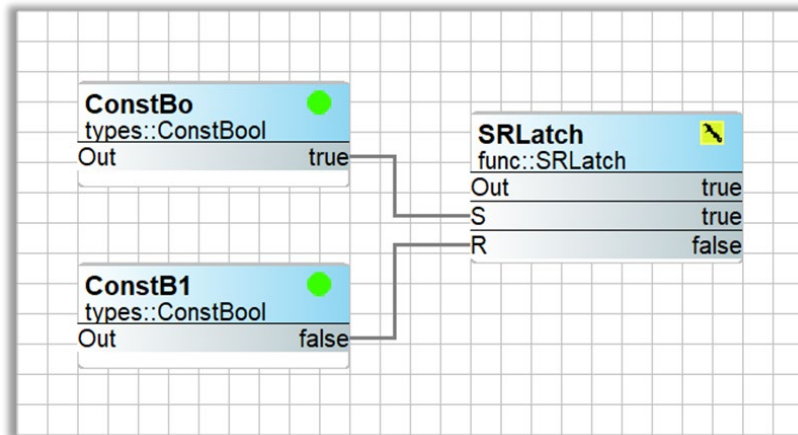
The SRLatch appears to be a straight-forward logic block. The output would become true if the set (S) pin is high and would go low if the reset (R) pin goes high. However, both the S and R pins are positive leading-edge sensitive. Regardless of their steady-state condition, the output (Out) will only change on the false-to-true transition of either input.

If this occurs on the S pin, the output goes high and will remain high until the R pin does its transition.

On the rare condition that both S and R transition from false-to-true during the same logic scan, R will take precedence because its state is tested last in the logic, and therefore the output will be false.

3.36   Creating the Loop Component — Basic Analog Controller
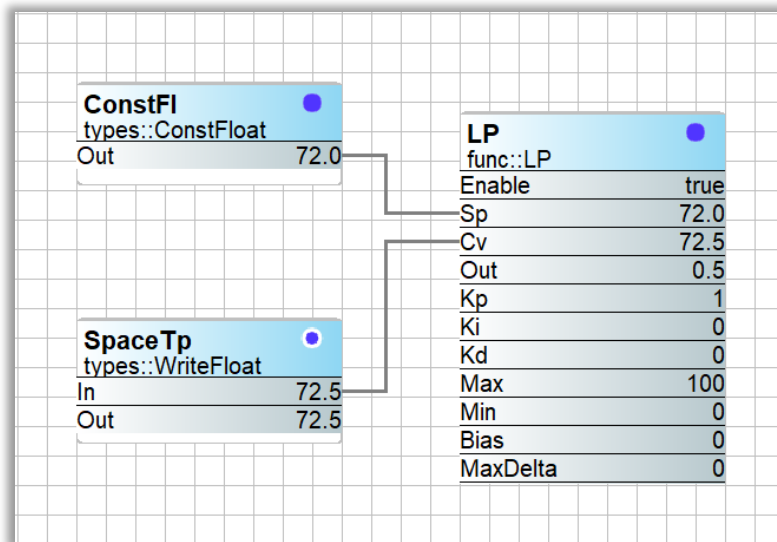
The LP or loop component is one of the most complex components. It can provide three modes of control P-proportional, I-integral, and D-derivative. In this example, we will assume a temperature loop with a setpoint (Sp) of 72 degrees and a controlled variable (Cv) currently as 72.5 degrees which is the space temperature that we want to control.

3.37 Creating the Loop Component — Basic PID Controller

Property	Value
⌄ LP	
Name	LP
Meta	269090817
Enable	true
Sp	72.0
Cv	72.5
Out	0.5
Kp	1
Ki	0
Kd	0
Max	100
Min	0
Bias	0
MaxDelta	0
Direct	true
ExTime	1000

Enable must be configured true, otherwise there is no control.

Kp is the proportional gain which defaults to 1. Notice that the error signal is Cv-Sp or 0.5. The error multiplied by the proportional gain of 1 yields an output of 0.5. If the Ki and Kd factors are used, their contributions are also multiplied by the proportional gain factor. Ki is the integral gain in units of resets per minute. It is multiplied by the error signal. Kd is the derivation gain in seconds, and it is also multiplied by the error signal.

Min and Max are the limits of the output signal. They can be set to any value. Bias can offset the output regardless of the error. MaxDelta sets the rate of change of the output within the output limits. This will slow the output swing.

For a cooling application, set Direct to true. For heating, set it to false. The loop equation is solved each execute time (ExTime) in milliseconds.

Bias only applied to proportional-only (P) control. When using a PI controller, reset-windup can be minimized by limiting the output range.

3.38 Creating a Linear Sequencer — Bar-Graph Representation of a Float
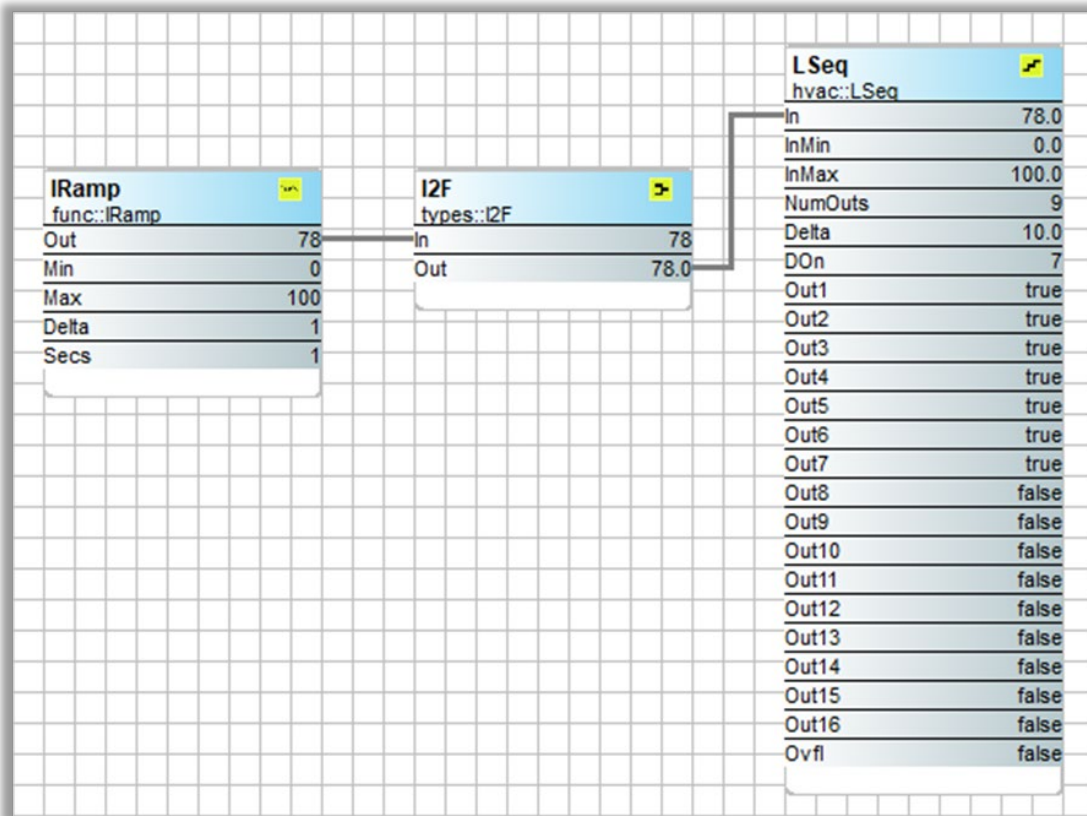
Property	Value
⌄ LSeq	
Name	LSeq
Meta	470351873
In	60.0
InMin	0.0
InMax	100.0
NumOuts	9
Delta	10.0
DOn	6
Out1	true
Out2	true
Out3	true
Out4	true
Out5	true
Out6	true
Out7	false
Out8	false
Out9	false
Out10	false
Out11	false
Out12	false
Out13	false
Out14	false
Out15	false
Out16	false
Ovfl	false

The linear sequencer (Lseq) provides a digital representation of an input float similar in operation to a bar graph on audio equipment. It is easier to understand its operation using an integer input. There are 16 possible Boolean outputs plus one overflow (Ovfl) flag. The input ramp provides a triangle wave from 0 to 100. The sequencer was configured for a 0 minimum input and 100 maximum input. The maximum number of outputs was configured for 9, yielding a Delta of 10.

The range of the linear sequencer is configured using InMin at the low end and InMax at the high end. Selecting the number of outputs (NumOuts) determines the difference (Delta) between successive outputs turning on. In this case, the range is 100, and the number of desired outputs is 9. Divide 100 by NumOuts +1, and you will get a Delta of 10.

You will notice that the input (In) is at 60, and D On is indicating that six outputs are on. With an input between 0-9, there are no outputs on, but once you hit a decade such as 10, 20 on up to 90, successive outputs will come on. At the maximum of 100, nine outputs will come on. If the input exceeds the maximum intended, the overflow flag will set, but the number of outputs will remain as specified by NumOuts.

3.39 Creating a Reheat Sequencer — Four Staged Outputs from a Float Input
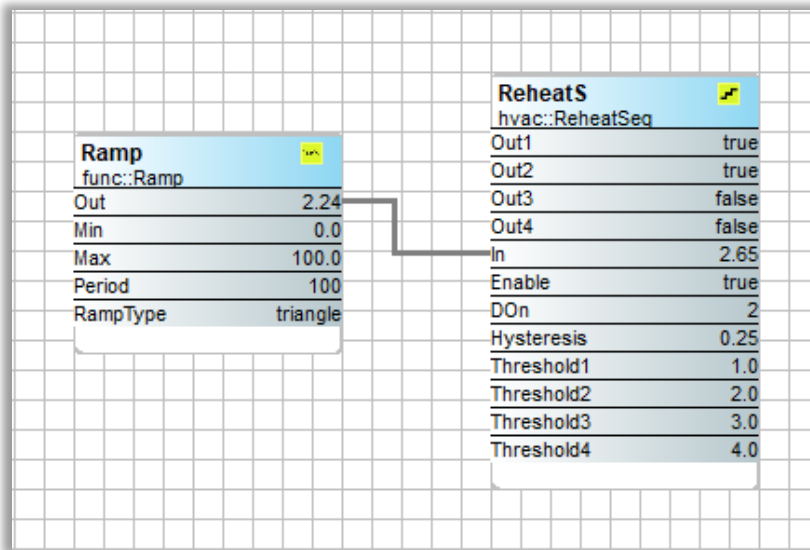
The reheat sequencer (ReheatS) provides a linear sequence of up to four outputs based upon their input float (In). The threshold for the four outputs can be configured for increasing values of the input. As the input increases to each threshold, the corresponding output will go on. As the input decreases below the threshold, the corresponding output will remain on until the hysteresis value is exceeded.

Enable must be set to true, otherwise the outputs will be false.

There are four possible threshold settings corresponding to four outputs. As the input signal increases to each threshold, its corresponding output goes on and stays on until the input drops below the threshold plus the value of the hysteresis.

The input signal is decreasing, but it has not exceeded the amount of the threshold, so output 3 (Out3) remains set. Once the signal is below 2.75, output 3 will go off.

3.40 Reset — Scaling a Float Input between Two Limits

Ramp		func::Ramp	
Out			28.24
Min			0.0
Max			100.0
Period			100
RampType			triangle

Reset		hvac::Reset	
Out			82.84
In			28.24
InMin			0.0
InMax			100.0
OutMin			32.0
OutMax			212.0
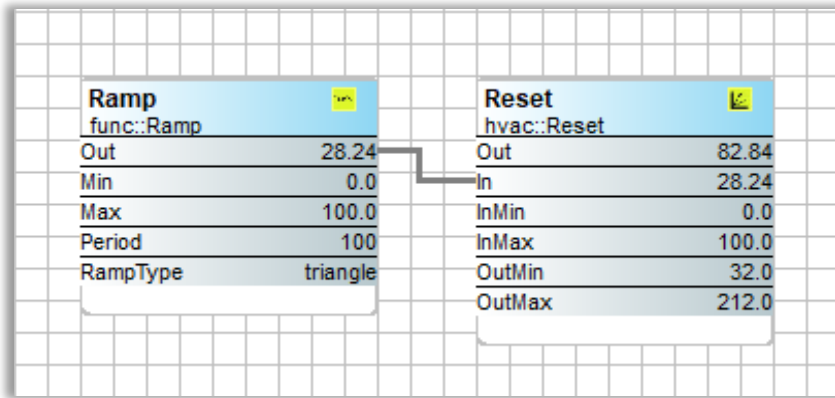
The Reset component (Reset) will scale the output linearly between two limits. The input ranges must be configured by setting InMin and InMax. The corresponding output for those two points must be configured as OutMin and OutMax. If the input signal exceeds the defined input range, the output will be clamped to one of the two output limits.

Property	Value
⌄ Reset	
Name	Reset
Meta	336003073
Out	80.09
In	26.71
InMin	0.0
InMax	100.0
OutMin	32.0
OutMax	212.0

In this example, we are converting degrees Celsius to degrees Fahrenheit within the 0 - 100-degree Celsius range. Therefore, we set OutMin an OutMax to the corresponding Fahrenheit values. All Celsius input values between these two limits will be interpolated thereby providing the correct Fahrenheit values.

3.41 Setting Tstat — Basic On/Off Temperature Controller

ConstFl		types::ConstFloat	
Out			72.0

SpaceTp		types::WriteFloat	
In			71.4
Out			71.4

Tstat		hvac::Tstat	
Diff			1.0
IsHeating			true
Sp			72.0
Cv			71.4
Out			true
Raise			true
Lower			false
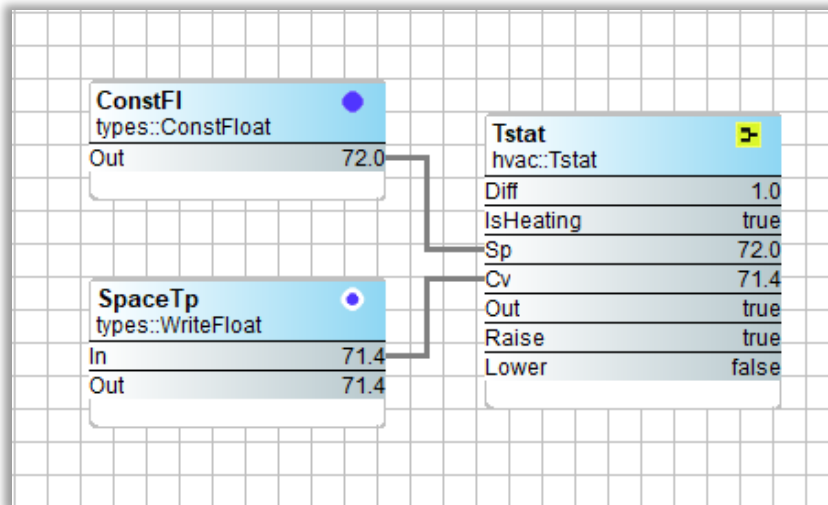
The Tstat is an on/off temperature controller for either heating or cooling. For heating configure, the IsHeating is set to true. The deadband can be set by the Diff value. If the controlled variable (Cv) deviates from the setpoint (Sp) by half the Diff value, the output (Out) will become true and stay set until Cv deviates from the setpoint by a like amount in the other direction. In this way, Diff also provide hysteresis. The Raise and Lower outputs are a function of the IsHeating setting.

If Is heating is true, Out=Lower, otherwise Out=Raise.

3.42   Setting the Real-Time Clock and Scheduling

DateTim	
datetimeStd::DateTimeServiceStd	
Nanos	679082239000000000
Hour	17
Minute	57
Second	19
Year	2021
Month	7
Day	8
DayOfWeek	4
UtcOffset	0
OsUtcOffset	false
Tz	

DailySc	
basicSchedule::DailyScheduleBool	
Start1	0:0
Dur1	0:0
Start2	0:0
Dur2	0:0
Val1	false
Val2	false
DefVal	false
Out	false

DailyS1	
basicSchedule::DailyScheduleFloat	
Start1	0:0
Dur1	0:0
Start2	0:0
Dur2	0:0
Val1	0.0
Val2	0.0
DefVal	0.0
Out	0.0

The DateTim component provides real-time information. There is no need to place it on the wiresheet.

