CONTEMPORARY CONTROLS®

Understanding Sedona Core Components using the BAScontrol Toolset

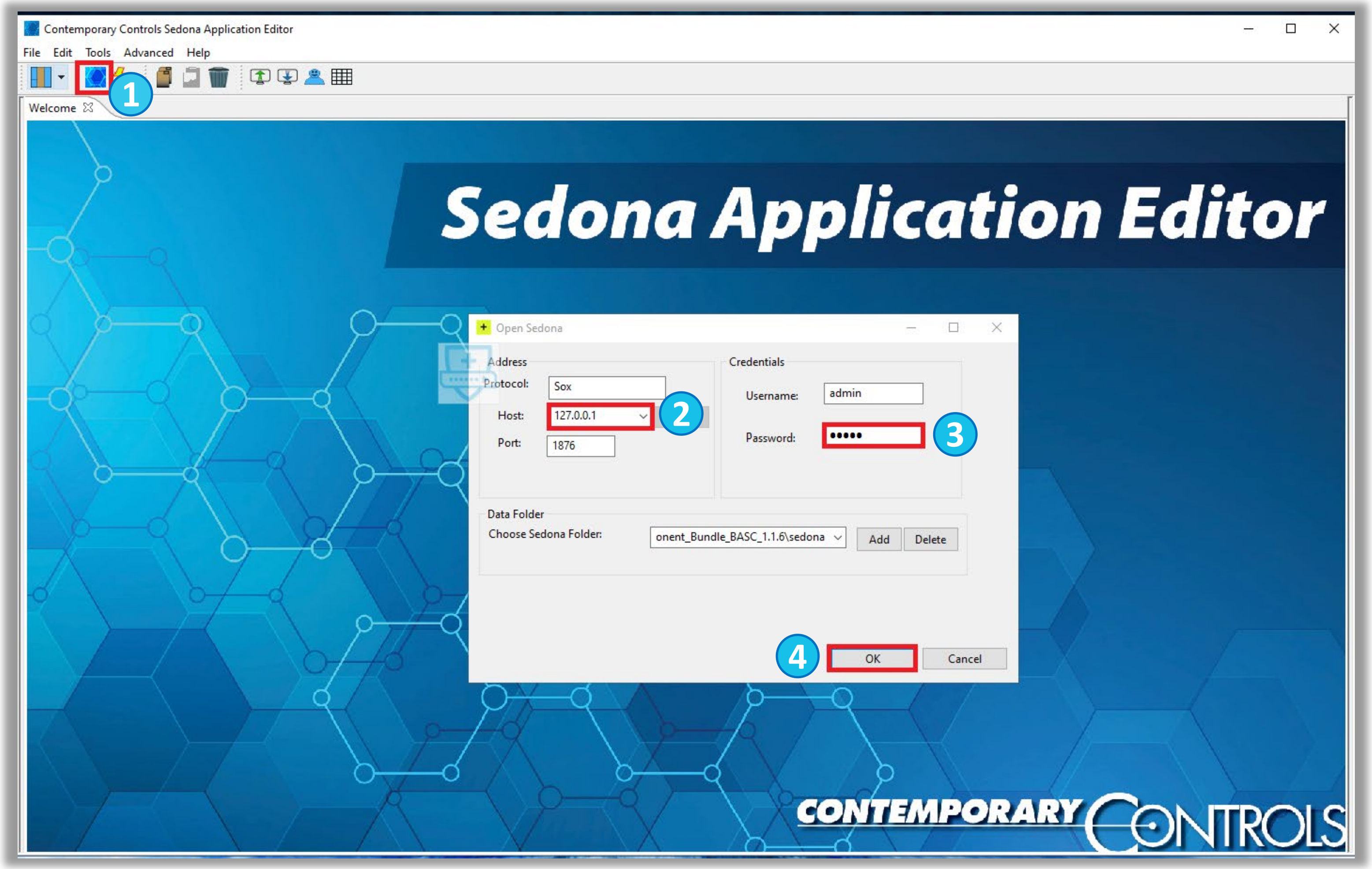
Interconnecting Components on Wiresheets to Create Applications

- Contemporary Controls has developed the BAScontrol Toolset, a free set of Sedona tools operating on a Windows PC, which includes
  - BASemulator – a utility used to emulate controller operation on a Windows PC.
  - Sedona Application Editor (SAE) – an editing tool used to create function block (component) wiresheet applications in the Sedona environment.
  - BASbackup – a project utility which provides a convenient way of storing/restoring and replicating real or emulated controller settings and configurations, as well as Sedona wiresheet applications.
  - For an overview of this toolset, refer to "Introduction to the BAScontrol Toolset."
- This presentation addresses how to use SAE to place Sedona components that are deployed in kits onto a wiresheet and then configured and linked with other components to create applications.
- The following examples demonstrate using this toolset with the BASemulator; however, these examples can be implemented on a real Sedona controller.

Getting Started – Launch SAE

From SAE:

1. Click the “Open Connection” icon on the toolbar.
2. Use address 127.0.0.1 to connect, unless you specified a different address when you launched the BASemulator. This address is always available in the “Host” drop-down selection and cannot be deleted. This is the default address for the BASemulator.
3. Enter the Username and Password. The default credentials are “admin” for both Username and Password.
4. Click “OK” to connect.


