Tridium's Sedona 1.2 Component Descriptions

Components are deployed in kits which are available from Tridium, 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 hardware 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.

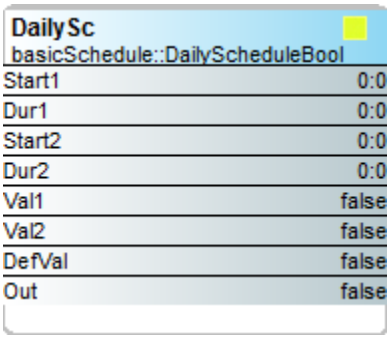
It is recommended that these kits or components not be modified from their original form to ensure the portability of components and kits among members of the Sedona community. If changes or enhancements need to be made by a Sedona developer, the developer can use what currently exists, modify or enhance 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. Many of the following components may have been expanded from their Niagara Workbench (Workbench) view in order to show all component slots and configuration detail. The Sedona Application Editor (SAE) tool only shows expanded views thereby always showing all the slots. The SAE view is the one used for the following component images. The default view of these components when using Workbench may not show the same level of detail.

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.

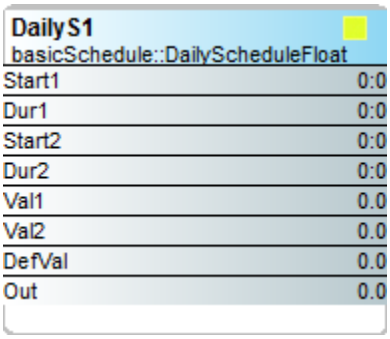
Duration periods — *Dur1 and Dur2* — are configured in minutes from zero to 1439 minutes.



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

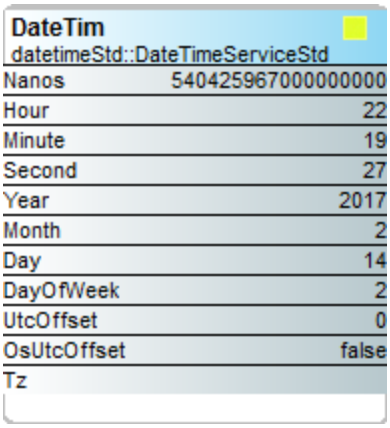


Daily Schedule Float — two-period float scheduler

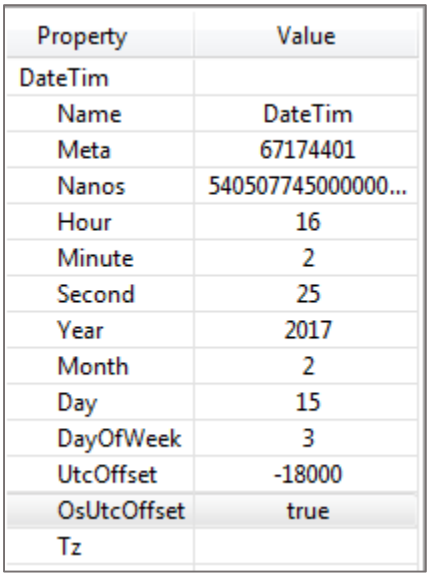
Configure Def Val to the intended output value if there are no active periods. Configure Val1 and Val2 for the desired output values during period 1 and period 2 respectively.

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

Date Time STD Kit (datetimeStd)



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 *DatTime* component regardless if this component is on the wire sheet or not.