However, if you need specific information from the component for the driving logic, you can connect to the various integer outputs, such as Hour, Minute and Second.

There are two schedule components which have different output types. One is for Boolean, and the other is for float. There is no need to connect the DateTim component to either of the schedulers. Each scheduler can handle two events over the 24-hour period by configuring the time and duration of each event. The output of each schedule will change with each event. If more events or more outputs are needed, multiple schedulers can be placed on the wiresheet.

Property	Value
∨ DailyS1	
Name	DailyS1
Meta	637796353
Start1	12:00 AM
Dur1	0:0
Start2	12:00 AM
Dur2	0:0
Val1	0.0
Val2	0.0
DefVal	0.0
Out	0.0

Configuration of the two scheduler components is similar. For the float version, Val1 and Val2 need to be specified along with the start times (Start1 and Start2) and the durations (Dur1 and Dur2). The output (Out) will assert either Val1 and Val2 during the scheduled times. If neither are programmed, the DefVal should be configured.

3.43   Creating Priority Arrays

Priorit	
pricomp::PrioritizedBool	
SourceLevel	in16
OverrideExpTime	0
In1	null
In2	null
In3	null
In4	null
In5	null
In6	null
In7	null
In8	null
In9	null
In10	null
In11	null
In12	null
In13	null
In14	null
In15	null
In16	true
Fallback	null
Out	true
MinActiveTime	0
MinInactiveTime	0

ConstBo
types::ConstBool
Out — null

ConstB1
types::ConstBool
Out — true
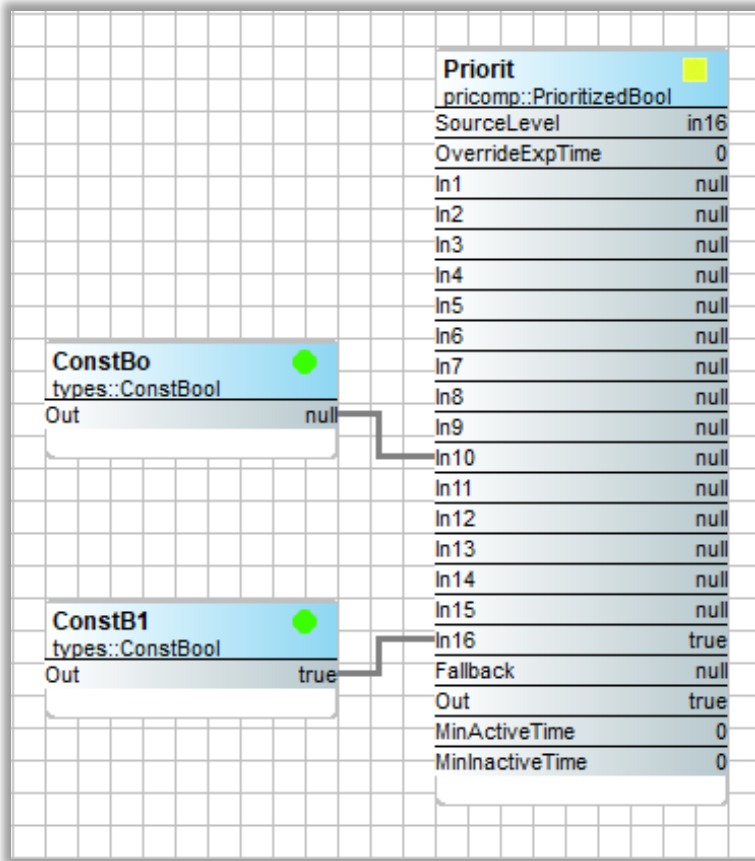
Priority array (Priorit) components exist for Boolean, float and integer values. Up to 16 levels of priority from In1 to In16 can be assigned. In1 has the highest priority and In16 the lowest. If a priority level is not assigned, it is marked as a Null and therefore ignored. If a Null is inputted to priority array as shown in this example, the priority array will ignore it and choose the next in line input. The Boolean version of the priority array has two timer settings – one for minimum active time and one for minimum inactive time. If the highest priority device changes from false to true and then back to false, the priority component will maintain the event for the configured times.

Priori1	
pricomp::PrioritizedFloat	
SourceLevel	in10
OverrideExpTime	0
In1	null
In2	null
In3	null
In4	null
In5	null
In6	null
In7	null
In8	null
In9	null
In10	5.0
In11	null
In12	null
In13	null
In14	null
In15	null
In16	6.0
Fallback	null
Out	5.0

ConstFl
types::ConstFloat
Out — 5.0

ConstF1
types::ConstFloat
Out — 6.0
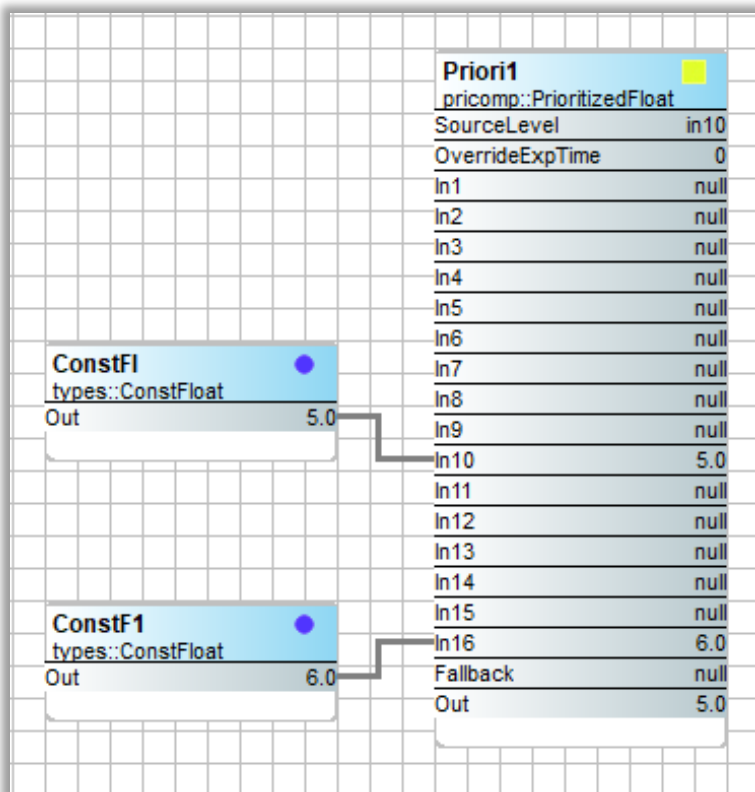
There is a fallback setting in each priority array that can be specified. If no valid priority input exists, the Fallback value is transferred to the output.
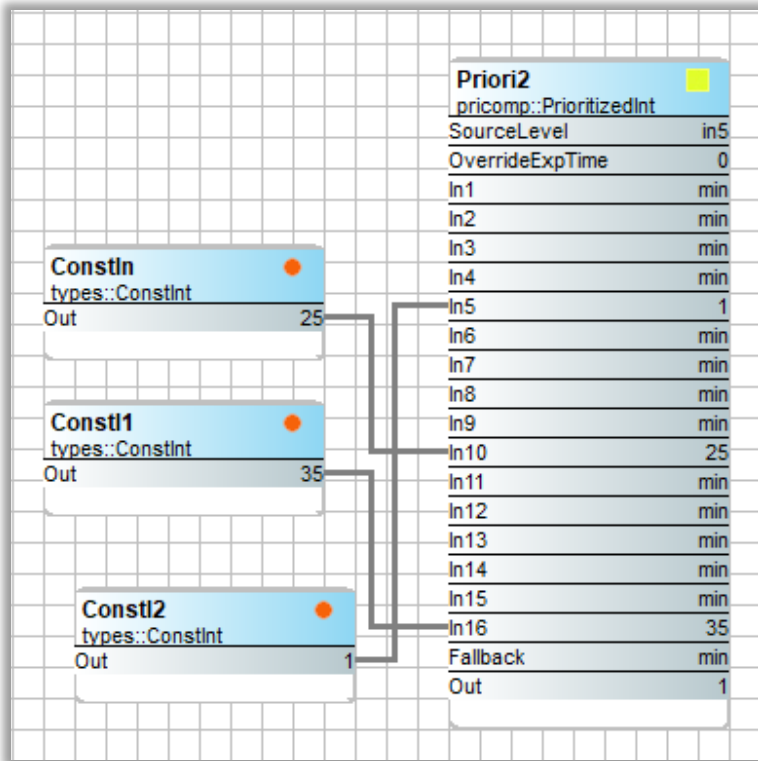
The OverrideExpTime guards against the possibility of an indefinite override condition.
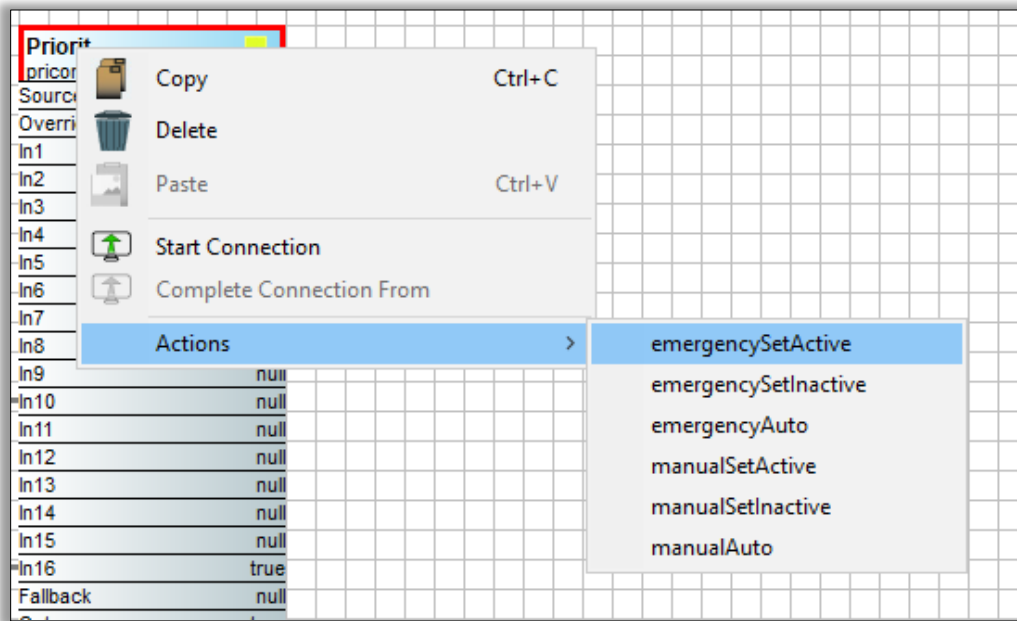
When you right-click on the priority component and select Actions, you will have several choices for overriding the current priority selection made by the component. The override choices vary depending on the type of variable supported by the priority component. In this example, the Priority Boolean was selected.

Setting an override using a tool is only temporary. Eventually, the component will time out and revert to normal priority selection.

4 Contemporary Controls' Platform Dependent Kits

4.1 BASremote

Pre-installed Sedona Component Kits in BAScontrol Unitary Controllers: The following table identifies what Sedona component kits are factory installed (√), and what kits can be optionally installed (opt.) on Contemporary Controls' family of BAScontrol unitary controllers.

4.1.1 BASremote Platform Kit (CControls_BASR8M_Platform)

BASremo	
CControls_BASR8M_Platform::BASremotePlatformService	
PlatformId	ccontrols-BASR8M-Platform-1.2.28
PlatformVer	1.2.28
MemAvailable	6336512

The BASremote platform kit has one component that advises the programmer how much usable memory is available for application programming. With a Linux platform, memory is seldom an issue.

4.1.2 BASremote Service Kit (CControls_BASR8M_Services)

The BASremote service kit allows Sedona application to tie into real-world inputs and outputs after object instance configuration. For the BASremote master, object instance assignments must match the I/O channel assignment. For configuring expansion module and virtual points, consult the BASremote User Manual for details. For the online status to revert to true, the point must be properly configured, must be actively scanned by the hardware and not be in a forced state.

InpBool	
CControls_BASR8M_Services::InpBool	
Instance	0
Out	false
Online	false

Input Boolean — BASremote binary input

Out = value of the real-world binary input

InpFloa	
CControls_BASR8M_Services::InpFloat	
Instance	0
Out	0.0
Online	false

Input Float — BASremote analog input or value

Out = value of the real-world analog input

OutBool	
CControls_BASR8M_Services::OutBool	
Instance	0
In	false
Online	false

Output Boolean — BASremote binary output

In = Boolean variable to be written to a real-world output

OutFloa	
CControls_BASR8M_Services::OutFloat	
Instance	6
In	0.0
Online	true

Output Float — BASremote analog output

In = Float variable to be written to a real-world output

OutFlo1	
CControls_BASR8M_Services::OutFloatCond	
Instance	0
In	0.0
Out	0.0
Enable	false
Online	false

Output Float Conditional — BASremote conditional analog output

In = Float variable to be written to a real-world output.

Out = Float value currently written to real world output.

Enable = Boolean which indicates whether a write should occur. True will allow the write to occur and False will inhibit any writes.

Sedona will, normally, write the outputs from its logic every cycle. This can be an issue for some Modbus registers controlled by the BASremote. The writes to these registers can be controlled via the enable signal. If enable is false, the Modbus register associated with this component will not be written.

SendEma	
CControls_BASR8M_Services::SendEmail	
EmailNumber	0
In	0.0
Enable	false

Send Email — BASremote email alert

In = Float value to be included in email.

Enable = Boolean used to indicate when to send an email.

Email Number = which email to send (it must match the web configuration).

The BASremote can send an email using this component when the Enable signal is true. The email must be configured in the configuration webpage of the BASremote. When the email is sent, the text of the email will contain the current input float value. One Email will be sent on the false-to-true transition of the Enable signal.

4.2   BAScontrol Unitary Controllers

BAScontrol Unitary Controllers Selection Guide

BAScontrol Model	Universal Inputs UI	Binary Inputs BI	Analog Outputs AO	Binary Outputs BO	Virtual Points AV/BV	Web Points WC	BACnet/IP Ethernet Ports	BACnet MS/TP Ports	BACnet Client
BAScontrol20 Firmware 3.1	8	4	4	4 Relay or Triac	24	48	1	0	No
BAScontrol22 Firmware 3.1	8	4	4	6	24	48	2	0	No
BAScontrol22C Firmware 3.2	8	4	4	6	24	48	2	0	Yes
BAScontrol22D Firmware 4.0	8	4	4	6	24	48	2	0	Yes
BAScontrol22S Firmware 4.0	8	4	4	6	24	48	1	1	Yes

Pre-installed Sedona Component Kits in BAScontrol Unitary Controllers: The following table identifies what Sedona component kits are factory installed (√), and what kits can be optionally installed (opt.) on Contemporary Controls' family of BAScontrol unitary controllers.