Contemporary Controls Sedona Application Editor

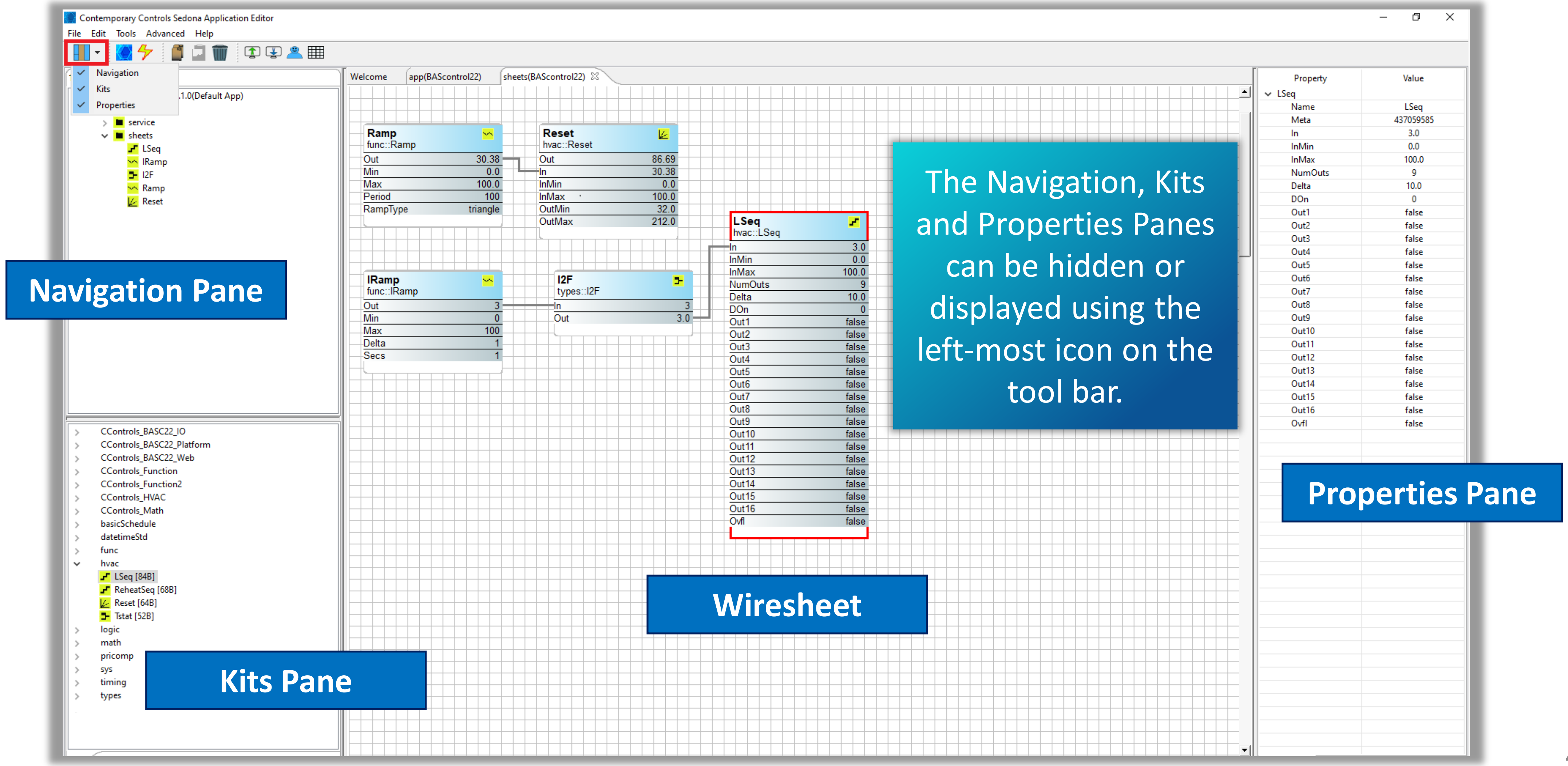
Open Sedona

Address
Protocol: Sox
Host: 127.0.0.1
Port: 1876

Credentials
Username: admin
Password: •••••

Data Folder
Choose Sedona Folder: onent_Bundle_BASC_1.1.6\sedona

Understanding the Wiresheet Structure



The Navigation, Kits and Properties Panes can be hidden or displayed using the left-most icon on the tool bar.

Navigation Pane

Kits Pane

Wiresheet

Properties Pane

Contemporary Controls Sedona Application Editor

File Edit Tools Advanced Help

✓ Navigation
✓ Kits
✓ Properties

.1.0(Default App)

- service
- sheets
  - LSeq
  - IRamp
  - I2F
  - Ramp
  - Reset

Welcome | app(BAScontrol22) | sheets(BAScontrol22)

Ramp	func::Ramp
Out	30.38
Min	0.0
Max	100.0
Period	100
RampType	triangle

Reset	hvac::Reset
Out	86.69
In	30.38
InMin	0.0
InMax	100.0
OutMin	32.0
OutMax	212.0

IRamp	func::IRamp
Out	3
Min	0
Max	100
Delta	1
Secs	1

I2F	types::I2F
In	3
Out	3.0

LSeq	hvac::LSeq
In	3.0
InMin	0.0
InMax	100.0
NumOuts	9
Delta	10.0
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

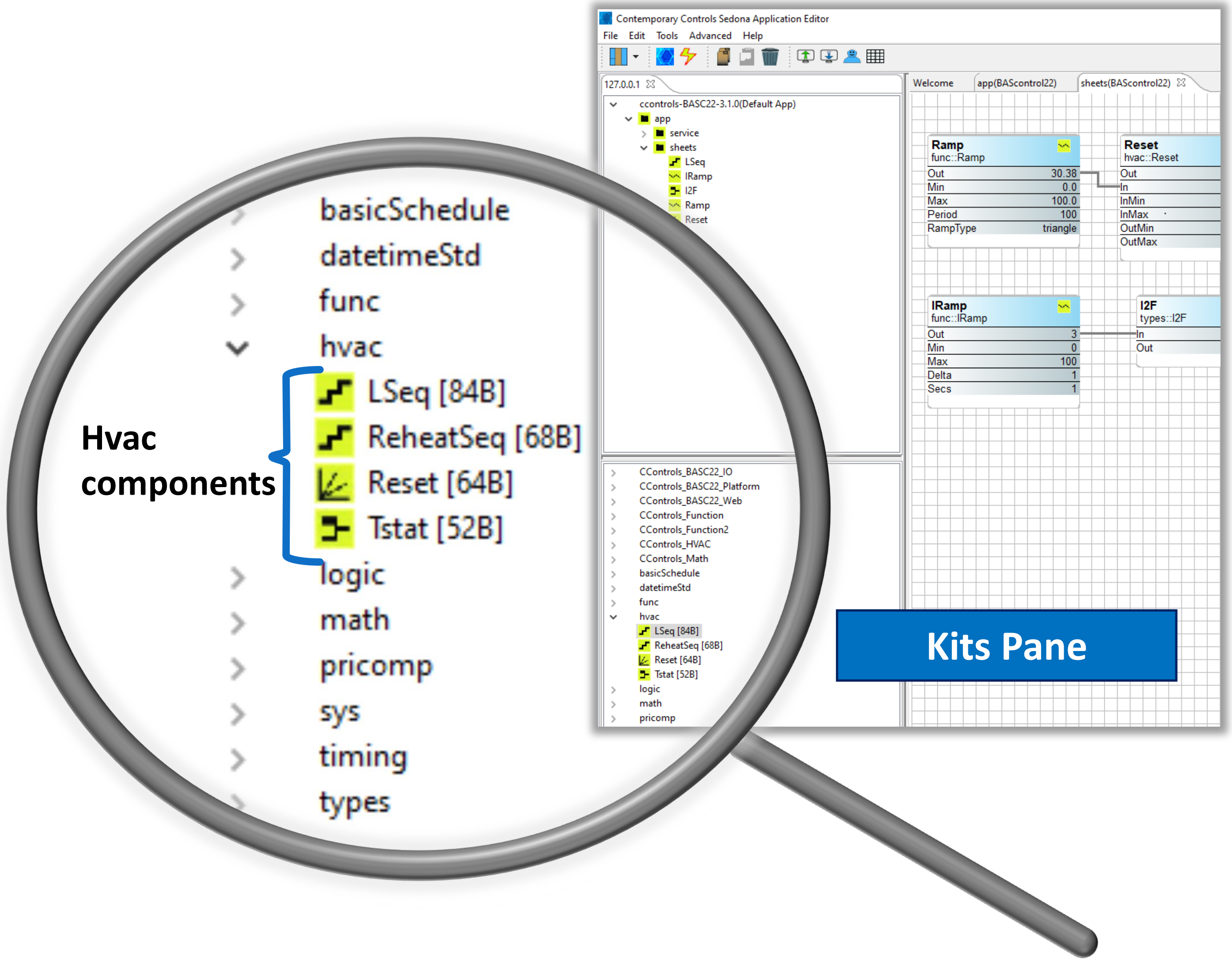
- CControls_BASC22_IO
- CControls_BASC22_Platform
- CControls_BASC22_Web
- CControls_Function
- CControls_Function2
- CControls_HVAC
- CControls_Math
- basicSchedule
- datetimeStd
- func
- hvac
  - LSeq [84B]
  - ReheatSeq [68B]
  - Reset [64B]
  - Tstat [52B]
- logic
- math
- pricomp
- sys
- timing
- types

Property	Value
LSeq	
Name	LSeq
Meta	437059585
In	3.0
InMin	0.0
InMax	100.0
NumOuts	9
Delta	10.0
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

Understanding Components

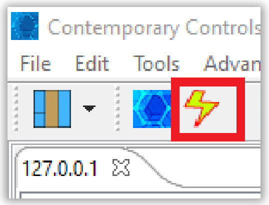
Components are typically sorted by function and deployed in kits.