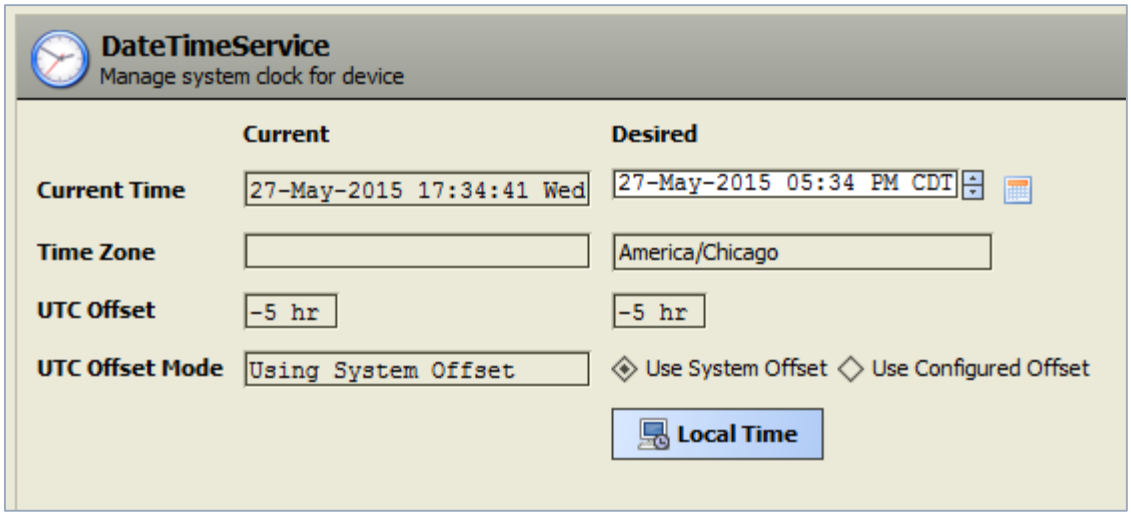
Property	Value
DateTim	
Name	DateTim
Meta	67174401
Nanos	540507745000000...
Hour	16
Minute	2
Second	25
Year	2017
Month	2
Day	15
DayOfWeek	3
UtcOffset	-18000
OsUtcOffset	true
Tz	

Please Note when Using Contemporary Control's Controllers with SAE

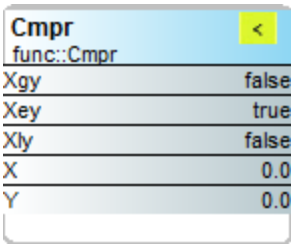
When using Contemporary Controls' controllers, make sure that the OsUtcOffset slot is set to *true* in the Property pane. Just click on the DateTim component and its properties will appear on the right side of the screen. To avoid confusing time settings, do not set the time with this component. Set 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.

Please Note when Using Contemporary Control's Controllers with Workbench

By double clicking the DateTim component, you will see the setup screen below. When using Contemporary Controls' controllers, make sure that the Use System Offset option is selected as shown. 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.

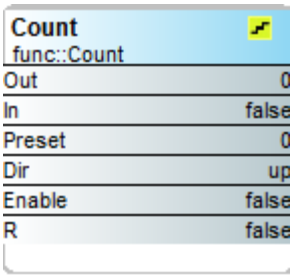


Function Kit (func)



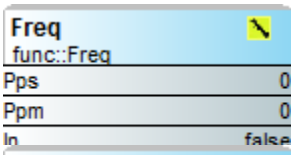
Comparison math — comparison (<=>) of two floats

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



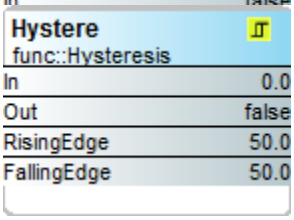
Integer counter — up/down counter with integer output

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.



Pulse frequency — calculates the input pulse frequency

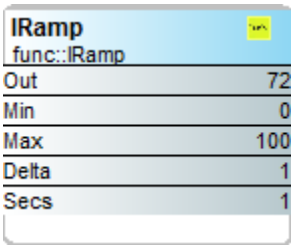
Pps = number of pulses per second of In
Ppm = number of pulses per minute of In



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

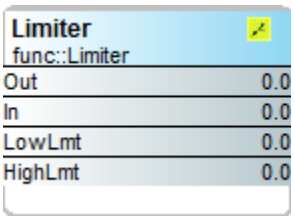
There are two internal floats called *Rising Edge* and *Falling Edge* which are configurable. If *Rising Edge* is greater than *Falling Edge*, then the following is true.