Sedona Component Kit Name	BASC-20R 20T	BASC-20CR	BASC-22R	BASC-22CR	BASC-22DR	BASC-22SR	BASC-22WR	BASC-22WSR
CControls_BASC20_IO	√							
CControls_BASC20_Platform	√							
CControls_BASC20_Web	√							
CControls_BASC20C_IO		√						
CControls_BASC20C_Platform		√						
CControls_BASC20C_Web		√						
CControls_BASCC_NETV		√		√	√	√	√	√
CControls_BASC22_IO			√					
CControls_BASC22_Platform			√					
CControls_BASC22_Web			√					
CControls_BASC22C_IO				√				
CControls_BASC22C_Platform				√				
CControls_BASC22C_Web				√				
CControls_BASC22D_IO					√			
CControls_BASC22D_Platform					√			
CControls_BASC22D_Web					√			
CControls_BASC22S_IO						√		
CControls_BASC22S_Platform						√		
CControls_BASC22S_Web						√		
CControls_BASC22W_IO							√	
CControls_BASC22W_Platform							√	
CControls_BASC22W_Web							√	
CControls_BASC22WS_IO								√

Sedona Component Kit Name								
CControls_BASC22WS_Platform								√
CControls_BASC22WS_Web								√
basicSchedule	√	√	√	√	√	√	√	√
datetimeSTD	√	√	√	√	√	√	√	√
func	√	√	√	√	√	√	√	√
hvac	√	√	√	√	√	√	√	√
logic	√	√	√	√	√	√	√	√
math	√	√	√	√	√	√	√	√
pricomp	√	√	√	√	√	√	√	√
sys	√	√	√	√	√	√	√	√
timing	√	√	√	√	√	√	√	√
types	√	√	√	√	√	√	√	√
CControls_Function	√	√	√	√	√	√	√	√
CControls_Function2	opt	opt	opt	opt	√	√	√	√
CControls_Math	opt	opt	opt	opt	√	√	√	√
CControls_Math2	opt	opt	opt	opt	√	√	√	√
CControls_HVAC	opt	opt	opt	opt	√	√	√	√
CControls_P_HVAC2	opt	opt	opt	opt	√	√	√	√

4.2.1   BAScontrol Platform Kits

Sedona Component Kit Name	BASC-20R 20T	BASC-20CR	BASC-22R	BASC-22CR	BASC-22DR	BASC-22SR	BASC-22WR	BASC-22WSR
CControls_BASC20_Platform	√							
CControls_BASC20C_Platform		√						
CControls_BASC22_Platform			√					
CControls_BASC22C_Platform				√				
CControls_BASC22D_Platform					√			
CControls_BASC22S_Platform						√		
CControls_BASC22WR_Platform							√	
CControls_BASC22WS_Platform								√

BASC22P
CControls_BASC22_Platform::BASC22PlatformService
PlatformId — ccontrols-BASC22-3.1.0
PlatformVer — BAScontrol 2.0.1
MemAvailable — 22848

The BAScontrol platform kit has only one component that advises the programmer how much usable memory is available for application programming. It is recommended that the usable memory not fall below 8,192 bytes. It can be found in the services folder and can be copied onto the wiresheet. The output type of this component is a Long.

4.2.2   BAScontrol I/O Kits

Sedona Component Kit Name	BASC-20R 20T	BASC-20CR	BASC-22R	BASC-22CR	BASC-22DR	BASC-22SR	BASC-22WR	BASC-22WSR
CControls_BASC20_IO	√							
CControls_BASC20C_IO		√						
CControls_BASC22_IO			√					
CControls_BASC22C_IO				√				
CControls_BASC22D_IO					√			
CControls_BASC22S_IO						√		
CControls_BASC22WR_IO							√	
CControls_BASC22WS_IO								√

The BAScontrol IO kits provide components necessary to interface Sedona logic to real world inputs and outputs. In addition to 20 or 22 real I/O points, these controllers accommodate 24 virtual points that can be treated as either inputs or outputs. Universal inputs and virtual points require configuration via a web browser. All components are platform dependent.

AO1 – AO4	Analog Output	analog voltage output points
BI1 – BI4	Binary Input	binary input points
BO1 – BO6	Binary Output	binary output points (B01-B04 on 20-point controllers)
UI1 – UI4	Universal Input	binary, analog voltage, thermistor, resistance, or accumulator
UI5 – UI8	Universal Input	binary, analog voltage, thermistor, or resistance
UC1 – UC4	Retentive Counters	up/down retentive universal counters
VT01 – VT24	Virtual Points	share data with BACnet/IP clients – VT01-VT08 are retentive
ScanTim	Scan Timer	monitors the time to execute Sedona logic
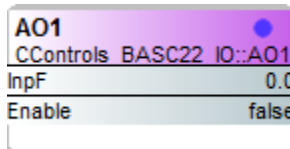
AO1 – AO4 Analog Output — analog voltage output point

Inp F = float value from 0–10 of respective point which translates to 0–10VDC output if Enable is true. If Enable is false, then output is controlled by a BACnet client.
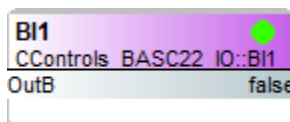
BI1 – BI4 Binary Input — binary input point

Out B is true if input point is asserted to common; otherwise Out B is false.

BO1 – BO6 Binary Output — binary output point (BO1-BO4 on 20-point units)

Inp B = Boolean value of respective point which will translate to either a contact closure or triac output (on triac models).
If Inp B and Enable are true, the contact closure is made or the triac is turned on. If Enable is false, then output is controlled by a BACnet client.

UI1 – UI4 Universal Input — binary, analog voltage, thermistor, resistance, or accumulator point

Each universal point must first be configured from the main web page. Click on the tag name for the universal point of interest which will then bring up a configuration page. Under BAS Channel Configuration use the Channel Type drop-down to select the channel type. The choices are as follows:

- 0-10 VDC analog voltage
- Binary input
- 10k thermistor type 2
- 10k thermistor type 3
- 20k thermistor
- 100K thermistor (Firmware 4.0)
- 1k-100kΩ resistance
- Pulse accumulator

If configured for analog input, thermistor, resistance or pulse accumulator, then;
Out F = float value of the input and Out I = the integer value of the input

If configured as a thermistor, or resistance, and an out-of-range condition is detected, Out F = the configured float Out of Bounds value, and Out I = the configured integer Out of Bounds value, and Out B = true to indicate a thermistor or resistance fault. The Out-of-Bounds value is first set on the point configuration web page.

If configured as a binary, then Out B = Boolean value of the input.
Out B is true if input point is asserted to common; otherwise Out B is false.

If configured as a pulse accumulator and Reset = false, then Out F = the accumulated float value, and Out I = the accumulated integer value.
If Reset = True, then Out F and Out I will equal zero.

UI5	●
CControls_BASC22_IO::UI5	
ChnType	Thm10KT3
OutF	2.5
OutB	true
OutI	2

UI6	●
CControls_BASC22_IO::UI6	
ChnType	Thm20K
OutF	2.5
OutB	true
OutI	2

UI7	●
CControls_BASC22_IO::UI7	
ChnType	Resistance
OutF	2.5
OutB	true
OutI	2

UI8	●
CControls_BASC22_IO::UI8	
ChnType	Input10V
OutF	0.00
OutB	false
OutI	0

UI5 – UI8 Universal Input — binary, analog voltage, thermistor, or resistance

Each universal point must first be configured from the main web page. Click on the tag name for the universal point of interest which will then bring up a configuration page. Under BAS Channel Configuration use the Channel Type drop-down to select the channel type. The choices are as follows:

- 0-10 VDC analog voltage
- Binary input
- 10k thermistor type 2
- 10k thermistor type 3
- 20k thermistor
- 100K thermistor (Firmware 4.0)
- 1k-100kΩ resistance

If configured for analog input, thermistor, or resistance, then;
Out F = float value of the input and Out I = the integer value of the input

If configured as a thermistor, or resistance, and an out-of-range condition is detected, Out F = the configured float Out of Bounds value, and Out I = the configured integer Out of Bounds value, and Out B = true to indicate a thermistor or resistance fault. The Out-of-Bounds value is first set on the point configuration web page.

If configured as a binary, then Out B = Boolean value of the input.
Out B is true if input point is asserted to common; otherwise Out B is false.

VT01 – VT24 Virtual Points — wiresheet read, or wiresheet write

Virtual points are used to share wiresheet data with a BACnet/IP client. A BACnet/IP client can "read" wiresheet data such as a calculated value or it can "write" to the wiresheet with a set-point or command. Virtual points are first configured from a web page to be a BACnet binary value (BV) or BACnet analog value (AV). The BACnet description field and units of measure can be set as well as the BACnet name which must be unique within the device. Next go to the Sedona tool to change the wiresheet Read or Write directions to either *InputTo* or *OutputFrom*. The title of the virtual point on the web page will change to *Write to Wiresheet* or *Read from Wiresheet* accordingly. These terms are from the perspective of a BACnet client. The four possibilities are shown on the left labelled as VT01 through VT04.

VT01	
CControls_BASC22_IO::VT01	
Initialized	true
ChnType	FloatInput
Reset	false
FloatV	0.0
BinaryV	false
WireSheet	InputTo

VT01 is configured as analog variable, Write to Wiresheet, which results in the component being a *FloatInput*. *Float V* represents the value written by the BACnet/IP client to the wiresheet.

VT02	
CControls_BASC22_IO::VT02	
Initialized	true
ChnType	FloatOutput
Reset	false
FloatV	43.0
BinaryV	true
WireSheet	OutputFrom

VT02 is configured as analog variable, Read from Wiresheet, which results in the component being a *FloatOutput*. *Float V* represents a value that can be read by the BACnet/IP client from the wiresheet.

VT03	
CControls_BASC22_IO::VT03	
Initialized	true
ChnType	BinaryInput
Reset	false
FloatV	0.0
BinaryV	false
WireSheet	InputTo

VT03 is configured as binary variable, Write to Wiresheet, which results in the component being a *BinaryInput*. *Binary V* represents the value written by the BACnet/IP client to the wiresheet.

VT04	
CControls_BASC22_IO::VT04	
Initialized	true
ChnType	BinaryOutput
Reset	false
FloatV	0.0
BinaryV	false
WireSheet	OutputFrom

VT04 is configured as binary variable, Read from Wiresheet, which results in the component being a *BinaryOutput*. *Binary V* represents a value that can be read by the BACnet/IP client from the wiresheet.

Note: VT01 through VT08 will retain their values after a power outage from one to seven days depending upon controller ambient temperature.

ScanTim	⏲
CControls_BASC22_IO::ScanTim	
SampleSize	10
TimeMs	6
MinimumMs	4
MaximumMs	7
AverageMs	6
Reset	false

ScanTim – monitors the execution time of Sedona logic

The scan timer monitors the current, minimum, maximum and average time it takes to execute a single scan of Sedona logic.

All outputs are integers. The average time is based upon the last ten samples. The result of which becomes the first value in the next ten samples. The component can be reset by right clicking the component and invoking an Action.

UC1	⚡
CControls_BASC22_IO::UC1	
Initialized	true
Count	0
CountF	0.0
Ovf	true
Clk	false
Enable	true
Rst	false
CDwn	false
Limit	0
HoldAtLimit	false

UC1 – UC4 — retentive up/down universal counters

Counts on the false to true transition of *Clk* if *Enable* is *true*. If *C Dwn* is *true*, counting is down until zero is reached. If *C Dwn* is *false*, counting is up to the limit of the counter (2147483647) before it rolls over to zero. If *Hold At Limit* is set to *true*, the counter will stop counting at the value set in the *Limit* slot on the property page. The *Ovf* flag is set *true* when the value of status equals or exceeds the limit value. The output *count* value can be displayed as an integer (*Count*) or a float (*Count F*). *Rst* set to *true* clears the counter and prevents further counting.

Note: VT01 through VT08 will retain their values after a power outage from one to seven days depending upon controller ambient temperature.

4.2.3 BAScontrol Web Kits

Sedona Component Kit Name	BASC-20R 20T	BASC-20CR	BASC-22R	BASC-22CR	BASC-22DR	BASC-22SR	BASC-22WR	BASC-22WSR
CControls_BASC20_Web	√							
CControls_BASC20C_Web		√						
CControls_BASC22_Web			√					
CControls_BASC22C_Web				√				
CControls_BASC22D_Web					√			
CControls_BASC22S_Web						√		
CControls_BASC22WR_Web							√	
CControls_BASC22WS_Web								√

Web components provide a convenient method of sharing data between web pages and the wiresheet. In these kits there are 48 web components that must be first configured on the wiresheet. Web components can be configured to input data from the web page or output binary, integer, or float data to the web page. In input mode, the range of the input can be restricted when entering values using a web browser. In output mode there is no range limitation. The description of the point can be set using a web browser. The four possibilities are shown labelled as WC01 through WC04.

WC01 is configured as an input which results in the component being an *Input* from the web page. The range of the input must be within *MinVal* and *MaxVal* which can be configured in the component. An input value greater or equal to 1 is treated as a *BinVal* of *true,* otherwise *false. IntVal* will display the truncated integer value of a float input while *FltVal* will display the actual float value.

WC02 is configured as an output float which results in the component being a *FloatOutput* to the web page. The value of *FltVal* will appear in the web page. *MinVal* and *MaxVal* are ignored.

WC03	
CControls_BASC22_Web::WC03	
WcType	Input
MinVal	0.0
MaxVal	100.0
FltVal	0.0
IntVal	0
BinVal	false

WC03 is configured as output integer which results in the component being an *IntegerOutput to the web page.* The value of *FltVal* will appear in the web page. *MinVal* and *MaxVal* are ignored.

WC04	
CControls_BASC22_Web::WC04	
WcType	Input
MinVal	0.0
MaxVal	100.0
FltVal	0.0
IntVal	0
BinVal	false

WC04 is configured as an output binary which results in the component being a *BinaryOutput* to the web page. The value of *BinVal* will appear in the web page. *MinVal* and *MaxVal* are ignored.

4.2.4   BAScontrol Network Variable Kit

Sedona Component Kit Name	BASC-20R 20T	BASC-20CR	BASC-22R	BASC-22CR	BASC-22DR	BASC-22SR	BASC-22WR	BASC-22WSR
CControls_BASCC_NETV		√		√	√	√	√	√

Controllers with BACnet client functionality are capable of reading from and writing to BACnet devices on either an IP or MS/TP network by using NetV (Network Variable) Sedona components. NetV Sedona components are only functional when installed on devices with BACnet client capability. NetV components provide the method for reading BACnet objects from remote BACnet server devices onto the client controller's wiresheet, or for writing to BACnet objects on the remote BACnet server from the client controller's wiresheet. The BAScontrol20C and BAScontrol22C (Firmware 3.2) have five NetV components from the factory—NetV, NetVAI4, NetVAO4, NetVBI4, and NetVBO4. The BAScontrol22D, BAScontrol22S, BAScontrol22W, and BAScontrol22WS (Firmware 4.0) have two additional components—NETVBV4 and NETVAV4. To configure a NetV component, drag and drop it on the wiresheet. The NetV component must have the same *Device Instance* as previously configured on the *Configure BACnet Servers* web page as well as the correct *Object Instance* of the target device object(s). For detailed instructions on setting up the BACnet servers on the client controller web page, see the product documentation available on the client controller's product web page.

```
NetV
CControls_BASCC_NETV::NetV          ●
DevInstance                      13178
ObjInstance                          1
ObjType                    AnalogInput
Priority                            10
DefOutF                            0.0
DefOutB                          false
DefOutI                              0
ValF                             83.01
ValB                              true
ValI                                83
Relinquish                       false
Enabled                          false
Status                          Online
```

NetV is the most universal component in the kit which can be configured to read or write a single point (object) on the target BACnet server device.

DevInstance slot must have the correct target device instance configured.

ObjectInstance slot must be configured with the correct target device object instance to be read/written.

ObjType can be of any of the following BACnet object types and configured as Analog Input (AI), Analog Output (AO), Analog Value (AV), Binary Input (BI), Binary Output (BO), or Binary Value (BV). Accumulator (ACC), Multistate Input (MSI), Multistate Output (MSO), or Multistate Value (MSV). ACC, MSI, MSO and MSV are only available with the BAScontrol Firmware 4.0 series controllers. If the point configured in the NetV is a writable point such as AO, AV, BO, or BV, ACC, MSO, or MSV, the *Enabled* slot must be set to *true,* if the object is read only such as AI or BI – set the *Enabled* slot to *false*.

Priority slot lets you configure the BACnet write priority for writable objects with range of 1 through 16 and default value of 10.