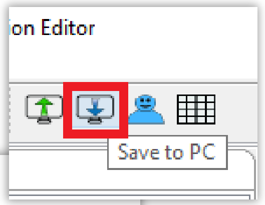
The components discussed in this presentation can be found in one of the core kits shown on the right that come installed with every Sedona controller.

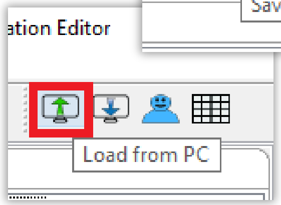


Saving Your Application and Project

- 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.

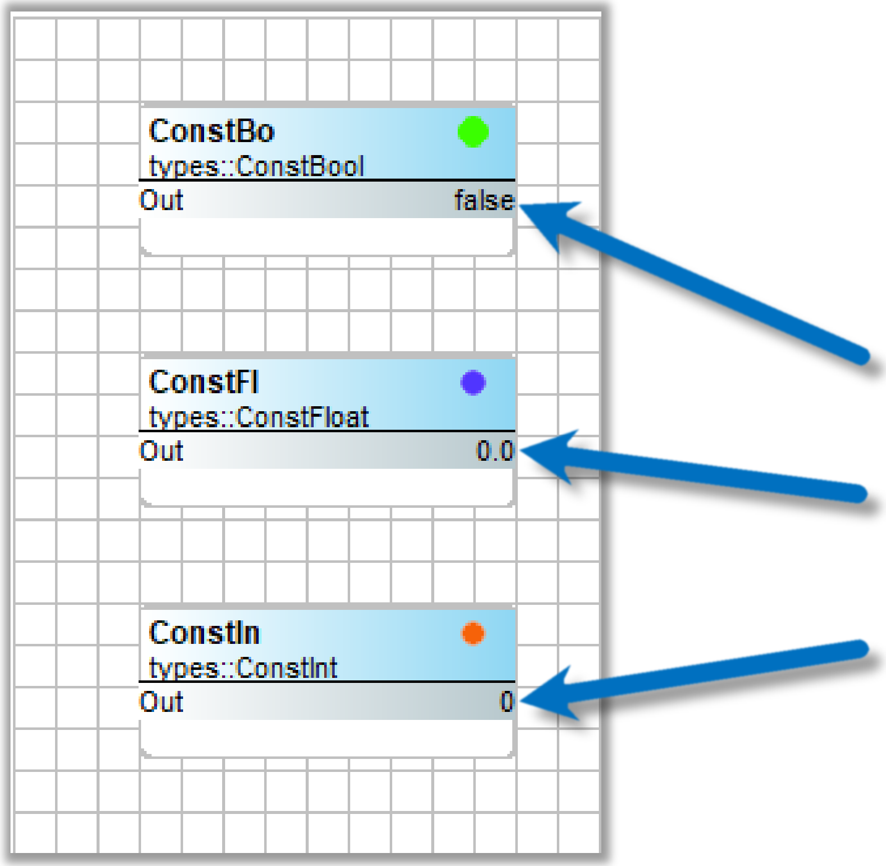






For an more information on saving your application and project, and the BAScontrol Toolset, refer to "Introduction to the BAScontrol Toolset."

Variable Types



Notice the format of the component output:

Boolean has true/false

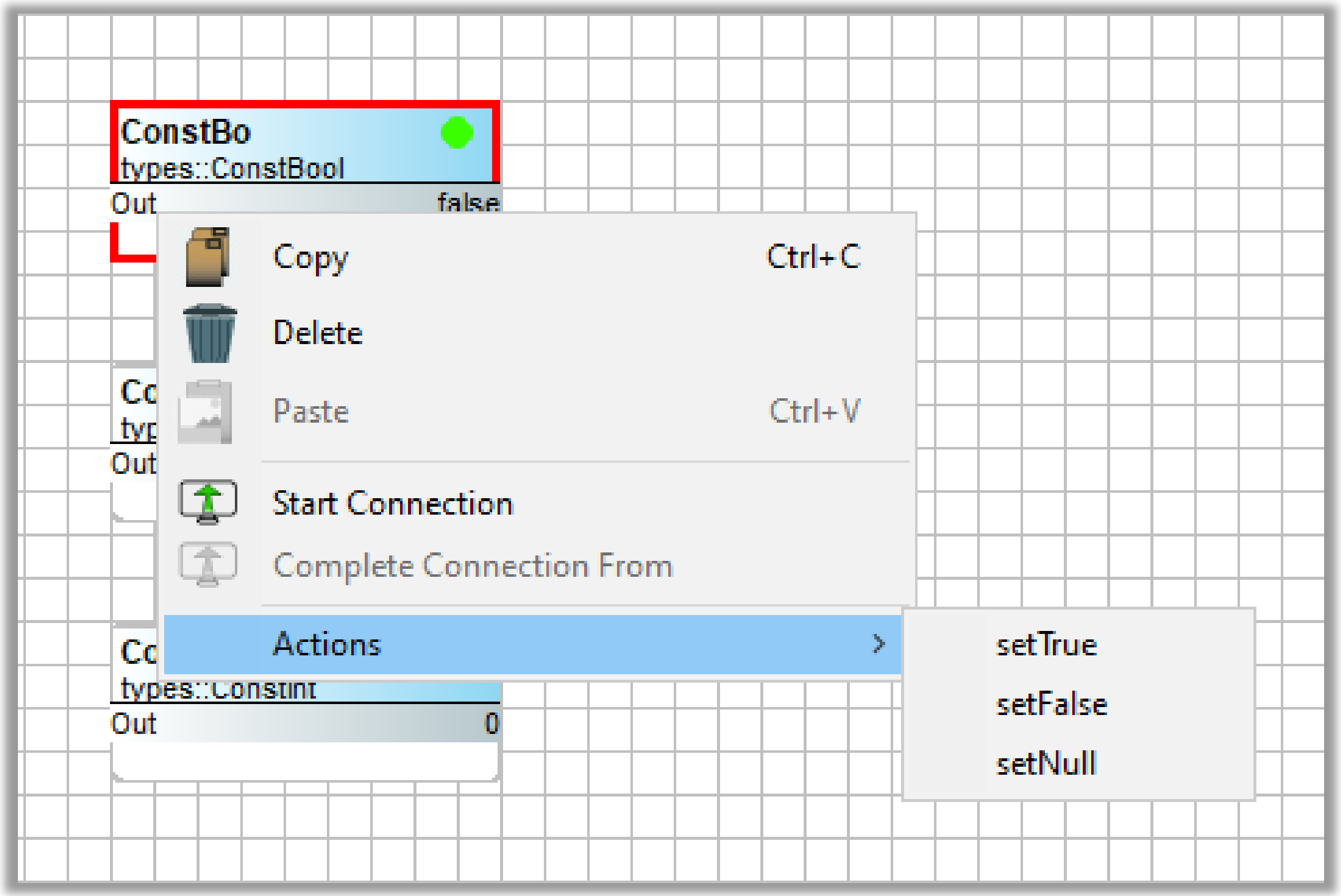
Floats have a decimal point

Integers have no decimal point

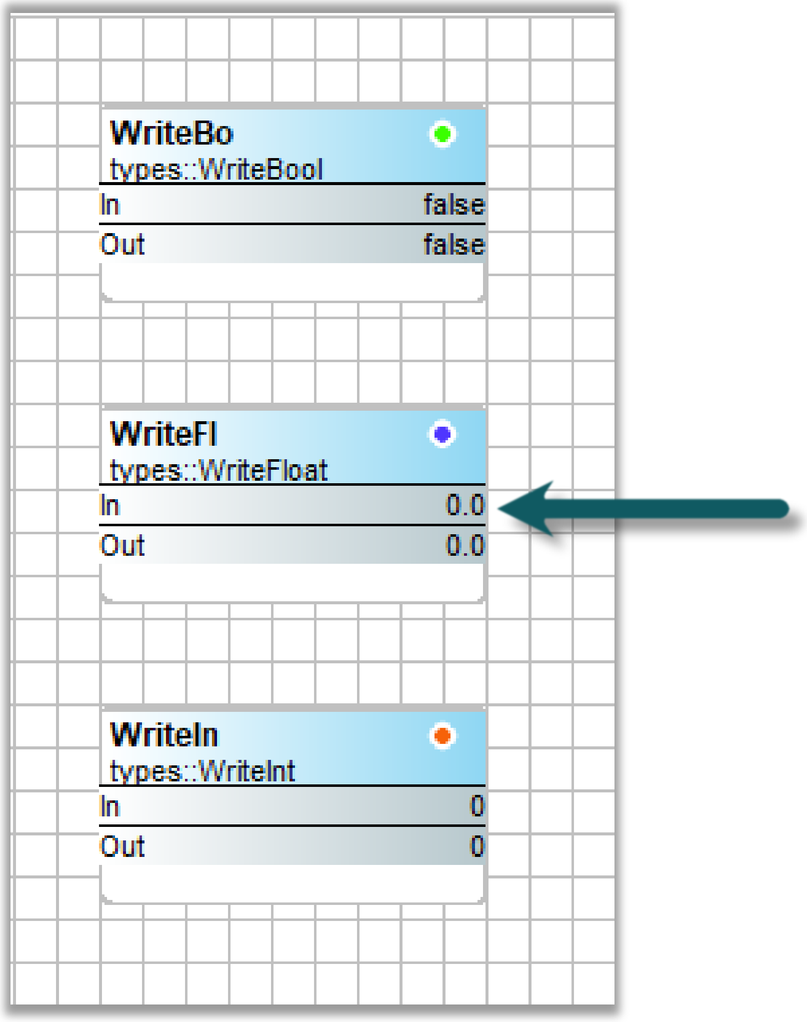
These are constant components that can be configured. However, they must be saved, or the settings will be lost.

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.



Using Write Constants

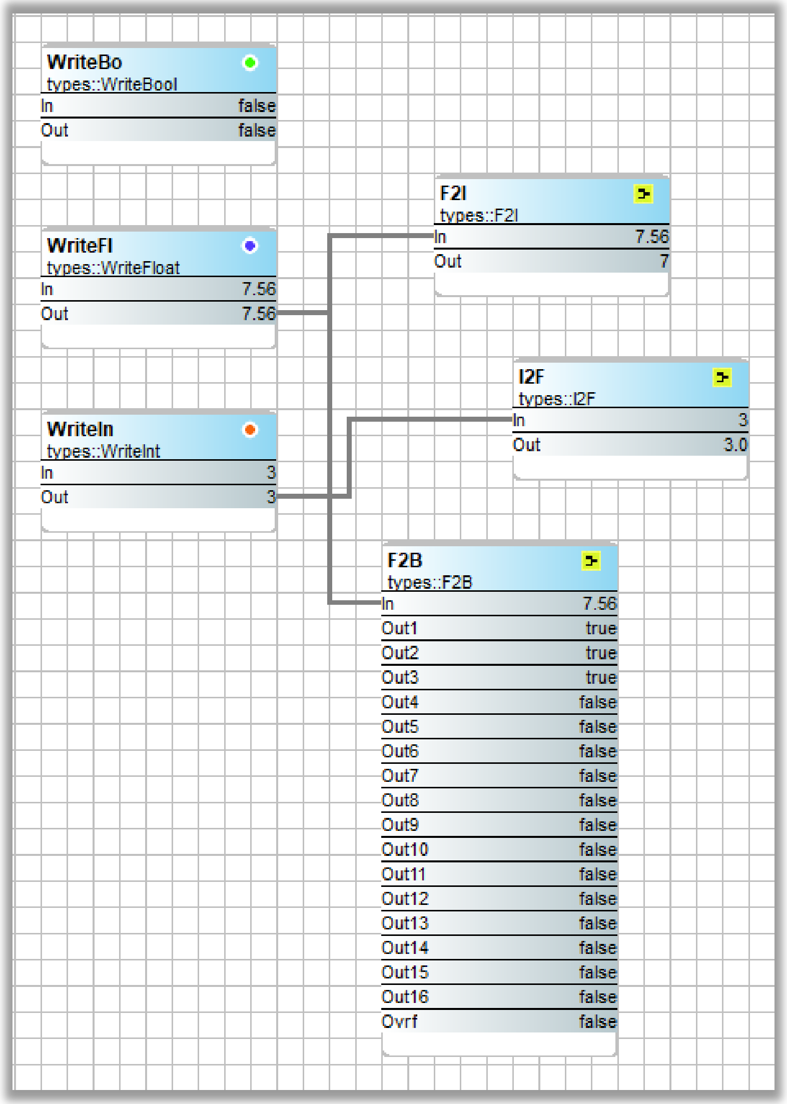

WriteBo
types::WriteBool
In false
Out false

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.