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.



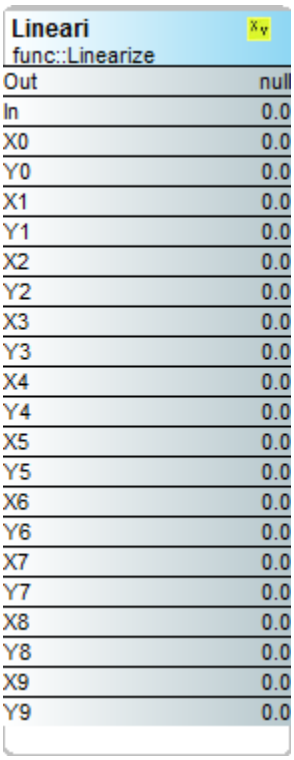
IRamp — generates a repeating triangular wave with an integer output

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 — 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

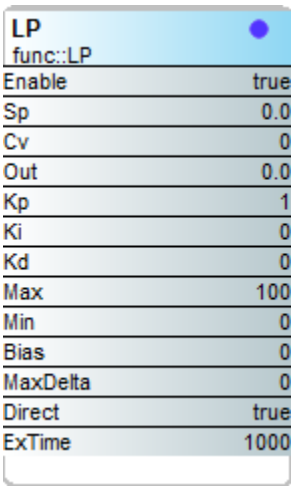


Linearize — piecewise linearization of a float

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



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 particular 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.

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:

$$Out = Kp(e) + Bias \text{ where } e = Cv - Sp \text{ and Bias equals zero}$$

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:

$$Out = Kp(e + Ki\int e\,dt) \qquad \text{(Bias is disabled when Ki is non-zero.)}$$

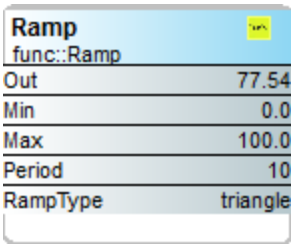
If there remains an error signal ($e \neq 0$), then the integral of the error over time will continue to drive the output until the error is driven to zero. The amount of action is determined by the *Ki* term. Notice that the integral term in the equation is also multiplied by the proportional gain before 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:

$$Out = Kp(e + Ki\int e\,dt + Kd\,de/dt)$$

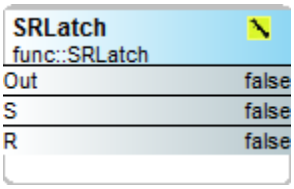
For processes with extremely long reaction times, derivative control could be helpful in reducing overshoot. *Kd* is entered in seconds. As mentioned before, it is seldom used because tuning a control loop with three parameters can be challenging.

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

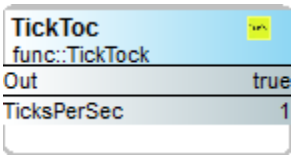
There are four configurable float parameters — *Min, Max, Period* and *Ramp Type*. For every scan cycle, the output increments by one unit until the output equals the *Max* value at which time it decrements until *Min* is reached. The result is a triangular wave with limits of *Max* and *Min* if *Ramp 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

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

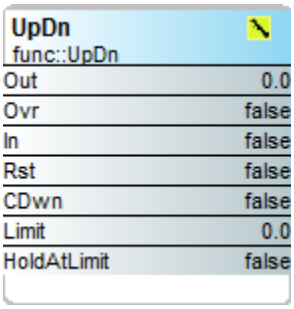
If S goes true and R does not change, then Out = true and remains true.
If R goes true and S does not change, then Out = false and remains false.
If both S and R go true on the same scan, then Out = false and remains false.