DefOutF, *DefOutB*, and *DefOutI* are slots used for setting a safe default value for an output or input in Float, Bool, and Integer data types. If the NetV object is offline, you need to be able to put the system Sedona logic in a safe state using these slots.

ValF, *ValB*, and *ValI* read the target object present value property when *Enabled* slot is set to *false* and command (write) the target object present value property when *Enabled* slot is set to *true*.

Relinquish slot is used to release the writable target object by removing the write and setting the specified write priority of the configured target object to *NULL* when triggered to *true*. The write to the target object at the specified priority is applied again when the *Relinquish* slot is toggled back to *false*.

Status slot indicates the *Online* or *Offline* state for the target object.

```
NETVAI4
CControls_BASCC_NETV::NETVAI4       ●
DevInstance                      13178
Inp1Instance                         1
Inp1Use                        NotUsed
Inp2Instance                         2
Inp2Use                          Input
Inp3Instance                         3
Inp3Use                          Input
Inp4Instance                         4
Inp4Use                          Input
Inp1                               0.0
Inp2                             84.23
Inp3                             84.08
Inp4                             83.27
Status                          Online
```

NETVAI4 component allows reading of up to 4 BACnet objects on the remote BACnet server device of type Analog Input (AI).

DevInstance slot must have the correct target device instance.

Inp1Instance, *Inp2Instance*, *Inp3Instance*, and *Inp4Instance* slots must be configured with the correct target device object instances to be read.

Inp1Use, *Inp2Use*, *Inp3Use*, and *Inp4Use* are slots which allow you to set the use for each of the 4 objects. Since the NETVAI4 is a read only component, the two options are *Input* and *Not Used*. Slots configured as *NotUsed* will not consume RAM memory.

Inp1, *Inp2*, *Inp3*, and *Inp4* slots display the target object present value property read from the target BACnet server device.

Status slot indicates the *Online* or *Offline* state for the target object.

```
NETVAO4
CControls_BASCC_NETV::NETVAO4      ●
DevInstance                    13178
Out1Instance                      13
Out1Use                        Input
Out1Priority                      10
Out2Instance                      14
Out2Use                       Output
Out2Priority                      10
Out3Instance                      15
Out3Use                        Input
Out3Priority                      10
Out4Instance                      16
Out4Use                       Output
Out4Priority                      10
Out1                             4.5
Relinquish1                    false
Out2                             0.0
Relinquish2                    false
Out3                            8.15
Relinquish3                    false
Out4                             0.0
Relinquish4                    false
Status                        Online
```

NETVAO4 component allows reading or writing of up to 4 BACnet objects of type Analog Output (AO). When the *OutUse* slot is configured for *Input,* the component will read the target device AO object present value property. When the *OutUse* slot is configured for *Output,* the component will write to the target device AO object present value property. When the *OutUse* slot is configured for *NotUsed,* the slot is not in use. Slots configured as *NotUsed* will not consume RAM memory.

OutPriority slots let you configure the BACnet write priority for writable target objects with range 1 through 16 and default value of 10.

Out1, Out2, Out3, and *Out4* slots read the target object present value property when *OutUse* slot is set to *Input* and command (write) to the target object present value property when *OutUse* slot is set to *Output.*

Relinquish slot is used to release the writable target object by removing the write and setting the specified write priority of the configured target object to *NULL* when triggered to *true.* The write to the target object at the specified priority is applied again when the *Relinquish* slot is toggled back to *false.*

Status slot indicates the *Online* or *Offline* state for the target object.

```
NETVAV4
CControls_BASCC_NETV::NETVAV4      ●
DevInstance                     1234
Out1Instance                     244
Out1Use                        Input
Out1Priority                      10
Out2Instance                     243
Out2Use                        Input
Out2Priority                      10
Out3Instance                     242
Out3Use                       Output
Out3Priority                      10
Out4Instance                     241
Out4Use                       Output
Out4Priority                      10
Out1                            17.1
Relinquish1                    false
Out2                           68.15
Relinquish2                    false
Out3                            98.0
Relinquish3                    false
Out4                            19.0
Relinquish4                    false
Status                        Online
```

NETVAV4 component allows reading or writing of up to 4 BACnet objects of type Analog Value (AV). When the *OutUse* slot is configured for *Input,* the component will read the target device AV object present value property. When the *OutUse* slot is configured for *Output,* the component will write to the target device AV object present value property. When the *OutUse* slot is configured for *NotUsed,* the slot is not in use. Slots configured as *NotUsed* will not consume RAM memory.

OutPriority slots let you configure the BACnet write priority for writable target objects with range 1 through 16 and default value of 10.

Out1, Out2, Out3, and *Out4* slots read the target object present value property when *OutUse* slot is set to *Input* and command (write) to the target object present value property when *OutUse* slot is set to *Output.*

Relinquish slot is used to release the writable target object by removing the write and setting the specified write priority of the configured target object to *NULL* when triggered to *true.* The write to the target object at the specified priority is applied again when the *Relinquish* slot is toggled back to *false.*

Status slot indicates the *Online* or *Offline* state for the target object.

NETVBI4	
CControls_BASCC_NETV::NETVBI4	●
DevInstance	13178
Inp1Instance	9
Inp1Use	Input
Inp2Instance	10
Inp2Use	Input
Inp3Instance	11
Inp3Use	Input
Inp4Instance	12
Inp4Use	Input
Inp1	false
Inp2	true
Inp3	false
Inp4	false
Status	Online

NETVBI4 component allows reading of up to 4 BACnet objects of type Binary Input (BI).

DevInstance slot must have the correct target device instance.
Inp1Instance, *Inp2Instance*, *Inp3Instance*, and *Inp4Instance* slots must be configured with the correct target object instances to be read.

Inp1Use, *Inp2Use*, *Inp3Use*, and *Inp4Use* are slots which allow you to set the use for each of the 4 objects. Since the NETVBI4 is a read only component, the two options are *Input* and *Not Used*. Slots configured as *NotUsed* will not consume RAM memory.

Inp1, *Inp2*, *Inp3*, and *Inp4* slots display the target object present value property read from the target BACnet server device.

Status slot indicates the *Online* or *Offline* state for the target object.

NETVBO4	
CControls_BASCC_NETV::NETVBO4	●
DevInstance	13178
Out1Instance	17
Out1Use	Input
Out1Priority	10
Out2Instance	18
Out2Use	Output
Out2Priority	10
Out3Instance	19
Out3Use	Output
Out3Priority	10
Out4Instance	20
Out4Use	Input
Out4Priority	10
Out1	true
Relinquish1	false
Out2	false
Relinquish2	false
Out3	false
Relinquish3	false
Out4	true
Relinquish4	false
Status	Online

NETVBO4 component allows reading or writing of up to 4 BACnet objects of type Binary Output (BO). When the *OutUse* slot is configured for *Input*, the component will read the target device BO or BV object present value property. When the *OutUse* slot is configured for *Output*, the component will write to the target device BO object present value property. When the *OutUse* slot is configured for *NotUsed*, the slot is not in use. Slots configured as *NotUsed* will not consume RAM memory.

OutPriority slots let you configure the BACnet write priority for writable objects with range 1 through 16 and default value of 10.

Out1, *Out2*, *Out3*, and *Out4* slots read the target object present value property when *OutUse* slot is set to *Input* and command (write) to the target object present value property when *OutUse* slot is set to *Output*.

Relinquish slot is used to release the writable target object by removing the write and setting the specified write priority of the configured target object to *NULL* when triggered to *true*. The write to the target object at the specified priority is applied again when the *Relinquish* slot is toggled back to *false*. The *Status* slot indicates the *Online* or *Offline* state for the target object.

NETVBV4	
CControls_BASCC_NETV::NETVBV4	●
DevInstance	1234
Out1Instance	247
Out1Use	Input
Out1Priority	10
Out2Instance	248
Out2Use	Output
Out2Priority	10
Out3Instance	246
Out3Use	Input
Out3Priority	10
Out4Instance	245
Out4Use	Output
Out4Priority	10
Out1	true
Relinquish1	false
Out2	false
Relinquish2	false
Out3	false
Relinquish3	false
Out4	true
Relinquish4	false
Status	Online

NETVBV4 component allows reading or writing of up to 4 BACnet objects of type Binary Value (BV). When the *OutUse* slot is configured for *Input,* the component will read the target device BV object present value property. When the *OutUse* slot is configured for *Output*, the component will write to the target device BV object present value property. When the *OutUse* slot is configured for *NotUsed*, the slot is not in use. Slots configured as *NotUsed* will not consume RAM memory.

OutPriority slots let you configure the BACnet write priority for writable objects with range 1 through 16 and default value of 10. *Out1*, *Out2*, *Out3*, and *Out4* slots read the target object present value property when *OutUse* slot is set to *Input* and command (write) to the target object present value property when *OutUse* slot is set to *Output*.

Relinquish slot is used to release the writable target object by removing the write and setting the specified write priority of the configured target object to *NULL* when triggered to *true*. The write to the target object at the specified priority is applied again when the *Relinquish* slot is toggled back to *false*. The *Status* slot indicates the *Online* or *Offline* state for the target object.

plat	
CControls_BASC22D_Platform::BASC22DPlatformService	▤
PlatformId	ccontrols-BASC22D-4.0.0
PlatformVer	BAScontrol 4.0.0
MemAvailable	33064

Client controllers can only keep up with a limited number of BACnet server devices/points because it is not intended for use as a building/supervisory controller. Each point polled by the client controller costs RAM memory which in turn is a trade-off as to how large the Sedona wiresheet logic in the controller can be. The client controller starts out with up to 33Kbytes of RAM memory in the BAScontrol22D. Activating the BACnet client with the first BACnet object uses 2Kbytes of RAM memory. Each additional BACnet object uses 300 bytes of RAM memory. Using NETVAI4, NETVBI4, NETVAO4, NETVBO4, NETVAV4, and NETVBV4 components optimizes memory use by reading or writing 4 objects at a time with one component versus using a single NetV component for a single BACnet object. Client controller RAM memory can be monitored in real time by use of the *plat* Sedona component located in the *Service* folder or the *Platform Service* component dragged and dropped from the *sys* kit anywhere on the wiresheet. The minimum *MemAvailable* value is 8192bytes (about 8Kbytes) needed for the controller to operate. Once this minimum value is reached no more components can be placed on the wiresheet. The Sedona Application Editor will prevent you from doing so by immediately erasing a dropped component when the minimum *MemAvailable* value is reached.

5 Contemporary Controls' Platform Independent Kits

5.1   Function Kit (CControls_Function)

What follows are component descriptions for Sedona components in the CControls_Function kit. This is a custom hardware independent kit that can be used with any Sedona 1.2.28 platform.

Cand2	
CControls_Function::Cand2	
Inp1	false
Inp2	false
Out	false
OutNot	true

Two-input Boolean product – two-input AND/NAND gate

$Out = In1 \bullet In2$

$Out\ Not = \overline{Out}$

Cand4	
CControls_Function::Cand4	
Inp1	false
Inp2	false
Inp3	false
Inp4	false
Out	false
OutNot	true

Four-input Boolean product – four-input AND/NAND gate

$Out = In1 \bullet In2 \bullet In3 \bullet In4$

$Out\ Not = \overline{Out}$

Cand6	
CControls_Function::Cand6	
Inp1	false
Inp2	false
Inp3	false
Inp4	false
Inp5	false
Inp6	false
Out	false
OutNot	true

Six-input Boolean product – six-input AND/NAND gate

$Out = In1 \bullet In2 \bullet In3 \bullet In4 \bullet In5 \bullet In6$

$Out\ Not = \overline{Out}$

Cand8	
CControls_Function::Cand8	
Inp1	false
Inp2	false
Inp3	false
Inp4	false
Inp5	false
Inp6	false
Inp7	false
Inp8	false
Out	false
OutNot	true

Eight-input Boolean product – eight-input AND/NAND gate

$Out = In1 \bullet In2 \bullet In3 \bullet In4 \bullet In5 \bullet In6 \bullet In7 \bullet In8$

$Out\ Not = \overline{Out}$

Cor2
CControls_Function::Cor2

Inp1	false
Inp2	false
Out	false
OutNot	true

Two-input Boolean sum – two-input OR/NOR gate

$Out = In1 \mid In2$

$Out\ Not = \overline{Out}$

Cor4
CControls_Function::Cor4

Inp1	false
Inp2	false
Inp3	false
Inp4	false
Out	false
OutNot	true

Four-input Boolean sum – four-input OR/NOR gate

$Out = In1 \mid In2 \mid In3 \mid In4$

$Out\ Not = \overline{Out}$

Cor6
CControls_Function::Cor6

Inp1	false
Inp2	false
Inp3	false
Inp4	false
Inp5	false
Inp6	false
Out	false
OutNot	true

Six-input Boolean sum – six-input OR/NOR gate

$Out = In1 \mid In2 \mid In3 \mid In4 \mid In5 \mid In6$

$Out\ Not = \overline{Out}$

Cor8
CControls_Function::Cor8

Inp1	false
Inp2	false
Inp3	false
Inp4	false
Inp5	false
Inp6	false
Inp7	false
Inp8	false
Out	false
OutNot	true

Eight-input Boolean sum – eight-input OR/NOR gate

$Out = In1 \mid In2 \mid In3 \mid In4 \mid In5 \mid In6 \mid In7 \mid In8$

$Out\ Not = \overline{Out}$

Cmt
CControls_Function::Cmt

| Comment | 123456789012345678901234567890123456789012345678901234 |

Comment

A comment field from 1-64 characters used for documentation purposes.

Dff
CControls_Function::Dff

Preset	false
Reset	false
D	false
Clk	false
Out	false
OutNot	true

"D" Flip-Flop – D-style Edge-triggered Single-bit Storage

If Preset = True and Reset = False then Out = True

If Reset = True then Out = False regardless of all other inputs

On the rising edge of Clk with Preset = False and Reset = False;

If D = false then Out = false

If D= true then Out = true

Out Not = $\overline{Out}$

CtoF
CControls_Function::CtoF

InTempDegC	0.0
OutTempDegF	32.0

°F to °C – Fahrenheit to Celsius Temperature Conversion

Out = 9/5 ∗ In + 32

FtoC
CControls_Function::FtoC

InTempDegF	0.0
OutTempDegC	-17.77

°C to °F – Celsius to Fahrenheit Temperature Conversion

Out = 5/9 ∗ (In − 32)

HLpre
CControls_Function::HLpre

Out	true
OutNot	false

High – Low Preset – defined logical true and false states

Out = true

Out Not = false

PsychrE
CControls_Function::PsychrE

InTempDegF	0.0
InRelativeHumidityPct	0.0
OutDewPointDegF	0.0
OutEnthalpyBtu_per_lb	0.0
OutSatPressure_psi	0.0
OutVaporPressure_psi	0.0
OutWetBulbTempDegF	0.0

Psychrometric Calculator – English Units

Inputs are Dry-bulb temperature (°F) and Relative Humidity (%) Outputs are Dew Point (°F), Enthalpy (Btu/lb), Saturation Pressure (psi), Vapor Pressure (psi) and Wet-bulb temperature (°F)

Input temperature range 32-120°F; Input relative humidity range 10-100%

PsychrS
CControls_Function::PsychrS

InTempDegC	0.0
InRelativeHumidityPct	0.0
OutDewPointDegC	0.0
OutEnthalpy_kJ_per_kg	0.0
OutSatPressure_kPa	0.0
OutVaporPressure_kPa	0.0
OutWetBulbTempDegC	0.0

Psychrometric Calculator – SI Units

Inputs are Dry-bulb temperature (°C) and Relative Humidity (%) Outputs are Dew Point (°C), Enthalpy (kJ/kg), Saturation Pressure (kPa), Vapor Pressure (kPa) and Wet-bulb temperature (°C)

Input temperature range 0-48.8 °C; Input relative humidity range 10-100%

Simplified Psychrometric Chart

Wet-Bulb Temperature
Saturation Line
Relative Humidity
Enthalpy
Dew Point Temperature
Humidity Ratio
Dry-Bulb Temperature
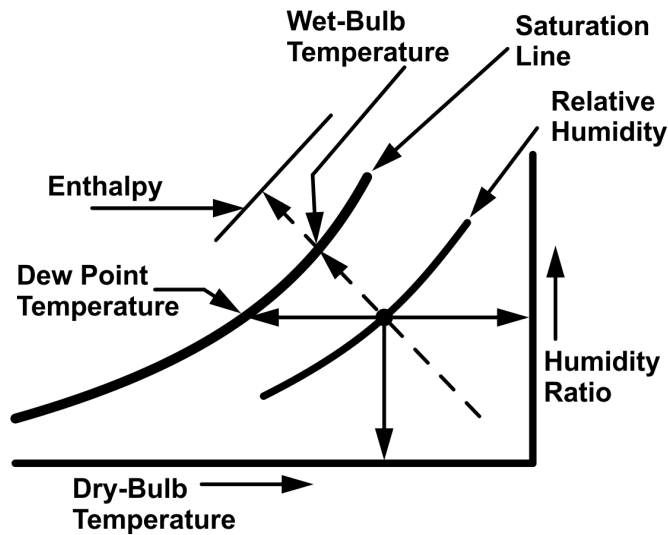
A simplified psychrometric chart greatly removes the detail of a professional chart. On the X-axis is the dry-bulb temperature with a typical range from 32°F to 120°F. This is the same temperature you measure with a thermometer or wall-mounted thermostat. Lines of constant dry-bulb temperature are for all practical purposes vertical. On the Y-axis is the humidity ratio (lbw/lba) in lbs-water vapor to lbs-air ranging from zero to over 0.028. Lines of constant humidity ratio are horizontal. The left curved heavy line is called the saturation line indicating 100% saturation of water vapor or 100% relative humidity. Curves of lesser relative humidity would exist to the right of the saturation line. Along the saturation line you can determine both dew point temperature and wet-bulb temperature although their lines of constant temperature are different. For dew point, the lines are horizontal while the lines of constant wet-bulb are diagonal and almost parallel with lines of constant enthalpy.