Converting Between Component Types



```
WriteBo
types::WriteBool
In        false
Out       false

WriteFl
types::WriteFloat
In        7.56
Out       7.56

WriteIn
types::WriteInt
In        3
Out       3

F2I
types::F2I
In        7.56
Out       7

I2F
types::I2F
In        3
Out       3.0

F2B
types::F2B
In        7.56
Out1      true
Out2      true
Out3      true
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-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.

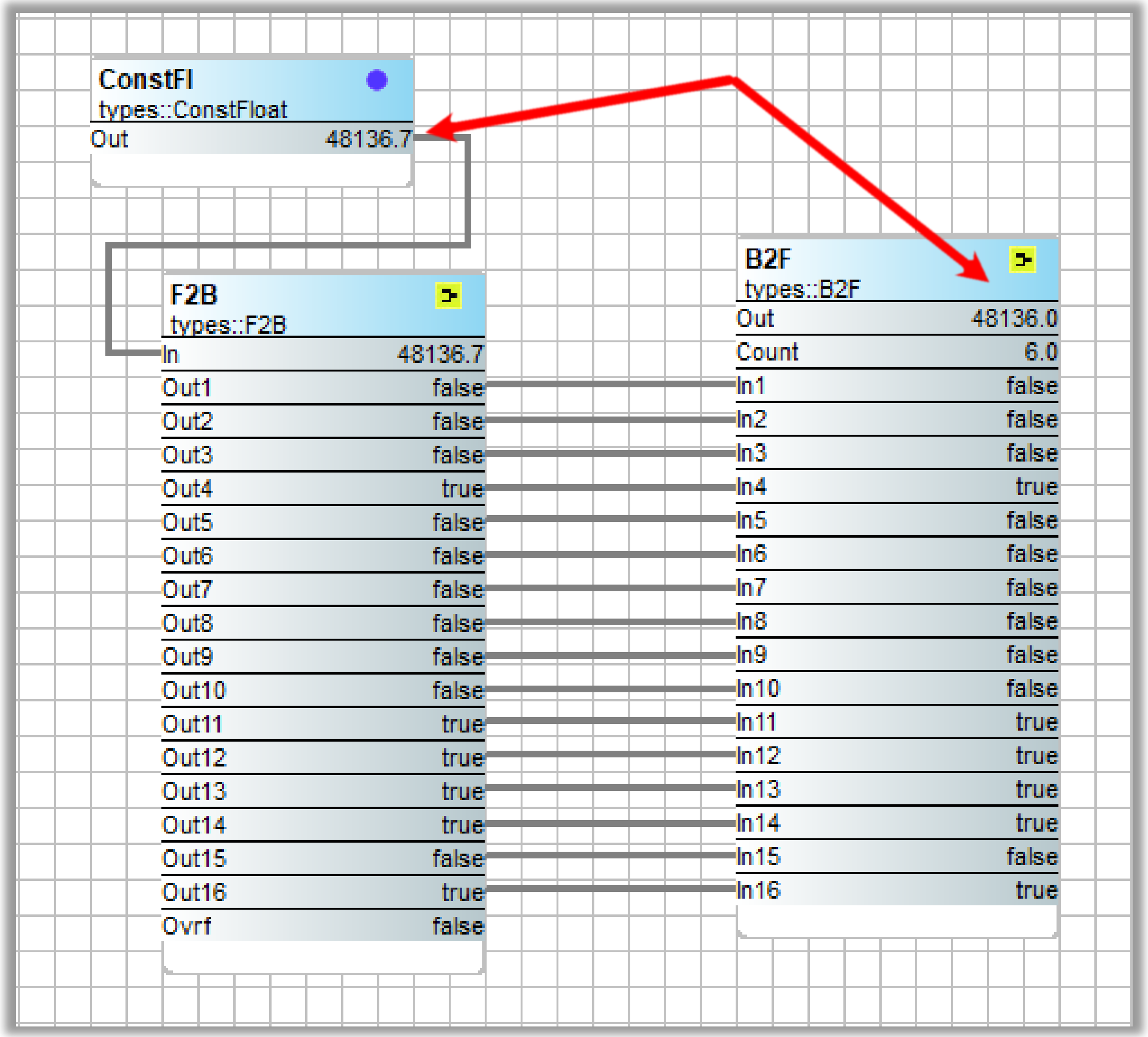
You can also convert a float to a binary using the Float-to-Binary component. However, notice that the resulting 0000 0000 0000 0111 binary representation is actually a decimal 7 and again the original float value was truncated.

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

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.



ConstFl
types::ConstFloat

Out	48136.7

F2B
types::F2B

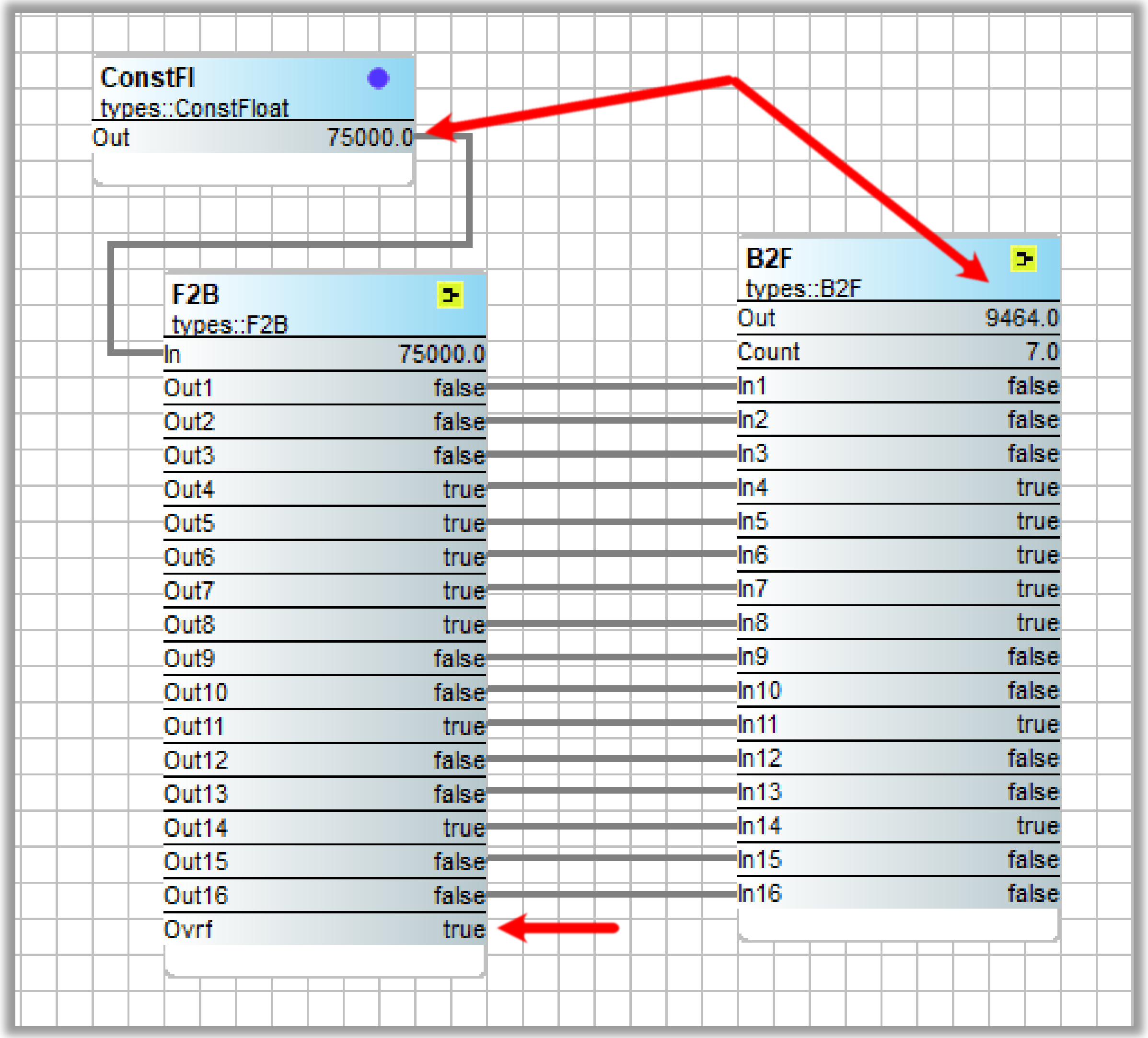
In	48136.7