Ticking clock — an astable oscillator used as a time base

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



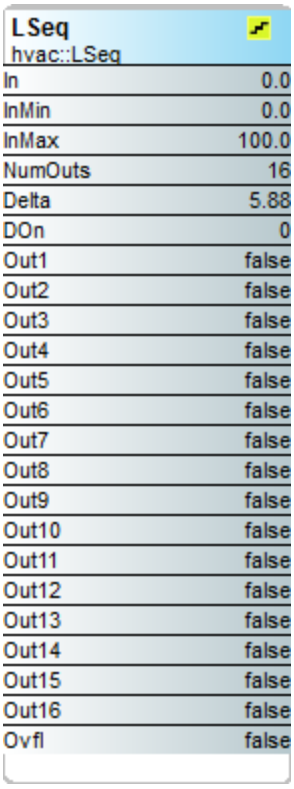
Float counter — up/down counter with float output

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

HVAC Kit (hvac)



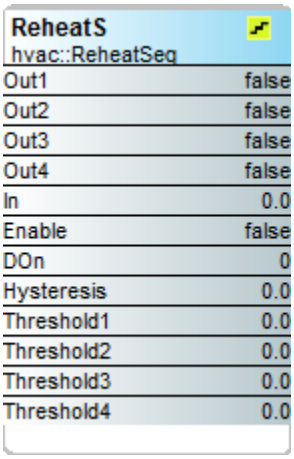
Linear Sequencer — bar graph representation of input value

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.

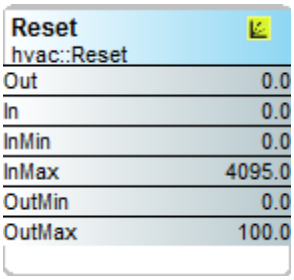


Reheat Sequence — linear sequence up to four outputs

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

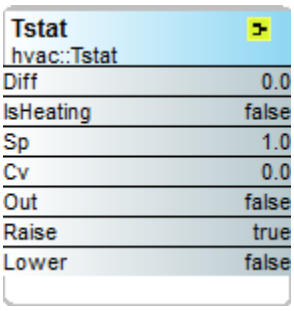
These outputs will remain true until the input value falls below the corresponding threshold value by an amount greater than the configurable parameter *Hysteresis*. Output signal *D On* indicates how many outputs are true. Configurable parameter *Enable* must be true otherwise all outputs will be false.



Reset — 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*



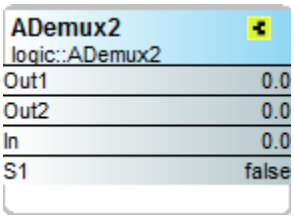
Thermostat — on/off temperature controller

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

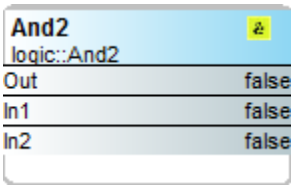
Logic Kit (logic)



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

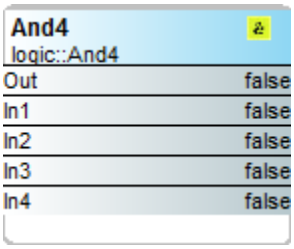
If S1 is false then Out1 = In while Out2 = the last value of In just before S1 changed.

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



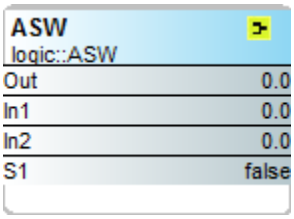
Two-input Boolean product — two-input AND gate

Out = In1 • In2



Four-input Boolean product — four-input AND gate

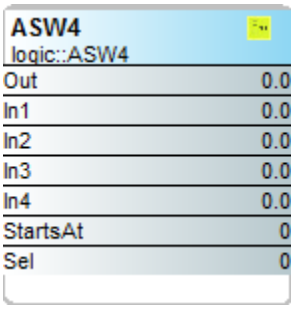
Out = In1 • In2 • In3 • In4



Analog switch — selection between two float variables

If S1 is false, then Out = In1

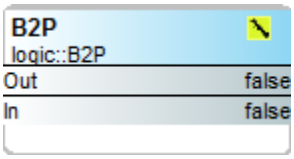
If S1 is true, then Out = In2



Analog switch — selection between four floats

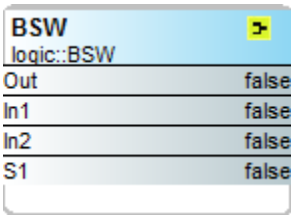
Configurable integer parameter Starts At sets the base selection.