Looking at the PsychrE component and the simplified chart we can study one example. Notice in the component that the two inputs are 70°F dry-bulb and 50% relative humidity. With these two values a single point on the psychrometic chart can be located. If you follow the horizontal line to the left you can determine the dew point temperature and to the right the humidity ratio. If you follow the diagonal line to the upper-left you can learn the wet-bulb and enthalpy values. We still have not determined the saturation pressure or the vapor pressure but these values can be derived with help from the humidity ratio. The PsychrE can make the calculations in the English system and the PsychrS can make the calculations in the SI system. Although simple conversions can be made between the two systems or to reflect the output values in different units of measure, be careful when working with enthalpy. With the English system, the change in enthalpy is referenced to a 0°F while in the SI system the reference is 0°C so a straight forward conversion between the two systems is not possible. Also note the limited range of the two psychrometric components. Both components are limited to an equivalent input range of 0-120°F dry-bulb and 10-100% relative humidity.

Set/Clear Latch – single-bit level-triggered single-bit data storage

The following logic applies to the state of Set or Clear:

If Set is true and Clear is false, then Out = true

If Clear is true, then Out = false regardless of the state of Set

Out Not = $\overline{Out}$

5.2   HVAC Kit (CControls_HVAC)

What follows are component descriptions for Sedona components in the CControls_HVAC kit. This is a custom hardware independent kit that can be used with any Sedona 1.2.28 platform.

AnaHiLo	⬨
CControls_HVAC::AnaHiLo	
LimitDelay	1.0
HiLimit	10.0
LoLimit	-10.0
Differential	0.1
HoldAtLimit	false
LimitOutEnable	false
In	0.0
Out	0.0
OverLimit	false
UnderLimit	false

Analog High/Low (AnaHiLo)

This component has two independent usages. It can be used to initiate an alarm or event for an analog point, or it could be used to limit the output range of an analog point. Both capabilities can be used at the same time.

If HoldAtLimit is false, the output (Out) will follow the input (In). If HoldAtLimit is true, Out will be clamped to the value

of HiLimit if the input exceeds this high limit or clamped to the value of LoLimit if the input becomes less than the low limit. Both HiLimit and LoLimit are configurable float values while HoldAtLimit is a configurable Boolean value.

This same component can be used to create an alarm or an event independent of the setting of HoldAtLimit. If LimitOutEnable is true, then OverLimit will become true if In remains above the high limit for a time greater or equal than the value set in configurable float slot LimitDelay. Once OverLimit is true, it will remain true until In decreases below the high limit by an amount greater or equal than the value set in configurable float slot Differential. A similar action will occur if In remains below the low limit for a time greater or equal than the value set in LimitDelay. Once UnderLimit is true, it will remain true until In increases above the low limit by an amount equal or greater than the value set in Differential.

If the configurable slot LimitOutEnable is false, then both OverLimit and UnderLimit are false.

AntiSCY	⬨
CControls_HVAC::AntiSCY	
MinRunTime	1.0
MinOffTime	1.0
In	false
Out	false
Reset	false

Anti-Short Cycle (AntiSCY)

This component is used to protect mechanical equipment from running for too short of a time or restarting without a sufficient delay. Usually used with compressors, it can be used to protect any HVAC piece of equipment.

There are two configurable time slots – MinRunTime and MinOffTime. Both are set in units of seconds. Assume Reset is false. Once the command signal to the input slot

(In) transitions from a false to true state, the output (Out) will immediately become true assuming that the previous command signal did not go false and then true within the time set in MinOffTime. Out will remain true for the entire time that the command signal is asserted and will continue to remain in the true state after the command signal returns to false for the amount of time set in MinRunTime. Once Out transitions to the false state, it will remain in the false state for an amount of time that is set in the MinOffTime slot regardless of the state of In. Once this timer times out, the In slot will be re-enabled.

If Reset is true, Out becomes false regardless of the state of In, or the state of the minimum run timer. All timers are cleared. Normal component operation occurs once Reset returns to the false state.

BTU/Hour Calculator (BTUh)

```
BTUh                    ⬛
CControls_HVAC::BTUh
ExeDelay            0.0
OffCal              0.0
InGPM               0.0
InTemp              0.0
OutTemp             0.0
Out                 0.0
TonOutR             0.0
TonOutC             0.0
```

This component makes it easy to calculate the amount of heating or cooling of water that is occurring by monitoring the temperature differential across a piece of equipment and the flow rate. Although the result is BTU per hour, the industry would just say BTU.

This component executes the equation:

BTU/hr = 500*flow rate *(Outlet temperature–Inlet temperature)

TonOutR = GPM/12,000

TonOutC = GPM/15,000

Where;

Flow rate is in gallons per minute

Temperatures are in °F

The output slot (Out) follows the equation above with inputs of flow rate (InGPM), Outlet temperature (OutTemp) and Inlet temperature (InTemp). In order to ensure a positive result, the temperature differential is treated as an absolute number. OffCal is a configurable float slot that will add positive or negative bias to the calculation if the equation needs to be tweaked. ExeDelay is a configurable float that specifies the time between calculations. TonOutR reflects the refrigeration tons conversion from BTU/Hr. while TonOutC reflects cooling tower tons conversion. InGPM is a configurable float that can be pre-set for constant flow calculations.

Numeric Dampener (NumDamp)

```
NumDamp                 ⬛
CControls_HVAC::NumDamp
UpdateInterval      5.0
RiseIncrement       0.5
FallDecrement       0.5
RiseDampInhibit   false
FallDampInhibit   false
In                  0.0
Out                 0.0
```

This component functions as a digital filter to dampen the volatility of an input signal by creating an output version of the input signal but with modifications to the signal's rate-of-change, and amplitude change. Although the output (Out) will attempt to track the input (In), the rate-of-change and amplitude of the output will lag the input depending upon configurable settings.

UpdateInterval is a configurable float in units of seconds. It sets the update rate of the output. The output will not change until the update interval is satisfied. If sampling is to be instantaneous, set Updateinterval to zero.

RiseIncrement and FallIncrement are also configurable floats that set the incremental rising rate and falling rate of the output respectively. The output will not change more than the increments entered assuming that rise and fall rate inhibits are false.

There are two configurable Boolean inputs RiseDampInhibit and FallDampInhibit. In the false state, incremental changes to the output are allowed for rising and falling outputs. Depending upon which of these two inhibits are set to true, the rising (or falling) rates will not be modified by the component. These two inhibits do not impact the sampling rate.

If UpdateInterval is zero and both RiseDampInhibit and FallDampInhibit are true, Out will equal In.

Enhanced PID Loop Controller (EnhPID)

```
EnhPID                    ●
CControls_HVAC::EnhPID
Enable                true
Sp                     0.0
Cv                     0.0
Out                    0.0
Kp                     1.0
Ki                     0.0
Kd                     0.0
Max                  100.0
Min                    0.0
Bias                   0.0
MaxDelta               0.0
Direct                true
ExTime                1000
```

This component appears as an exact replica of the LP component in Tridium's Func kit but it has one significant change in the way it handles the configurable Boolean slot Enable. When Enable is false in the LP slot, the LP component ceases to function and holds its output (Out) at its last state. When Enable returns to true, the LP picks up where it left off. This may not be acceptable in all applications.

With the Enhanced PID Loop Controller, if Enable becomes false, proportional control ceases and the internal timers and accumulators for integral and derivative action are cleared. Out will return to the Bias setting at a rate based upon MaxDelta. When Enable returns to true, the component returns to normal operation as if it began from a power-up condition.

If the intended application does not programmatically utilize the Enable slot, there is no need to use the Enhanced PID Loop Controller over the LP component. However, if this is not the case the action of the LP component under a disable state should be studied for acceptability.

To learn more about Enhanced PID Loop Controller, study the description for the LP component which is included in Tridium's 1.2 Func kit.

Lead Lag Sequence Controller (LeadLag)

```
LeadLag                   ◥
CControls_HVAC::LeadLag
RunTime               10.0
ProofDelay             1.0
OverlapTime            0.0
OutQty                 Two
In                    false
OutA                  false
OutB                  false
OutC                  false
OutD                  false
ProofA                false
ProofB                false
ProofC                false
ProofD                false
Alarm                 false
```

This component monitors and controls up to four devices (usually pumps) dedicated to one critical process. If a pump fails to come on-line or goes off-line when it should be on-line, the next-in-line pump will take over. In order to even out running times on all pumps, the next-in-line pump is selected on a time basis but before the primary pump (lead pump) is taken off-line, the next-in-line pump (lag pump) must be proven to be on-line before the changeover. Configuration can be for 2, 3 or 4 pumps.

RunTime is a configurable float that sets the desired run time for all pumps. Its time is in minutes. ProofDelay is a configurable float that sets the time limit before declaring a commanded pump has failed to come on line. OverlapTime is a configurable float that sets the time where two pumps will run together during the proving time and beyond. The last two slots are in seconds. OutQty is a configurable multistate that can be two, three or four. It sets the pump sequence to A-B, A-B-C or A-B-C-D before repeating. Each pump in the sequence must have a proving signal such as an auxiliary switch on a motor contactor, current switch or a flow switch. These are applied to ProofA through ProofD. Only those proving signals used in the sequence need to be connected.

A single command signal set to true is applied to the input (In) and starts the sequence. If this signal goes false all pumps stop. Assuming it is pump A's turn to come on, its proving signal is monitored to verify that it is on-line and will continuously be monitored throughout its run time cycle. At the end of run time, the lag pump (pump B) is commanded on and its proving signal is monitored for activation. If proving occurs, the lead pump will shut down after the overlap time and the lag pump now becomes the lead pump (pump B). If the lag pump

fails to come on, the Alarm slot will become true. The next-in-line pump (pump C – in a three or four pump sequence) is commanded on and its proving signal is monitored for activation. If it is proven, the Alarm slot will return to false. If a lead pump failure is sensed during run time, the Alarm slot becomes true, and the lag pump will be commanded on and proving initiated. Upon successful proving the Alarm slot will be cleared. The sequence continues to repeat based on the run time setting in the component.

Outside Air True Blend (OATrueB)

This component makes a dynamic calculation of the percentage opening of an outside air damper which is helpful when creating economizers or for demand control ventilation to reduce CO2. In order to use this component, it will be necessary to monitor outside-air temperature (OAT), mixed-air temperature (MAT) and return-air temperature (RAT).

OATrueB	
CControls_HVAC::OATrueB	
ExeDelay	1.0
OffCal	0.0
OutsideAT	0.0
ReturnAT	0.0
MixedAT	0.0
Output	0.0
Fault	true

The component solves the following equation for Outside Air %:

OAT% = 100 ∗ (MAT-RAT)/OAT-RAT)

Where; MAT, RAT, and OAT are in either °C or °F.

The output (Out) is OAT%. Execution delay (ExeDelay) is the delay between calculations in seconds. OffCal is a bias that can be applied to the output calculation to tweak the result. OutsideAT, ReturnAT and MixedAT are OAT, RAT and MAT respectively.

The value of MAT should always be between OAT and RAT and therefore the Output value should be between 0 and 100%.

If the calculated output is below zero, the Output value will be clamped at zero and the Fault slot will be set to true. Likewise, if the calculated output exceeds 100 then the Output will be clamped to 100 and the Fault slot will be set to true. If OAT=RAT then the Output will be zero and the Fault slot will be set to true.

Run Proving (RnProof)

RnProof	
CControls_HVAC::RnProof	
ProofDelay	1.0
In	false
Proof	false
Out	false
OutNot	true
Fail	false
FailInhibit	false

This component is a simple adaptation of the more complex LeadLag component. With this component, one device (pump or motor) is commanded on and proven to be on otherwise a failure is noted.

When the command signal applied to the input (In) becomes true, the output (Out) is driven true and the OutNot is driven false. ProofDelay is a configurable float that sets the time limit before declaring the commanded pump has failed to come on-line. The proving signal can come from an auxiliary switch on a motor contactor, current switch or a flow switch and is applied to the Proof input. If the pump is determined to have failed to come on-line, the slot Fail becomes true as long as FailInhibit is false and will stay on until the command signal returns to the false state. If FailInhibit is true, the alarm slot Fail is disabled.

Period Driven Clock (TockTic)

TockTic	
CControls_HVAC::TockTic	
Period	1.0
Enable	true
Out	true

TockTic should not be confused with the TickToc component in Tridium's Func kit. With the Tridium component, the clock frequency is set in steps per second between 1 and 10. With TockTic, the frequency is set by the period allowing for much slower frequencies thereby providing a much slower counting source.

Enable defaults to true and must be true to enable the output (Out). Period is a configurable float slot with a minimum value of 0.5 seconds. When enabled, Out will provide a square wave with a frequency of one divided by the specified period.

5.3   Math Kit (CControls_Math)

What follows are component descriptions for Sedona components in the CControls_Math kit. This is a custom hardware independent kit that can be used with any Sedona 1.2.28 platform.

Add	+
CControls_Math::Add	
Inp1	0.0
Inp2	0.0
Out	0.0

Two-Input Addition with Configurable Inputs (Add)

Solves the equation Out = Inp1 + Inp2

The two input slots are configurable.

Div	+
CControls_Math::Div	
Inp1	0.0
Inp2	0.0
Out	0.0
Div0	true

Two-Input Divide with Configurable Inputs (Div)

Solves the equation Out = Inp1 / Inp2

The two input slots are configurable. If Inp2 is zero, the Divide by zero slot (Div0) is true.

Mul	x
CControls_Math::Mul	
Inp1	0.0
Inp2	0.0
Out	0.0

Two Input Multiply with Configurable Inputs (Mul)

Solves the equation Out = Inp1 ∗ Inp2

The two input slots are configurable.

Sub	−
CControls_Math::Sub	
Inp1	0.0
Inp2	0.0
Out	0.0

Two-Input Subtract with Configurable Inputs (Sub)

Solves the equation Out = Inp1 - Inp2

The two input slots are configurable.