Out1	false
Out2	false
Out3	false
Out4	true
Out5	false
Out6	false
Out7	false
Out8	false
Out9	false
Out10	false
Out11	true
Out12	true
Out13	true
Out14	true
Out15	false
Out16	true
Ovrf	false

B2F
types::B2F

Out	48136.0
Count	6.0
In1	false
In2	false
In3	false
In4	true
In5	false
In6	false
In7	false
In8	false
In9	false
In10	false
In11	true
In12	true
In13	true
In14	true
In15	false
In16	true

Converting from Float-to-Boolean and Boolean-to-Float (continued)

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.



ConstFl
types::ConstFloat

Out	75000.0

F2B
types::F2B

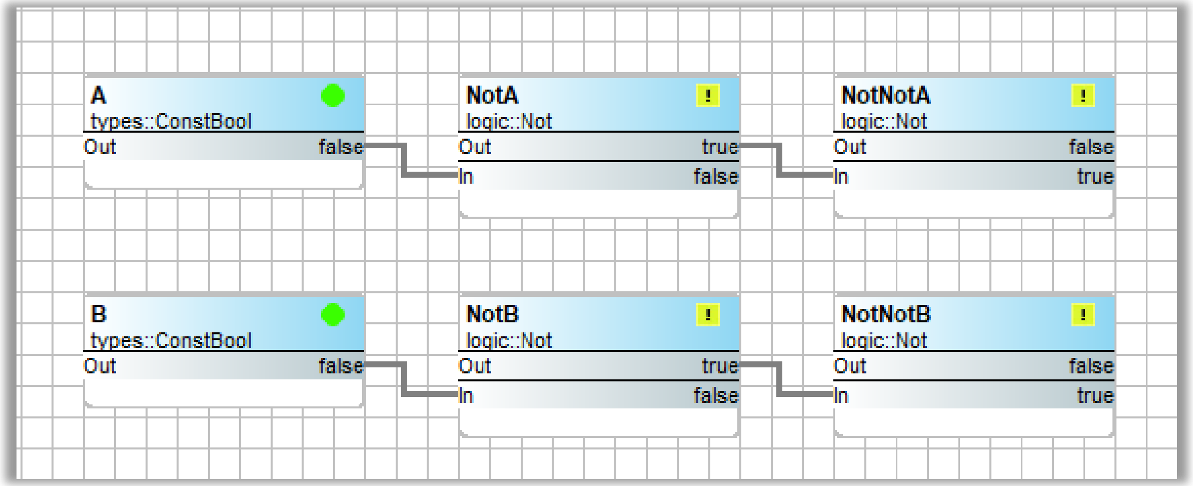
In	75000.0
Out1	false
Out2	false
Out3	false
Out4	true
Out5	true
Out6	true
Out7	true
Out8	true
Out9	false
Out10	false
Out11	true
Out12	false
Out13	false
Out14	true
Out15	false
Out16	false
Ovrf	true

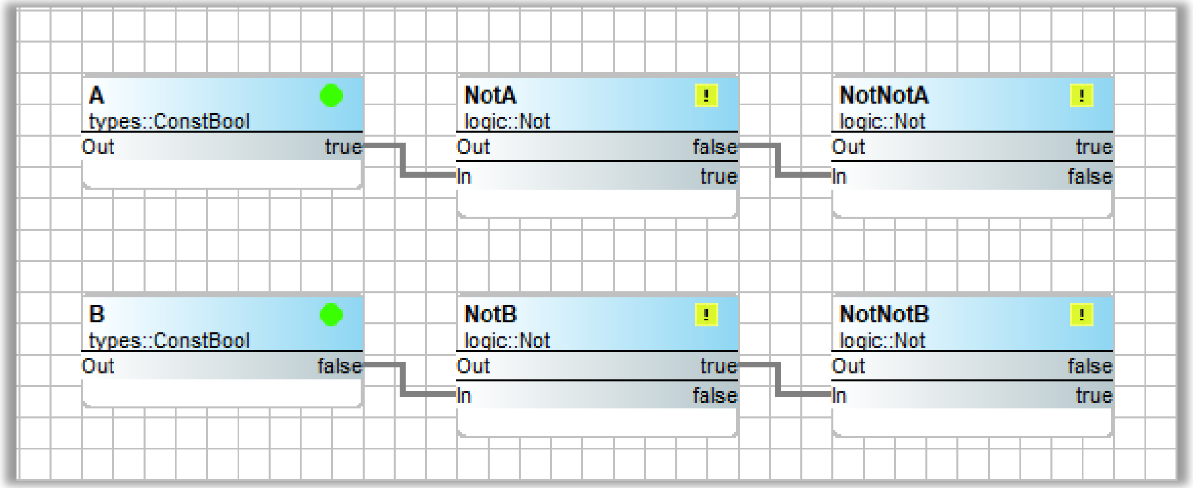
B2F
types::B2F

Out	9464.0
Count	7.0
In1	false
In2	false
In3	false
In4	true
In5	true
In6	true
In7	true
In8	true