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



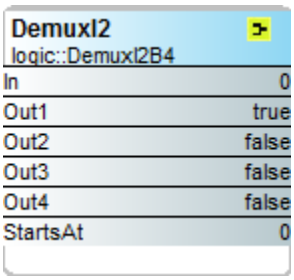
Binary to pulse — simple mono-stable oscillator (single-shot)

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



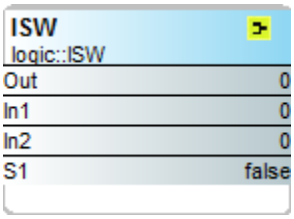
Boolean Switch — selection between two Boolean variables

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



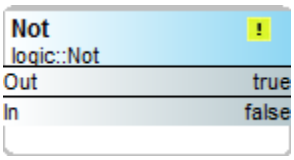
Four-output Demux — integer to Boolean de-multiplexer

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



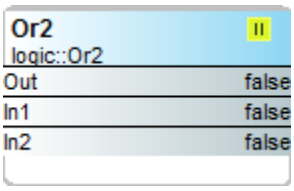
Integer switch — selection between two integer variables

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



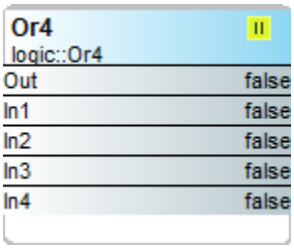
Not — inverts the state of a Boolean

Out = $\overline{In}$



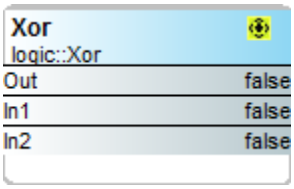
Two-input Boolean sum — two-input OR gate

Out = In1 | *In2*



Four-input Boolean sum — four-input OR gate

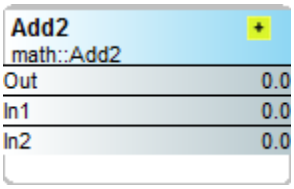
Out = In1 | *In2* | *In3* | *In4*



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

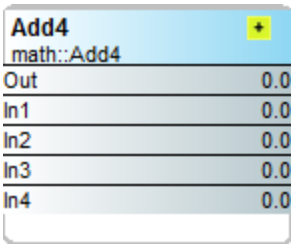
Out = In1 $\oplus$ *In2* = $\overline{In1}$ • $\overline{In2}$ | *In1* • *In2*

Math Kit (math)



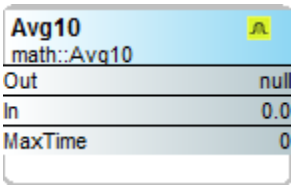
Two-input addition — results in the addition of two floats

Out = In1 + In2



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

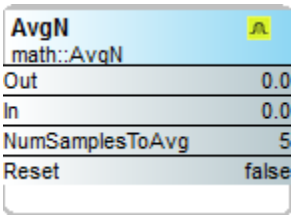
Out = In1 + In2 + In3 + In4



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

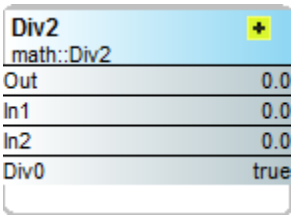
The float input *In* is sampled once every scan and stored. If the input does not change in value on the next scan, it is not sampled again — unless sufficient time passes that exceeds the internal integer *Max Time* with units of milliseconds. In this instance the input is sampled and treated as another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



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

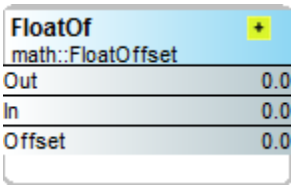
Out = (sum of the last N values)/N

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.