5.4   Function 2 Kit (CControls_Function2)

The CControls_Function2 kit is hardware independent and therefore can operate with any Sedona 1.2 virtual machine. The kit mostly contains variations of components found in the Tridium release kits but with inputs modified to be configurable. The advantage of configurable inputs is that they save memory by eliminating the need to add constant components and links to set a non-configurable input to a constant value. Configurable inputs do not need external links to retain value. The component's inputs can be set with a Sedona tool. If there is an external link from another component, the linked output of the driving component determines the input value of the receiving component and not its configuration. If power is cycled to the controller, configurable inputs retain their value which is not the case with non-configurable inputs. However, the controller program must first be saved on the controller before configuration retention is guaranteed before power cycling.

ASWC1	⊃
CControls_Function2::ASWC	
Out	0.0
In1	0.0
In2	0.0
S1	false

Analog switch — selection between two float variables (ASWC)

If S1 is false, then Out = In1

If S1 is true, then Out = In2

The two input slots are configurable.

ASW4C1	
CControls_Function2::ASW4C	
Out	0.0
In1	0.0
In2	0.0
In3	0.0
In4	0.0
StartsAt	0
Sel	0

Analog switch — selection between four floats (ASW4C)

Configurable integer parameter *StartsAt* sets the base selection.

If integer Sel <= StartsAt, then Out = In1

If integer Sel = StartsAt + 1, then Out = In2

If integer Sel = StartsAt + 2, then Out = In3

If integer Sel = StartsAt + 3, then Out = In4

For all values of Sel that are 4 greater than StartsAt then Out = In4

The four input slots and *StartsAt* are configurable.

BSWC1	⊃
CControls_Function2::BSWC	
Out	false
In1	false
In2	false
S1	false

Boolean Switch — selection between two Boolean variables (BSWC)

If S1 is false, then Out = In1

If S1 is true, then Out = In2

The two input slots are configurable.

ISWC1	⊃
CControls_Function2::ISWC	
Out	0
In1	0
In2	0
S1	false

Integer switch — selection between two integer variables (ISWC)

If S1 is false, then Out = In1

If S1 is true, then Out = In2

The two input slots are configurable.

Comparison math — comparison (<=>) of two floats (CmprC)

CmprC1	<
CControls_Function2::CmprC	
Xgy	false
Xey	true
Xly	false
X	0.0
Y	0.0

If X > Y, then Xgy is true

If X = Y, then Xey is true

If X < Y, then Xly is true

The two input slots X and Y are configurable.

Max Enumeration – maximum integer valve from weighted inputs

MAXE1	
CControls_Function2::MAXE	
OutI	0
OutF	0.0
In1	false
In2	false
In3	false
In4	false
In5	false
StartsAt	0
Cascade	0

If In1 is true, then A = 1

If In2 is true, then B = 2

If In3 is true, then C = 3

If In4 is true, then D = 4

If In5 is true, then E = 5

Q = Max[A, B, C, D, E]

If Q = 0, then Out = cascade; otherwise, Out = Q + StartsAt

Both float and integer Output values provided. *StartsAt* is configurable.

ASWC Test Program

After configuration as shown, first save to controller and then cycle power. Configured inputs should survive a power cycle.

TickTo1	
func::TickTock	
Out	true
TicksPerSec	1

ASWC1	
CControls_Function2::ASWC	
Out	20.0
In1	10.0
In2	20.0
S1	true

ASWC4 Test Program

After configuration as shown, first save to controller and then cycle power. Configured inputs and *StartsAt* should survive a power cycle
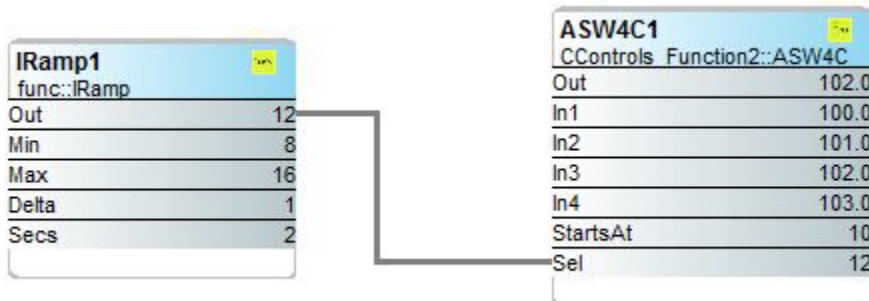
IRamp1	
func::IRamp	
Out	12
Min	8
Max	16
Delta	1
Secs	2

ASW4C1	
CControls_Function2::ASW4C	
Out	102.0
In1	100.0
In2	101.0
In3	102.0
In4	103.0
StartsAt	10
Sel	12

BSWC Test Program

After configuration as shown, first save to controller and then cycle power. Configured inputs should survive a power cycle.

TickTo2	
func::TickTock	
Out	true
TicksPerSec	1

BSWC1	
CControls_Function2::BSWC	
Out	true
In1	true
In2	false
S1	false

ISWC Test Program

After configuration as shown, first save to controller and then cycle power. Configured inputs should survive a power cycle.

TickTo3	
func::TickTock	
Out	false
TicksPerSec	1

ISWC1	
CControls_Function2::ISWC	
Out	20
In1	10
In2	20
S1	true

CmprC Test Program

After configuration as shown, first save to controller and then cycle power. Configured inputs should survive a power cycle.

Ramp1	
func::Ramp	
Out	72.76
Min	0.0
Max	100.0
Period	10
RampType	triangle

CmprC1	
CControls_Function2::CmprC	
Xgy	false
Xey	false
Xly	true
X	50.0
Y	76.78

.
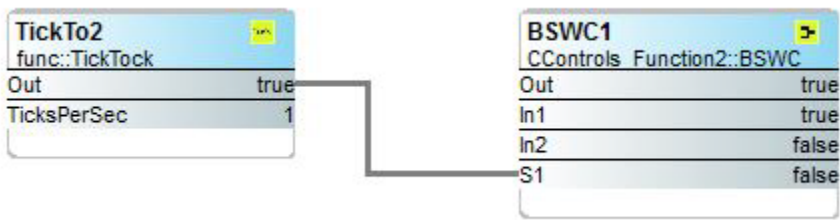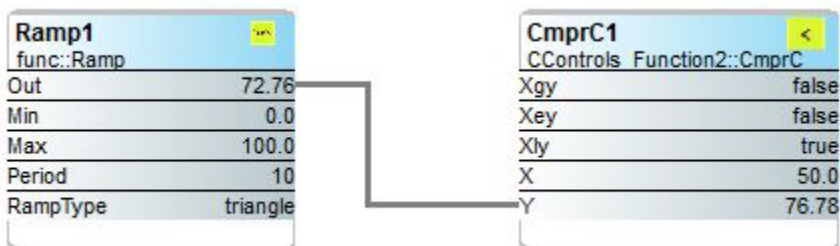
MAXE Test Program

After configuration as shown, first save to controller and then cycle power. *StartsAt* on MAXE2 and MAXE3 should survive a power cycle.

The following program demonstrates how to cascade three MAXE components to triple the number of possible enumerated values. MAXE1 is at the beginning of the cascade so its *Cascade* input is left open and its *StartsAt* is left at 0. MAXE2 is in the middle of the cascade so its *Cascade* input is tied to MAXE1's integer output. Is *StartsAt* value is set to 5. MAXE3 is at the end of the cascade so its *Cascade* input is tied to MAXE2's integer output and its *StartsAt* value is set to 10. The result of 15 weighted inputs can be read at the output of MAXE3 as an enumerated value.

Enter power of 2 values minus 1 into ConstFl. For example, to test enumerated value 14 enter 2^14 minus 1 which is equal to 16383. Since *In4* on MAXE3 is true and *In5* of MAXE3 is false, the output of MAXE would be 14. All other inputs are ignored because they have lower weighting.
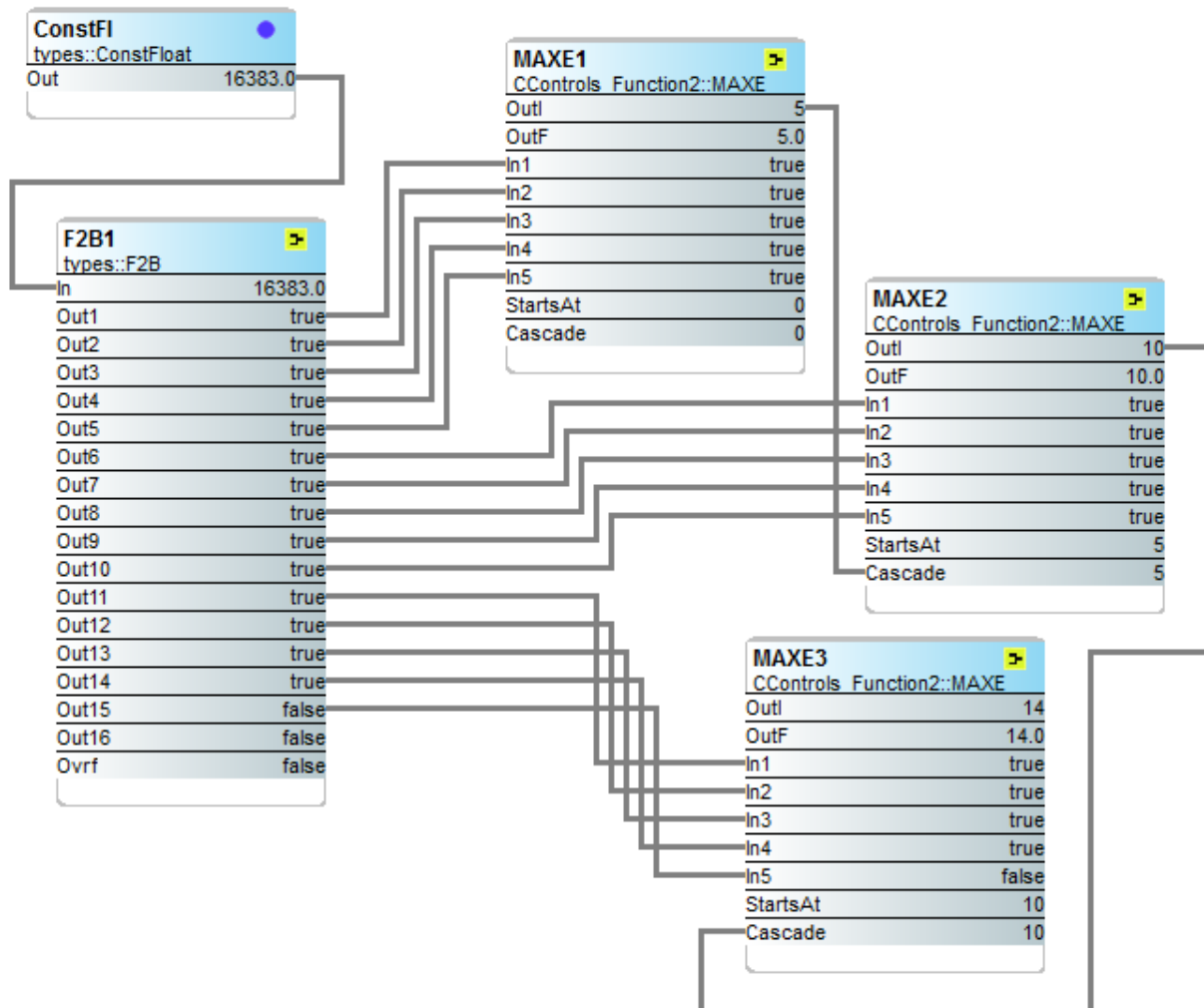
5.5   Math2 Kit (CControls_Math2)

Absolute Value – Absolute value of a float input

Out = |In|

If In = 0 then Out = 0

If In is positive then Out is positive; If In is negative then Out is positive

Degrees to Radian Conversion – Decimal degrees converted to radians

Out = In / (180/π)

Radian to Degree Conversion – Radians converted to decimal degrees

Out = In ∗ (180/π)

Pi Multiplier – the constant Pi multiplied by a constant float

Out = In ∗ (π)

Radian to Sine Conversion – the Sine value of the input in radians

Out = Sin (In)

Square Root – the square root of a float

Out = √In

Minimum Delta – Minimizing output changes from input jitter

All slots are floats. The configurable slot called Delta provides a deadband around the input such that the last output will hold value if the input remains within the deadband. To guard against input creep, once the sample time in seconds is passed, the output will then equal the input and the sampling continues. If the input exceeds the deadband within the sampling time, the output will then equal the input and the sampling time will reset.

Gen16T	
CControls_Math2::Gen16T	
In	3.5
Out	12.5
Range	Contained
Default_Output	null
In1	null
Value1	null
In2	-6.0
Value2	36.0
In3	-5.0
Value3	25.0
In4	-4.0
Value4	16.0
In5	-3.0
Value5	9.0
In6	-2.0
Value6	4.0
In7	-1.0
Value7	1.0
In8	0.0
Value8	0.0
In9	1.0
Value9	1.0
In10	2.0
Value10	4.0
In11	3.0
Value11	9.0
In12	4.0
Value12	16.0
In13	5.0
Value13	25.0
In14	6.0
Value14	36.0
In15	null
Value15	null
In16	null
Value16	null

General Table – Piecewise linearization of a function

The Gen8T, Gen16T and Gen32T components provide 8, 16 and 32 table entries respectively allowing a linear or non-linear input function, which is a float, to be modified based upon table entries. If the input value is the same as a table entry, the output will equal the value of that table entry. If the input is between two table entries, then the output will equal the linearized value between the two table entries. If the input exceeds the boundary limits of the table, the output will depend upon the *Range* mode.

Constrained – *Out is equal to the boundary value*
EdgeSlope – *Out assumes the slope value of the two closest table entries*
PassThrough – *Out is equal to the input*
Default_OutValue – *Out is equal to the value of the Default_Output slot*

Any null entry is ignored. The piecewise linear equation is as follows:

$$Out = y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0}$$

where y is the value for input value x between coordinates x_0, y_0 and x_1, y_1

In the example to the left, a table is created with points on a curve that satisfies

With an input of 3.5, the output must create a value between points In11 (3.0) and In12 (4.0). The values for these two points are respectively:

 Y(3.0) = 9; Y(4.0) = 16

Substituting in the above equation yields Y(3.5) = 12.5 which approximates the true value of the function Y(3.5) = X^2 which is 12.25

MinMaxA
CControls_Math2::MinMaxAvg

Minimum	-2.0
Maximum	9.6
AverageValue	3.27
ValuesCounted	4
In1	1.0
In2	null
In3	-2.0
In4	null
In5	null
In6	4.5
In7	null
In8	9.6
In9	null
In10	null

Minimum Maximum Average – Determining the range of values from multiple points

The MinMaxA component determines the maximum value, the minimum value, and the average value of attached points on every scan. If only ten points are involved, just one MinMaxA component is needed. Above ten points, MinMaxA components can be cascaded to determine the resulting maximum and minimum values. To determine the average value, the Weighted Average (WgtAvg) component is needed, but it too can be cascaded to determine the ultimate average value.

Only connected input points are evaluated. Null values are ignored. All slots are floats.

WgtAvg
CControls_Math2::WgtAvg

AverageValue	0.0
ValuesCounted	0
In1	null
In1Count	0
In2	null
In2Count	0
In3	null
In3Count	0
In4	null
In4Count	0

Weighted Average – Determines average of multiple MinMaxA components

While single MinMaxA component is all that is needed to determine the average value of the inputs, this is not the case with multiple MinMaxA modules. What is needed is the weighted average value by considering the number of points being measured. Simply connect the *AverageValue* and *ValueCounted* outputs from each of the MinMaxA components to one of the *In* and *InCount* input pairs on the WtgAvg component. The weighted average value will appear at the *AverageValue* output along with the carry-forward value counts at the ValuesCounted output. This allows for cascading of WgtAvg components. If average is desired and not weighted average, simply enter a 1 into the configurable *InCount* input of that input pair.