In9	false
In10	false
In11	true
In12	false
In13	false
In14	true
In15	false
In16	false

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.

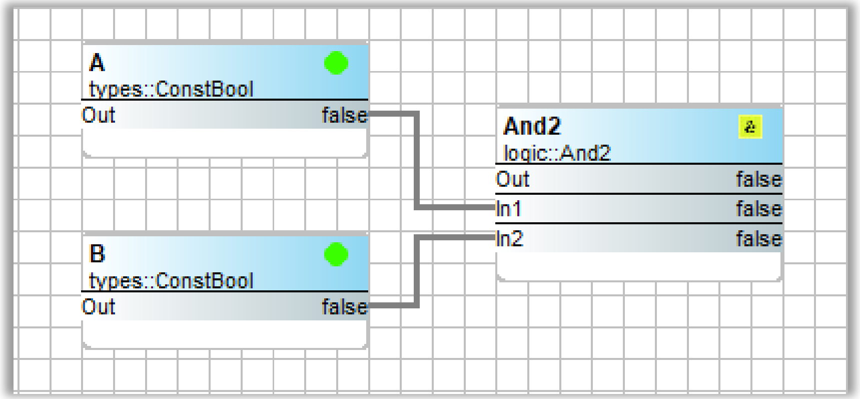




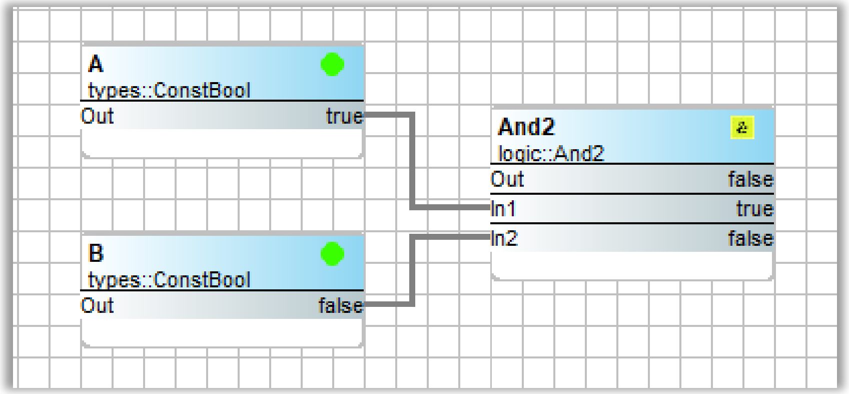
There are two Boolean variables A and B which are set to be false. Both feed a Not component that is usually called an inverter because it changes the initial variable to the opposite state, which it true. Going into another inverter changes the state back to the original states of A and B.

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.

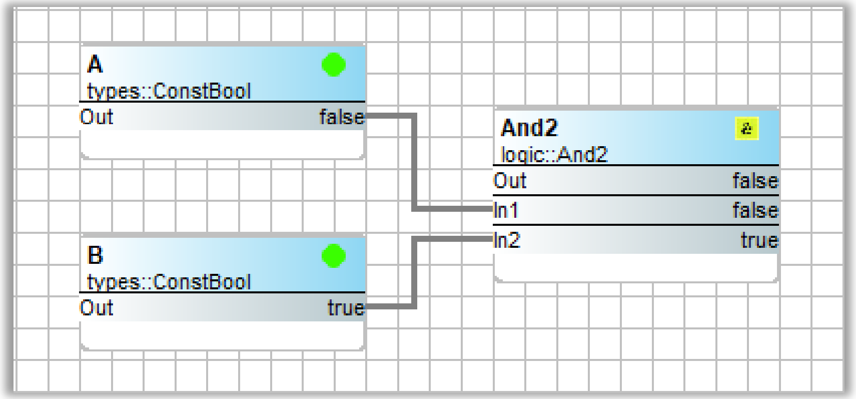
Boolean Product — “ANDing” Boolean Variables



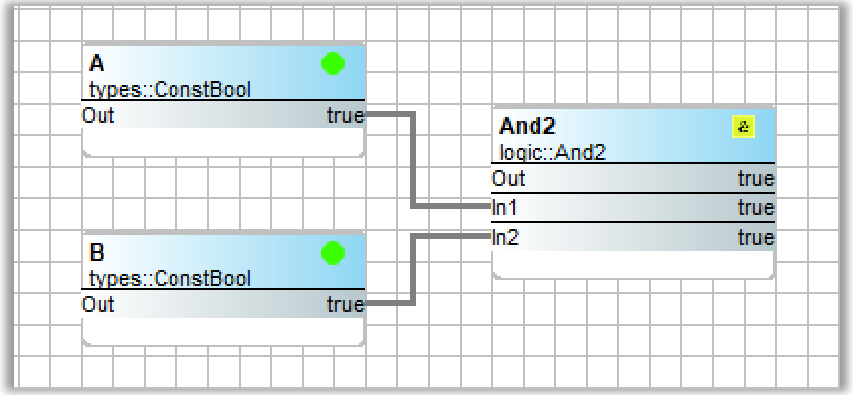
The AND component if frequently called an AND gate. If A is false and B is false, then the output is false.



For an AND gate, If A is true and B is false, then the output is false.

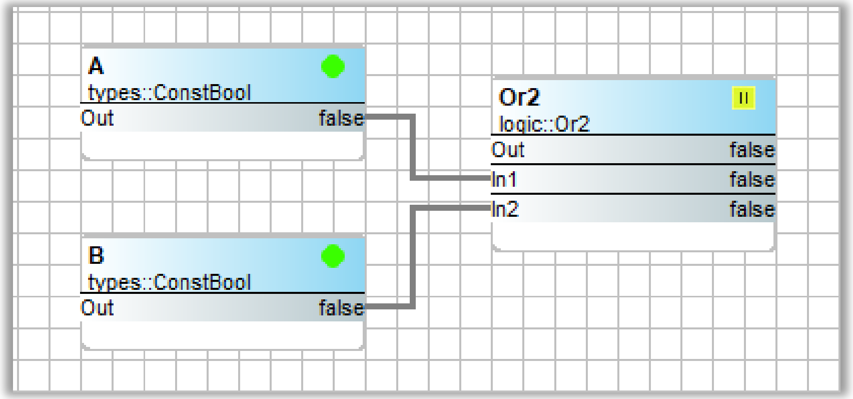


For an AND gate, If A is false and B is true, then the output is false.