Divide two — results in the division of two floats

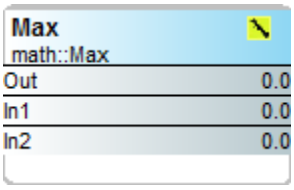
Out = In1/In2
Div0 = true if In2 is equal to zero



Float offset — float shifted by a fixed amount

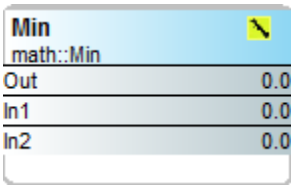
Out = In + Offset

Offset is a configurable float.



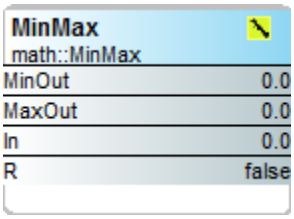
Maximum selector — selects the greater of two inputs

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



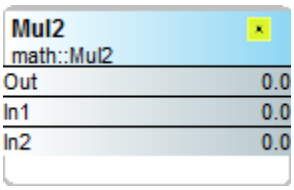
Minimum selector — selects the lesser of two inputs

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



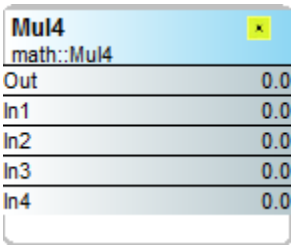
Min/Max detector — records both the maximum and minimum values of a float

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.



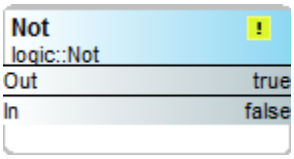
Multiply two — results in the multiplication of two floats

Out = In1 ∗ In2



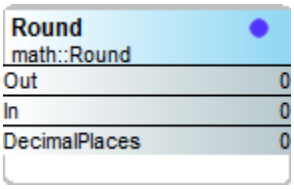
Multiply four — results in the multiplication of four floats

Out = In1 ∗ In2 ∗ In3 ∗ In4



Negate — changes the sign of a float

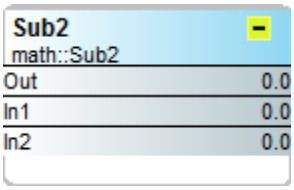
Out = – In



Round — rounds a float to the nearest N places

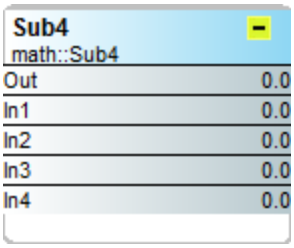
For N = -1, Out = In rounded to the nearest tens
For N = 0, Out = In rounded to the nearest units
For N = 1, Out = In rounded to the nearest tenth's
For N = 2, Out = In rounded to the nearest hundredths
For N = 3, Out = In rounded to the nearest thousandths

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



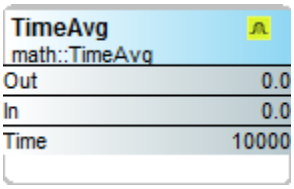
Subtract two — results in the subtraction of two floats