Only connected input points are evaluated. Null values are ignored. All slots are floats except for count inputs and output.

Cascading MinMaxA Components

If more than ten points need to be measured, it is necessary to cascade MinMaxA components. If you are only interested in minimum and maximum values, simply connect the Minimum and Maximum outputs to two inputs on another MinMaxA and use the outputs of the last MinMaxA for the two results. However, if a true averaged value is to be determined, follow the connections below. The last MinMaxA component must only connect to other MinMax components and not any points. For weighted average, connect up to four MinMaxA components to one WgtAvg component. After that, cascading can continue with additional WgtAvg components.

MinMaxA	
CControls_Math2::MinMaxAvg	
Minimum	-11.2
Maximum	9.6
AverageValue	0.97
ValuesCounted	4
In1	1.0
In2	null
In3	-11.2
In4	null
In5	null
In6	4.5
In7	null
In8	9.6
In9	null
In10	null

WgtAvg	
CControls_Math2::WgtAvg	
AverageValue	2.18
ValuesCounted	8
In1	0.97
In1Count	4
In2	null
In2Count	0
In3	null
In3Count	0
In4	3.4
In4Count	4

MinMax1	
CControls_Math2::MinMaxAvg	
Minimum	-7.5
Maximum	10.3
AverageValue	3.4
ValuesCounted	4
In1	null
In2	8.5
In3	null
In4	10.3
In5	null
In6	-7.5
In7	null
In8	null
In9	2.3
In10	null

MinMax2	
CControls_Math2::MinMaxAvg	
Minimum	-11.2
Maximum	10.3
AverageValue	0.3
ValuesCounted	4
In1	-11.2
In2	9.6
In3	-7.5
In4	10.3
In5	null
In6	null
In7	null
In8	null
In9	null
In10	null

5.6   HVAC2 Kit (CControls_P_HVAC2)

Overview of HVAC Macro Components

Macros are large, highly integrated Sedona components that perform multiple logical operations and have a wide range of configurable options in order to obtain a comprehensive control sequence. In contrast, standard Sedona components generally have just one logical operation with limited configuration options, that meets a singular purpose. The HVAC macros, developed by Contemporary Controls using Sedona developer tools, form the nucleus of heating/cooling applications. They include the following:

The **Staged Heat Cool** macro for multi-stage heating/cooling along with fan stop/start control, emergency shutdown, and optional buffered analog outputs.

The **Wall Setter** macro provides a convenient interface to either digital (network communicating) wall sensors such as BASstat Wi-Fi, or analog wall setters such as the Senva AQW series with provisions for space temperature thermistor, slider potentiometers, temporary occupancy, and CO2 transmitters.

The **Economizer** macro, in either SI or I-P units, implements various airside economizer schemes which include dry-bulb, enthalpy, demand control ventilation (DCV), and optional powered exhaust.

Note that several free downloadable HVAC applications utilize one or more of these macro components.

The use of macro components provides an opportunity to reduce both application memory space and wiresheet component density. Here is a "Before and After" example of the **Staged Heat Cool** macro:

Staged Heat Cool (StgHtCl)
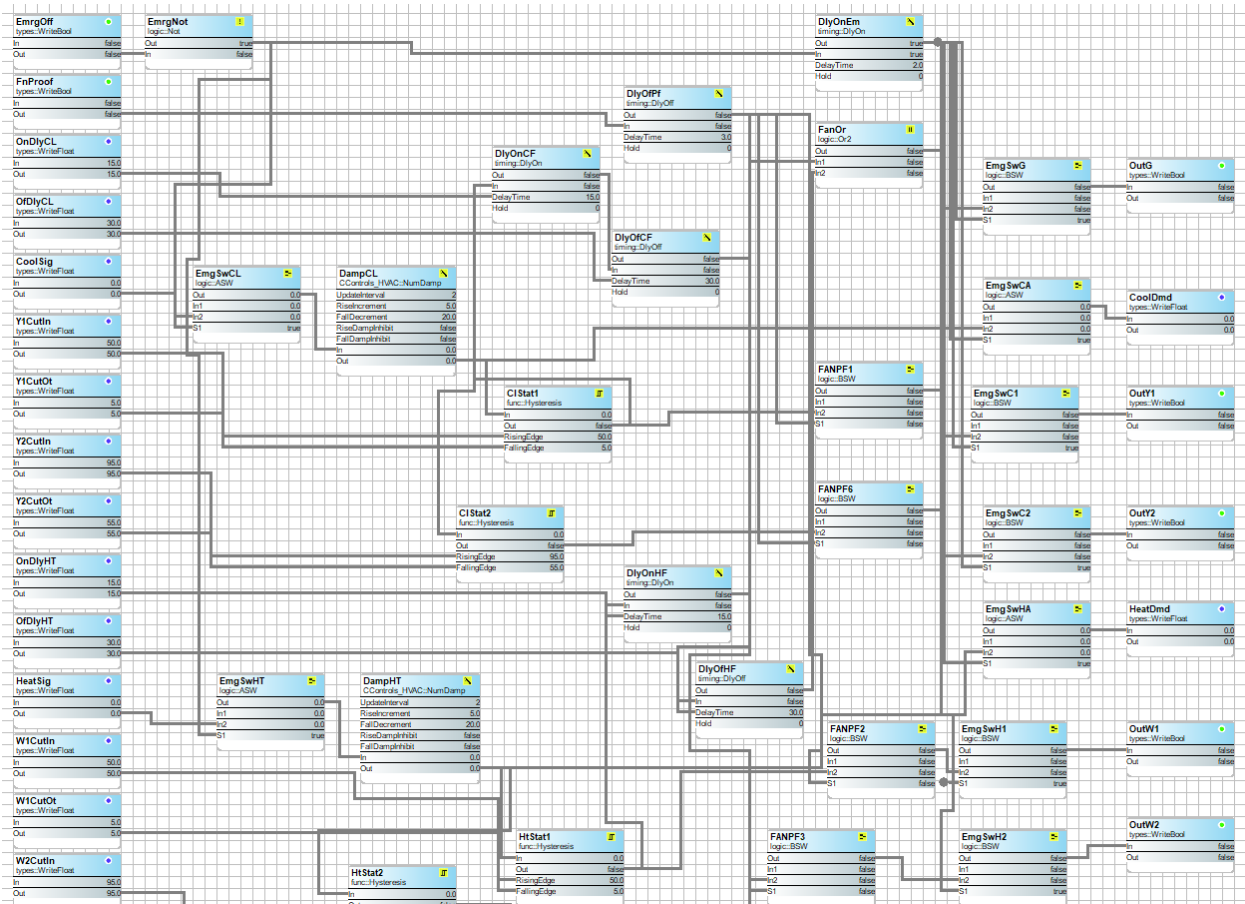
StgHtCl	
CControls_Macro::StgHtCl	
CoolAnalogInput	0.0
HeatAnalogInput	0.0
EmergencyOff	Normal
FanFlowProof	Standby
OutY1	false
OutY2	false
OutW1	false
OutW2	false
OutAutoG	false
OutCoolDemand	0.0
OutHeatDemand	0.0
Y1CutIn	50.0
Y1CutOut	5.0
Y2CutIn	95.0
Y2CutOut	55.0
W1CutIn	50.0
W1CutOut	5.0
W2CutIn	95.0
W2CutOut	55.0
G_OnDelayCool	15.0
G_OffDelayCool	30.0
G_OnDelayHeat	5.0
G_OffDelayHeat	45.0

Summary: Combines one or two-stage heating/cooling binary outputs, supply fan stop/start control, fan proof protection, buffered analog outputs, and emergency (instantaneous) shutdown.

Outputs from separate 0 to 100% Heat and Cool PID or Reset type components are input into the multistage component. Analog outputs are dampened.

There is provision for 2 stages of heat and 2 stages of cooling, but not all stages must be used. Each stage has definable Cut In and Cut Out values.

Auto Fan G is enabled (after the associated-Auto fan On Delay) whenever Y1 or W1 differential is Cut In.

If Fan Proof is off for more than 3 seconds, all Y and W outputs are disabled. Heat & Cool demand outputs are set to zero. Auto Fan G remains on.

If EmergShutdown is ON (true) all Boolean outputs are instantly turned off and analog heating and cooling demand is set to zero.

Slot Name	Slot Type	Notes and Default Values
CoolAnalogInput	Input (Float)	0-100% is the expected range
HeatAnalogInput	Input (Float)	0-100% is the expected range
EmergencyOff	Input (Binary)	false=Normal, true=Shutdown
FanFlowProof	Input (Binary)	false=Standby, true=Running
OutY1	Output (Binary)	Cooling stage 1
OutY2	Output (Binary)	Cooling stage 2
OutW1	Output (Binary)	Heating stage 1
OutW2	Output (Binary)	Heating stage 2
OutAutoG	Output (Binary)	Fan on during stage 1 of either heat or cool
OutCoolDemand	Output (Float)	0-100% is the expected range - dampened
OutHeatDemand	Output (Float)	0-100% is the expected range - dampened
Y1CutIn	Config Input (Float)	50%
Y1CutOut	Config Input (Float)	5%
Y2CutIn	Config Input (Float)	95%
Y2CutOut	Config Input (Float)	55%
W1CutIn	Config Input (Float)	50%
W1CutOut	Config Input (Float)	5%
W2CutIn	Config Input (Float)	95%
W2CutOut	Config Input (Float)	55%
G_OnDelayCool	Config Input (Float)	15 seconds

G_OffDelayCool	Config Input (Float)	30 seconds
G_OnDelayHeat	Config Input (Float)	5 seconds
G_OffDelayHeat	Config Input (Float)	45 seconds

Staged Heat Cool Macro – Suggested Applications

1. One or two stage cool, one or two stage heat, auto fan control
2. One or two stage cool, analog heat, auto fan control
3. Three or four stage heating **or** cooling output, auto fan control – note that this is accomplished by linking to the Cool and Heat analog inputs from one source, and then setting the 8 Cut In and Cut Out values to provide the desired sequence. If heating, use Y1 as W3 and Y2 as W4. If cooling, use W1 as Y3 and W2 as Y4.

Wall Setter (WallSet)

```
WallSet
CControls_Macro::WallSet
OccupyStatus              Unoccupied
SetpointOhms                     0.0
ZoneTempTherm                    0.0
ZoneTempVolts                    0.0
HumidityVolts                    0.0
CO2_Volts                        0.0
OutEffCoolingSP                 85.0
OutEffHeatingSP                 55.0
OutTemperature                   0.0
OutHumidity                      0.0
OutCO2ppm                        0.0
OutOccOvrd                   Standby
OutUnocCooling                   Off
OutUnocHeating                    On
UnocCoolingSP                   85.0
OccCoolingSP                    75.0
OccHeatingSP                    70.0
UnocHeatingSP                   55.0
CoolSPminLimit                  70.0
HeatSPmaxLimit                  72.0
SP_PotInMin                      0.0
SP_PotInMax                  10000.0
SP_PotOutMin                    65.0
SP_PotOutMax                    75.0
LocalSP_Deadband                 5.0
LocalOvrdDurationMinutes       120.0
CO2_OutMax                    2000.0
ZoneTempV_OutMin                50.0
ZoneTempV_OutMax                95.0
ZnTempTorV_Select         UseThermIn
OccupiedSP_Select            UseSP_Pot
```

Summary: Inputs from either digital (network communicating) or analog type wall setters provides space temperature, setpoint, humidity, and CO2 control. Occupancy input toggles the effective heat-cool setpoints used by downstream logic.

Local setpoint potentiometer occupied setpoints or network/fixed occupied setpoints can be dynamically selected.

If local SP pot is used, range and limits can be set. When in Unoccupied mode, Unoccupied Heat and Cool setpoints are used.

While in Unoccupied mode, high and low limits are in place. If zone temperature rises above UnocCooling SP then OutUnocCooling is enabled, if temp drops below UnocHeating SP then OutUnocHeating is enabled. There is a fixed 2-degree differential on the Unoccupied Heat or Cool outputs.

If local zone temp thermistor is shorted a "fail" value of 40 initiates timed override.

If local zone temp source is voltage rather than thermistor, set ZnTempTorV_Select to true.

If fixed (or network) occupied setpoints are used set OccupiedSP_Select to true.

Minimum cooling and maximum heating setpoint limits can be utilized.

When using local occupied setpoints, the Deadband is fixed. When using network setpoints the Deadband represents the minimum spread between heating and cooling setpoints.

All voltage inputs are assumed to be 0 to 10 vdc. If a different range is required, the Reset component must be used to convert the transducer range to the required 0 to 10 input.

Slot Name	Slot Type	Notes and Default Values
OccupyStatus	Input (Binary)	false=Unoccupied, true=Occupied
SetpointOhms	Input (Float)	0-10k ohms is the expected range
ZoneTempTherm	Input (Float)	10k T2 or T3 thermistor
ZoneTempVolts	Input (Float)	0-10 volts is the expected range
HumidityVolts	Input (Float)	0-10 volts is the expected range
CO2_Volts	Input (Float)	0-10 volts is the expected range
OutEffCoolingSP	Output (Float)	Effective (in use) cooling setpoint
OutEffHeatingSP	Output (Float)	Effective (in use) heating setpoint
OutTemperature	Output (Float)	Outputs actual temperature

OutHumidity	Output (Float)	Outputs actual humidity %
OutCO2ppm	Output (Float)	Outputs actual CO2 ppm
OutOccOvrd	Output (Binary)	true=Temporary Occupy Override (trigger @ ZnTemp of 40)
OutUnocCooling	Output (Binary)	true=Zone temp is above Unoc Cool SP
OutUnocHeating	Output (Binary)	true=Zone temp is below Unoc Heat SP
UnocCoolingSP	Config Input (Float)	85f
OccCoolingSP	Config Input (Float)	75f
OccHeatingSP	Config Input (Float)	70f
UnocHeatingSP	Config Input (Float)	55f
CoolSPminLimit	Config Input (Float)	70f
HeatSPmaxLimit	Config Input (Float)	72f
SP_PotInMin	Config Input (Float)	0 ohms
SP_PotInMax	Config Input (Float)	10k ohms
SP_PotOutMin	Config Input (Float)	65f
SP_PotOutMax	Config Input (Float)	75f
SP_Deadband	Config Input (Float)	5f
LocalOvrdDuration	Config Input (Float)	120 minutes
CO2_OutMax	Config Input (Float)	2000 ppm
ZoneTempV_OutMin	Config Input (Float)	50f
ZoneTempV_OutMax	Config Input (Float)	95f
ZnTempTorV_Select	Config Input (Binary)	false=Use Thermistor In, true=Use Volts In
OccupiedSP_Select	Config Input (Binary)	false=Use local SP pot, true=Use fixed Occ SPs

Wall Setter Macro – Suggested Applications

1. Analog type wall setter (Ex: Senva AQW) for zone temp and occupied setpoint. Optional override button, zone RH and zone CO2. Occupancy call via BAS head-end or local schedule.
2. BACnet communicating type wall setter (Ex: CCSI BASstat Wi-Fi) for zone temp and occupied setpoint. Occupancy call via BAS head-end or local schedule. Note: If the communicating wall setter has only one setpoint value then link to **both** OccCoolingSP and OccHeatingSP input slots. The original value will become the OccCoolingSP. The OccHeatingSP will automatically be the original value **minus** the Deadband.
3. BAS head-end provided zone temp, RH, CO2, occupied setpoints, occupancy schedule.