Out = In1 – In2



Subtract four — results in the subtraction of four floats

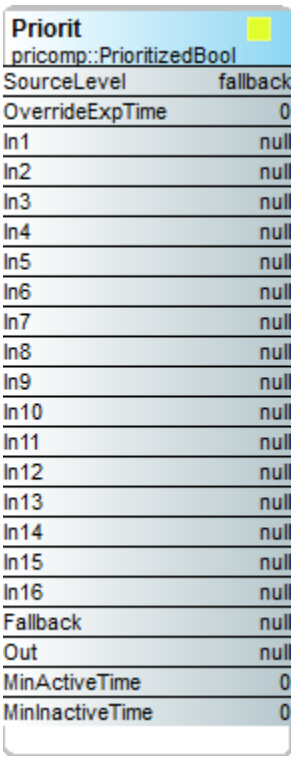
Out = In1 – In2 – In3 – In4



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

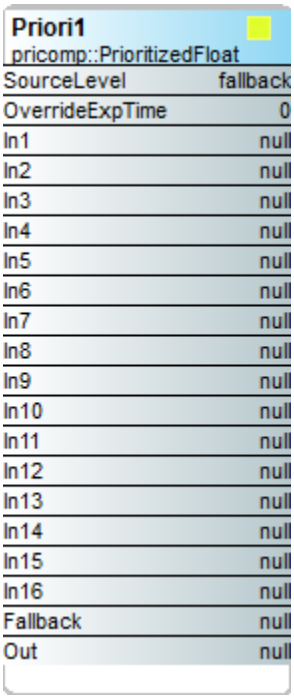
Out = Avg[In] over the integer time in milliseconds.

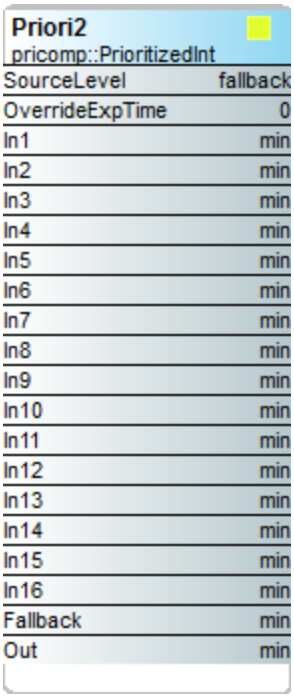
Priority Kit (pricomp)



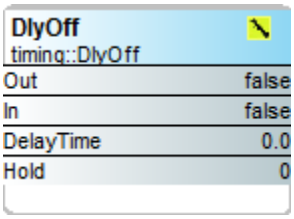
Priority array (Priorit) components exist for Boolean, float and integer variables. Up to 16 levels of priority from In1 to In16 can be assigned. In1 has the highest priority and In16 the lowest. With few exceptions, all can be pinned out. If a priority level is not assigned, it is marked as a Null and 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

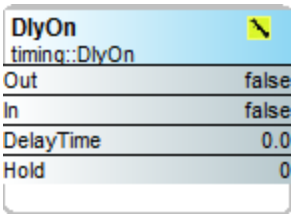

Priori2
pricomp::PrioritizedInt
SourceLevel fallback
OverrideExpTime 0
In1 min
In2 min
In3 min
In4 min
In5 min
In6 min
In7 min
In8 min
In9 min
In10 min
In11 min
In12 min
In13 min
In14 min
In15 min
In16 min
Fallback min
Out min

Timing Kit (timing)



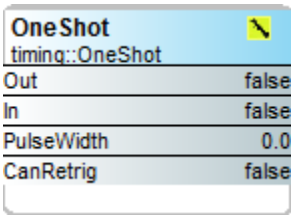
Off delay timer — time delay from a true to false transition of the input

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.



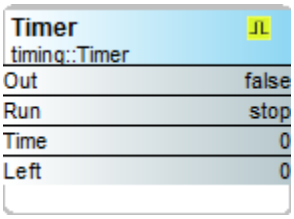
On delay timer — time delay from a false to true transition of the input

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.



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.

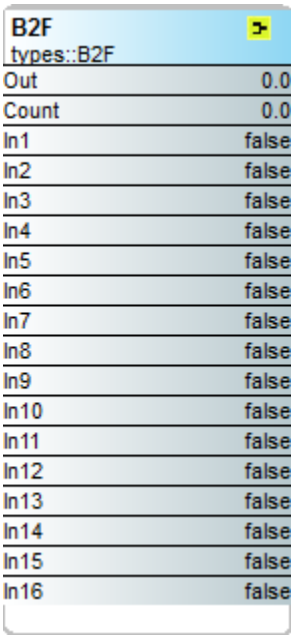


Timed pulse — predefined pulse output

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.

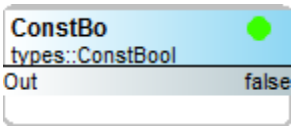
Types Kit (types)



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

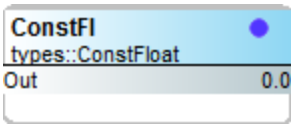
Out = encoded value of binary input with In16 being the MSB and In1 being the LSB

Count = sum of the number of active inputs



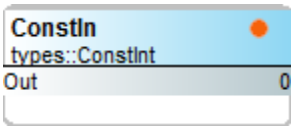
Boolean Constant — a predefined Boolean value

Out = a Boolean value that is internally configurable



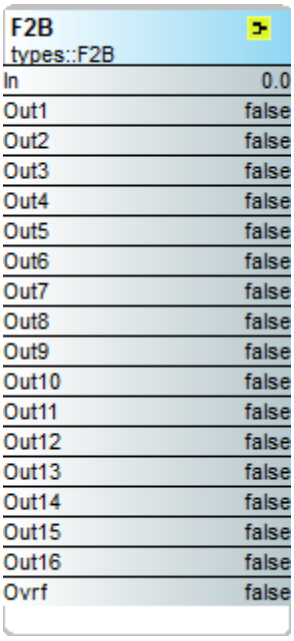
Float Constant — a predefined float value

Out = a float value that is internally configurable



Integer Constant — a predefined integer value

Out = an integer value that is internally configurable

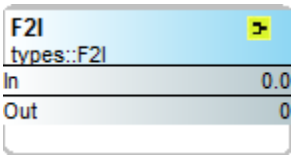


Float to binary decoder — float to 16-bit binary conversion

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

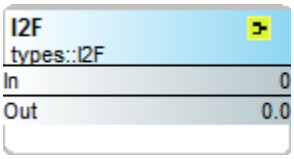
Although the input requires a float, fractional amounts are ignored during the conversion.



Float to integer — float to integer conversion

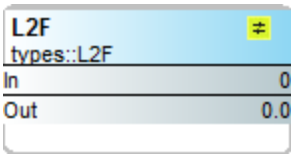
Out = In except that the output will be a whole number

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



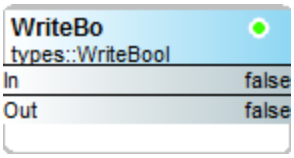
Integer to Float — integer to float conversion

Out = In except that the output will become a float



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

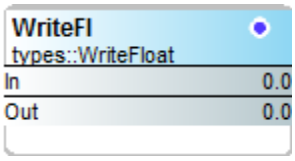
Out = In except that the output will become a float from a 64-bit signed integer



Write Boolean — setting a writable Boolean value

Out = In

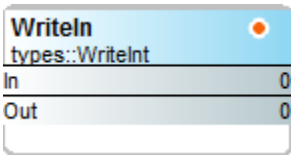
Unlike ConstBo, this component has an input. Could be helpful when transferring a variable between two wire sheets.



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 wire sheets.

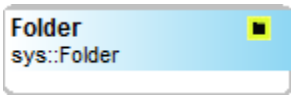


Write Integer — setting an integer value

Out = In

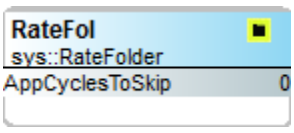
Unlike ConstIn, this component has an input. Could be helpful when transferring a variable between two wire sheets.

Sys Kit (sys)



Folder — gateway to a wire sheet

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



Rate Folder — gateway to a wire sheet 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.