Economizer-English Units (EconoE)

EconoE	
CControls_P_HVAC2::EconoE	
OccupyStatus	Unoccupied
SfanStatus	Standby
ReturnTemp	0.0
ReturnHumidity	0.0
ReturnCO2ppm	0.0
MixedTemp	0.0
OutsideTemp	0.0
OutsideHumidity	0.0
EffectiveCoolingSP	0.0
RetDamperPosVolts	0.0
OutDamperPosVolts	0.0
PurgeCommand	Standby
OutEconoDamperV	2.0
OutRetDmprEffPos	0.0
OutOA_DmprEffPos	0.0
OutOA_TrueBlend	0.0
OutRA_Enthalpy	0.0
OutOA_Enthalpy	0.0
OutOA_Dewpoint	0.0
EconCoolingDemand	0.0
DCV_CO2_Demand	0.0
MechCoolingEnab	Enable
ExhFanEnab	Standby
DryWetModeSelect	Dry bulb
FixedDiffModeSelect	Differential (Dual)
DCV_ModeSelect	Use MinVent only
MinVentPosSP	10.0
MaxDCV_PosSP	50.0
DCV_CO2_Setpoint	1000.0
DryBulbLimitSP	72.0
EnthalpyLimitSP	26.0
MA_LowLimitSP	40.0
ActuatorMinVolts	2.0
MechCoolDelayMinutes	5.0
ExFanCutInSP	80.0
ExFanCutOutSP	40.0
EconPID_Kp	6.0
EconPID_Ki	1.5
CO2_PID_Kp	2.0
CO2_PID_Ki	0.5

Summary: A comprehensive "virtual" airside economizer. Configuration options include single or dual dry-bulb, single or dual enthalpy, and demand control ventilation (DCV), powered exhaust fan, and a Purge mode. Economizer intercepts the call for mechanical cooling if economization can be implemented.

When econ cooling is enabled modulate econ damper to maintain Eff Cool SP minus 1F.

If Mixed Air temp drops below Low Limit of 40F (adj), restrict damper output to 10%.

If Econ damper is 100% open for more than 5 minutes (adj), enable Mech Cooling.

If economizer cooling is disabled, then enable (allow) Mech cooling.

If DCV mode is enabled modulate damper to maintain DCV CO2 setpoint but limit econ damper to Max DCV. In DCV mode set MinVentPos to 0*. When not using DCV mode, set MinVentPos to 10%* (*or according to Balancer or True Blend).

If powered exhaust fan is used adjust cut in and cut out as per Balancer. Default is On at 80%, Off at 40%. 30 sec fixed start delay.

In Purge mode drive econ to 100% and enable Exh fan. Note that MAT Low Limit protection is still in effect during Purge.

Configure for dry or wet, and fixed or differential modes using Select inputs as appropriate. Consult Climate Zone Chart for area settings.

An 80F Outside Air economizer high limit lockout applies to all dry or wet modes.

OutOA-TrueBlend is a calculation of the actual percentage of outside air that is being allowed in.

Slot Name	Slot Type	Notes and Default Values
OccupyStatus	Input (Binary)	false=Unoccupied, true=Occupied
SfanStatus	Input (Binary)	false=Standby, true=Running
ReturnTemp	Input (Float)	10k T2 or T3 thermistor
ReturnHumidity	Input (Float)	0-10 volts is the expected range
ReturnCO2ppm	Input (Float)	0-10 volts is the expected range
MixedTemp	Input (Float)	10k T2 or T3 averaging thermistor

OutsideTemp	Input (Float)	10k T2 or T3 thermistor
OutsideHumidity	Input (Float)	0-10 volts is the expected range
EffectiveCoolingSP	Input (Float)	Effective (in use) cooling setpoint
RetDamperPosVolts	Input (Float)	0-10 (or 2 to 10) volts is the expected range
OutDamperPosVolts	Input (Float)	0-10 (or 2 to 10) volts is the expected range
PurgeCommand	Input (Binary)	true=Econ damper at 100%, Exh fan enabled
OutEconoDamperV	Output (Float)	Output voltage signal to drive RA/OA damper actuators
OutRetDmprEffPos	Output (Float)	Position feedback % from RA damper
OutOA_DmprEffPos	Output (Float)	Position feedback % from OA damper
OutOA_TrueBlend	Output (Float)	Outputs the actual % of outside air being injected
OutRA_Enthalpy	Output (Float)	Outputs actual return air enthalpy
OutOA_Enthalpy	Output (Float)	Outputs actual outside air enthalpy
OutOA_Dewpoint	Output (Float)	Outputs actual outside air dew point
EconCoolingDemand	Output (Float)	Outputs Econ PID out %
DCV_CO2_Demand	Output (Float)	Outputs CO2 PID out %
MechCoolingEnab	Output (Binary)	Enables/disables mechanical cooling
ExhFanEnab	Output (Binary)	Enables/disables powered exhaust fan
DryWetModeSelect	Config Input (Binary)	false(def)=Dry bulb, true=Wet bulb (Enthalpy)
FixedDiffModeSelect	Config Input (Binary)	false=Fixed limit (Single), true(def)=Differential (Dual)
DCV_ModeSelect	Config Input (Binary)	false(def)=Use MinVent only (No DCV), true=Use DCV mode
MinVentPosSP	Config Input (Float)	10% (Should be set by Air Balancer or True Blend%)
MaxDCV_PosSP	Config Input (Float)	50% (Should be set by Air Balancer)
DCV_CO2_Setpoint	Config Input (Float)	1000 ppm (800 ppm to 1600 ppm range)
DryBulbLimitSP	Config Input (Float)	72F (Adjust based on climate zone: 65 to 75 range)
EnthalpyLimitSP	Config Input (Float)	26btu/lb (Adjust based on climate zone: 21 to 29 range)
MA_LowLimitSP	Config Input (Float)	40F (35F to 55F range)
ActuatorMinVolts	Config Input (Float)	2v (usually either 2 or 0 is expected)
MechCoolDelayMinutes	Config Input (Float)	5 minutes (after Econ PID reaches 99% output)
ExFanCutInSP	Config Input (Float)	On at 80% OA damper position
ExFanCutOutSP	Config Input (Float)	Off at 40% OA damper position
EconPID_Kp	Config Input (Float)	6 (P gain)
EconPID_Ki	Config Input (Float)	1.5 (I gain)
CO2_PID_Kp	Config Input (Float)	2 (P gain)
CO2_PID_Ki	Config Input (Float)	.5 (I gain)

Economizer-SI Units (EconoS)

| EconoS | |
CControls_P_HVAC2::EconoS	
OccupyStatus	Unoccupied
SfanStatus	Standby
ReturnTemp	0.0
ReturnHumidity	0.0
ReturnCO2ppm	0.0
MixedTemp	0.0
OutsideTemp	0.0
OutsideHumidity	0.0
EffectiveCoolingSP	0.0
RetDamperPosVolts	0.0
OutDamperPosVolts	0.0
PurgeCommand	Standby
OutEconoDamperV	2.0
OutRetDmprEffPos	0.0
OutOA_DmprEffPos	0.0
OutOA_TrueBlend	0.0
OutRA_Enthalpy	0.0
OutOA_Enthalpy	0.0
OutOA_Dewpoint	0.0
EconCoolingDemand	0.0
DCV_CO2_Demand	0.0
MechCoolingEnab	Enable
ExhFanEnab	Standby
DryWetModeSelect	Dry bulb
FixedDiffModeSelect	Differential (Dual)
DCV_ModeSelect	Use MinVent only
MinVentPosSP	10.0
MaxDCV_PosSP	50.0
DCV_CO2_Setpoint	1000.0
DryBulbLimitSP	22.0
EnthalpyLimitSP	42.0
MA_LowLimitSP	5.0
ActuatorMinVolts	2.0
MechCoolDelayMinutes	5.0
ExFanCutInSP	80.0
ExFanCutOutSP	40.0
EconPID_Kp	6.0
EconPID_Ki	1.5
CO2_PID_Kp	2.0
CO2_PID_Ki	0.5

Summary: A comprehensive "virtual" airside economizer. Configuration options include single or dual dry-bulb, single or dual enthalpy, and demand control ventilation (DCV), powered exhaust fan, and a Purge mode. Economizer intercepts the call for mechanical cooling if economization can be implemented.

When econ cooling is enabled modulate econ damper to maintain Eff Cool SP minus 1c.

If Mixed Air temp drops below Low Limit of 5c (adj), restrict damper output to 10%.

If Econ damper is 100% open for more than 5 minutes (adj), enable Mech Cooling. If economizer cooling is disabled, then enable (allow) Mech cooling.

If DCV mode is enabled modulate damper to maintain DCV CO2 setpoint but limit econ damper to Max DCV. In DCV mode set MinVentPos to 0*. When not using DCV mode, set MinVentPos to 10%* (*or according to Balancer or True Blend)

If powered exhaust fan is used adjust cut in and cut out as per Balancer. Default is On at 80%, Off at 40%. 30 sec fixed start delay.

In Purge mode drive econ to 100% and enable Exh fan. Note that MAT Low Limit protection is still in effect during Purge.

Configure for dry or wet, and fixed or differential modes using Select inputs as appropriate. Consult Climate Zone Chart for area settings.

A 27c Outside Air economizer high limit lockout applies to all dry or wet modes.

OutOA-TrueBlend is a calculation of the actual percentage of outside air that is being allowed in.

Slot Name	Slot Type	Notes and Default Values
OccupyStatus	Input (Binary)	false=Unoccupied, true=Occupied
SfanStatus	Input (Binary)	false=Standby, true=Running
ReturnTemp	Input (Float)	10k T2 or T3 thermistor
ReturnHumidity	Input (Float)	0-10 volts is the expected range

ReturnCO2ppm	Input (Float)	0-10 volts is the expected range
MixedTemp	Input (Float)	10k T2 or T3 averaging thermistor
OutsideTemp	Input (Float)	10k T2 or T3 thermistor
OutsideHumidity	Input (Float)	0-10 volts is the expected range
EffectiveCoolingSP	Input (Float)	Effective (in use) cooling setpoint
RetDamperPosVolts	Input (Float)	0-10 (or 2 to 10) volts is the expected range
OutDamperPosVolts	Input (Float)	0-10 (or 2 to 10) volts is the expected range
PurgeCommand	Input (Binary)	true=Econ damper at 100%, Exh fan enabled
OutEconoDamperV	Output (Float)	Output voltage signal to drive RA/OA damper actuators
OutRetDmprEffPos	Output (Float)	Position feedback % from RA damper
OutOA_DmprEffPos	Output (Float)	Position feedback % from OA damper
OutOA_TrueBlend	Output (Float)	Outputs the actual % of outside air being injected
OutRA_Enthalpy	Output (Float)	Outputs actual return air enthalpy
OutOA_Enthalpy	Output (Float)	Outputs actual outside air enthalpy
OutOA_Dewpoint	Output (Float)	Outputs actual outside air dew point
EconCoolingDemand	Output (Float)	Outputs Econ PID out %
DCV_CO2_Demand	Output (Float)	Outputs CO2 PID out %
MechCoolingEnab	Output (Binary)	Enables/disables mechanical cooling
ExhFanEnab	Output (Binary)	Enables/disables powered exhaust fan
DryWetModeSelect	Config Input (Binary)	false(def)=Dry bulb, true=Wet bulb (Enthalpy)
FixedDiffModeSelect	Config Input (Binary)	false=Fixed limit (Single), true(def)=Differential (Dual)
DCV_ModeSelect	Config Input (Binary)	false(def)=Use MinVent only (No DCV), true=Use DCV mode
MinVentPosSP	Config Input (Float)	10% (Should be set by Air Balancer or True Blend%)
MaxDCV_PosSP	Config Input (Float)	50% (Should be set by Air Balancer)
DCV_CO2_Setpoint	Config Input (Float)	1000 ppm (800 ppm to 1600 ppm range)
DryBulbLimitSP	Config Input (Float)	22c (Adjust based on climate zone: 18 to 24 range)
EnthalpyLimitSP	Config Input (Float)	42kJ/kg (Adjust based on climate zone: 33 to 53 range)
MA_LowLimitSP	Config Input (Float)	5c (2c to 13c range)
ActuatorMinVolts	Config Input (Float)	2v (usually either 2 or 0 is expected)
MechCoolDelayMinutes	Config Input (Float)	5 minutes (after Econ PID reaches 99% output)
ExFanCutInSP	Config Input (Float)	On at 80% OA damper position
ExFanCutOutSP	Config Input (Float)	Off at 40% OA damper position
EconPID_Kp	Config Input (Float)	6 (P gain)
EconPID_Ki	Config Input (Float)	1.5 (I gain)
CO2_PID_Kp	Config Input (Float)	2 (P gain)
CO2_PID_Ki	Config Input (Float)	.5 (I gain)

Enhanced Thermostat – On/Off temperature control or limit detection

The Enhanced Thermostat provides basic temperature control or limit detection when the measured variable (*Cv*) exceeds or equals the *Setpoint* by an amount determined by the *Differential*.

For conventional on/off cooling applications the *Action* slot should be set to *Direct* and the *DiffType* slot to *Split*. When a rising measured variable equals or exceeds the *Setpoint* by half the *Differential*, *Out* will be *True*. For the output to return to a *False* state, the measured variable must be equal to or be below the *Setpoint* by half of the *Differential*.

For conventional on/off heating applications the *Action* slot should be set to *Reverse* and the *DiffType* slot to *Split*. When a lowering measured variable is equal to or is below the *Setpoint* by half the *Differential, Out* will be *True*. For the output to return to a *False* state, the measured variable must be equal to or be above the *Setpoint* by half of the *Differential*.

For low temperature (freeze) limit detection, set the *Action* slot to *Reverse* and the *DiffType* slot to *Full*. When the falling measured variable is equal to or is below the *Setpoint, Out* will be *True*. For the output to return to *False*, the measured variable must be equal to or exceed the *Setpoint* by the amount of the *Differential.*

For high temperature limit detection, set the *Action* slot to *Direct* and the *DiffType* slot to *Full*. When a rising measured variable equals to or exceeds the *Setpoint, Out* will be *True*. For the output to return to a *False* state, the measured variable must be equal to or be below the *Setpoint* by the amount of the *Differential*.

Below is a simple test program that allows the study of the Enhanced Thermostat component.
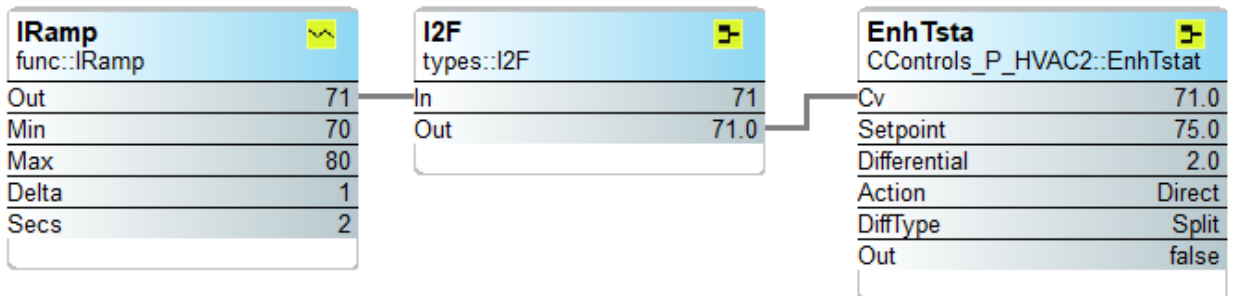
United States
Contemporary Control Systems, Inc.
2431 Curtiss Street
Downers Grove, Illinois 60515
USA

Tel: +1-630-963-7070
Fax: +1-630-963-0109

info@ccontrols.com

China
Contemporary Controls (Suzhou) Co. Ltd
19F, Metropolitan Towers
No.199 Shishan Road
Suzhou New District,
Suzhou, PR
China 215009

Tel: +86 512 68095866
Fax: +86 512 68093760

info@ccontrols.com.cn

United Kingdom
Contemporary Controls Ltd
14 Bow Court
Fletchworth Gate
Coventry CV5 6SP
United Kingdom

Tel: +44 (0)24 7641 3786
Fax:+44 (0)24 7641 3923

info@ccontrols.co.uk

Germany
Contemporary Controls GmbH
Fuggerstraße 1 B
04158 Leipzig
Germany

Tel: +49 341 520359 0
Fax: +49 341 520359 16

info@ccontrols.de

www.ccontrols.com

CONTEMPORARY CONTROLS®