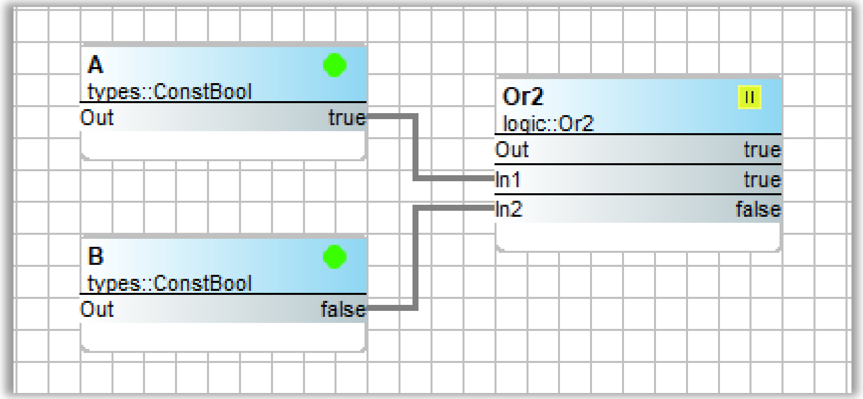


For an AND gate, If A is true and B is true, then the output is true.

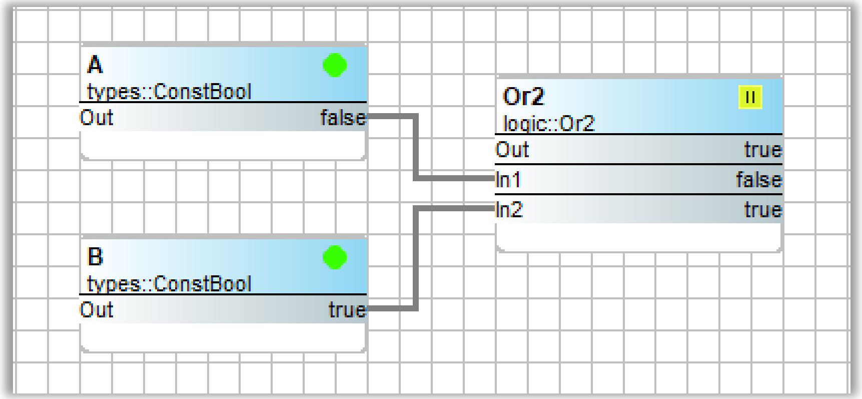
Boolean Sum — "ORing" Boolean Variables



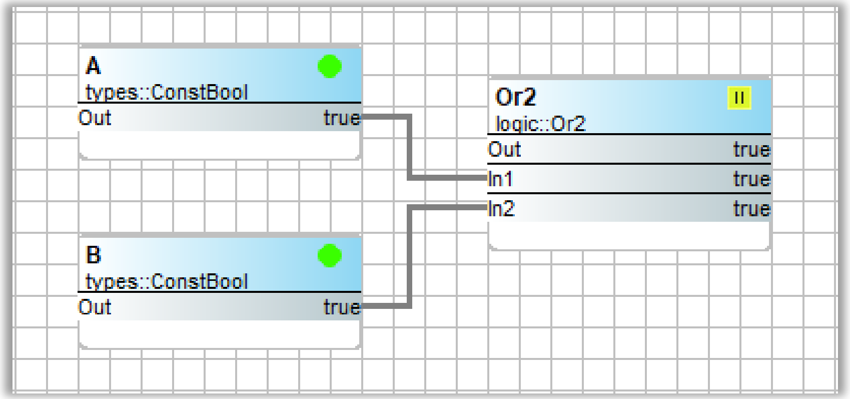
The OR component if frequently called an OR gate. If A is false and B is false, then the output is false.



For an OR gate, If A is true and B is false, then the output is true.

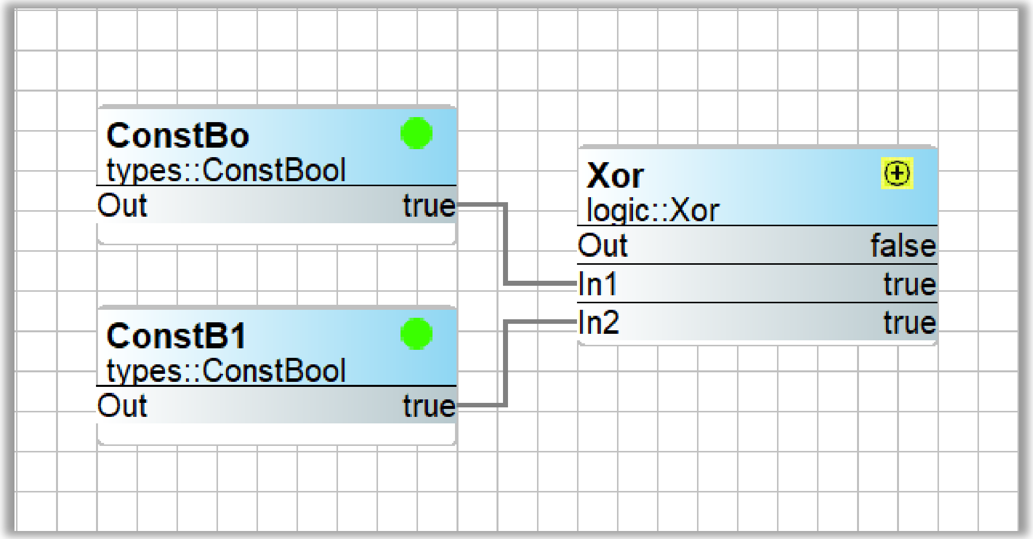


For an OR gate, If A is false and B is true, then the output is true.



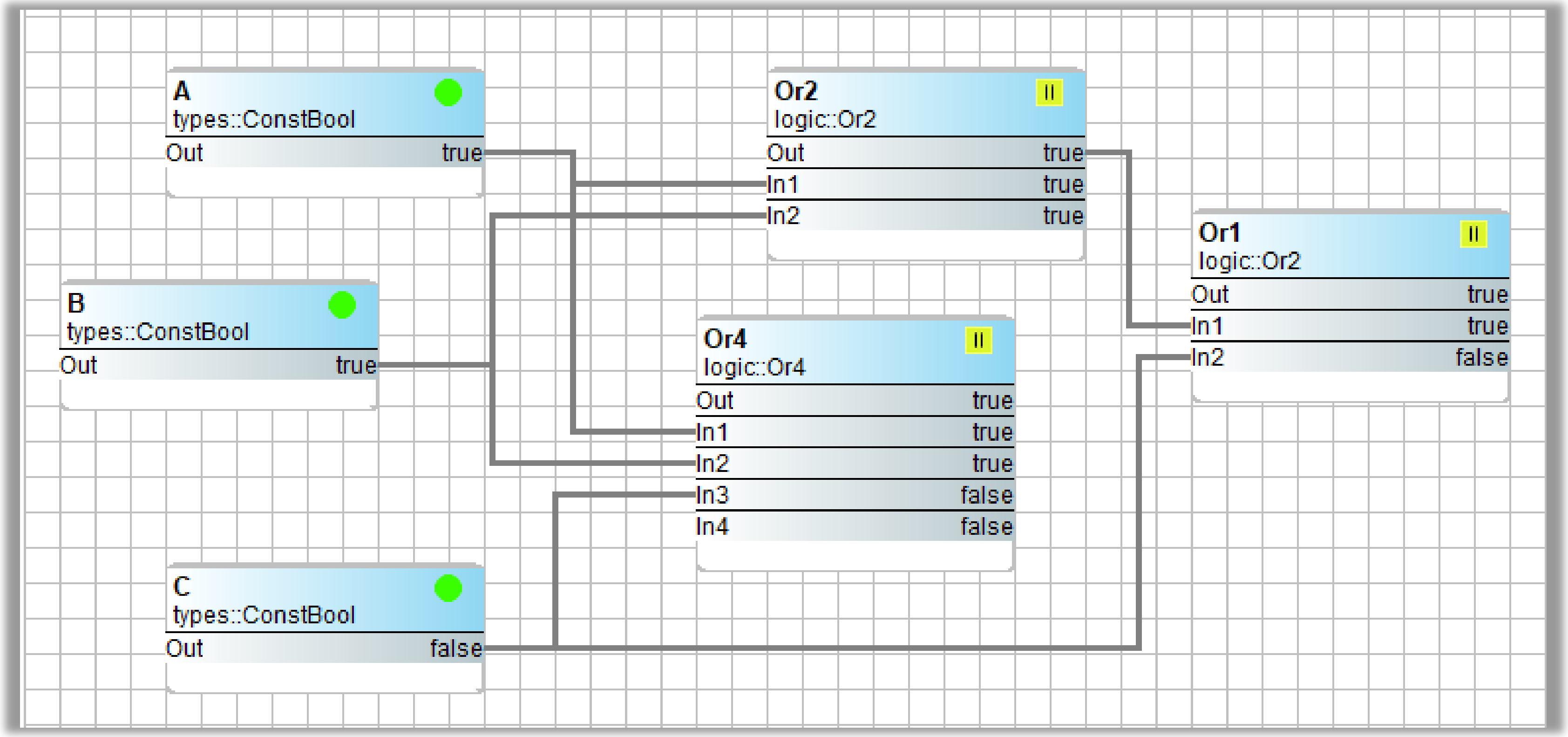
For an OR gate, If A is true and B is true, then the output is true.

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.

Cascading Logic Blocks and Unused Inputs

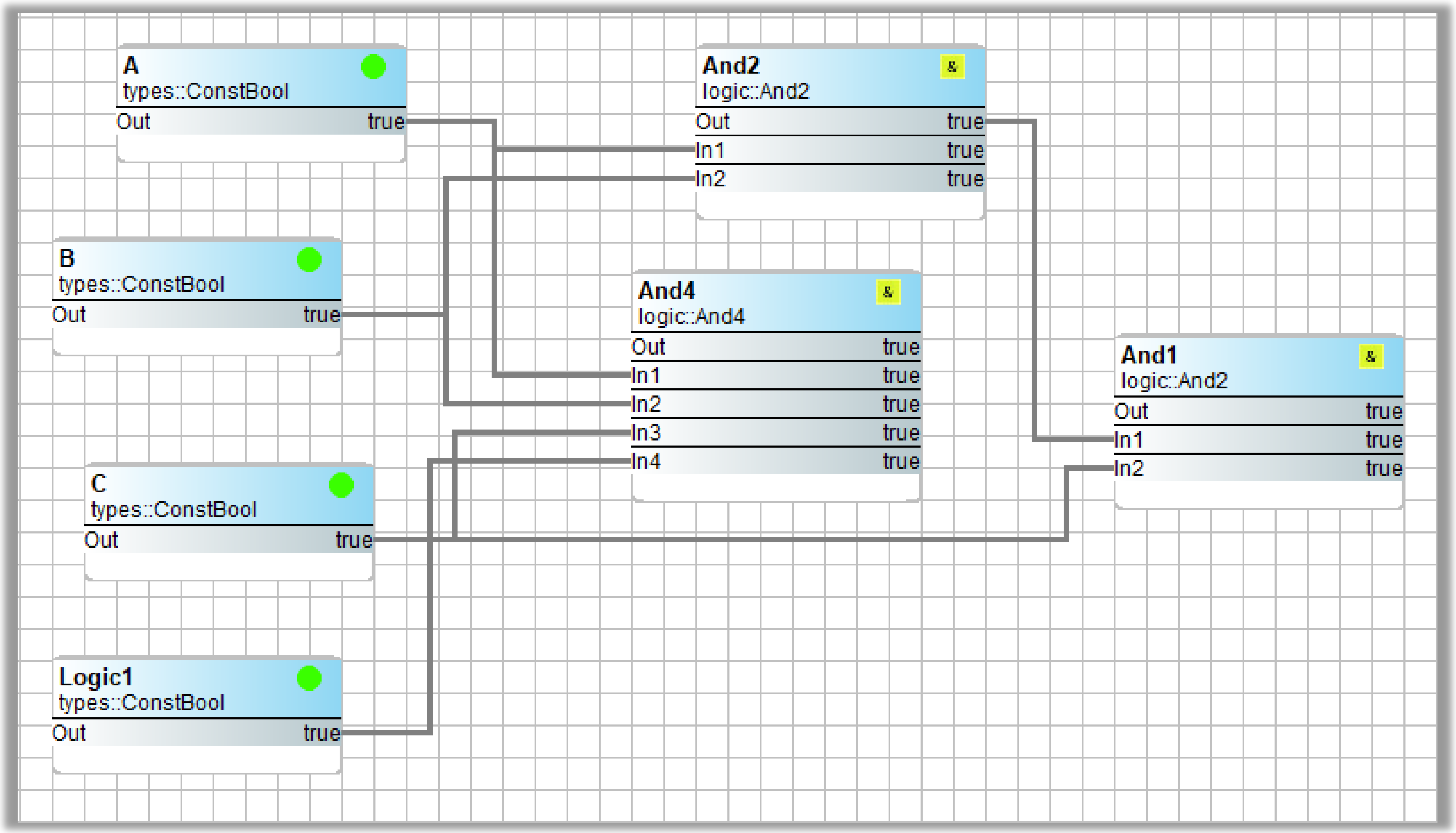


Four-input OR gates operate the same allowing more variables to be added to the logic.

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.

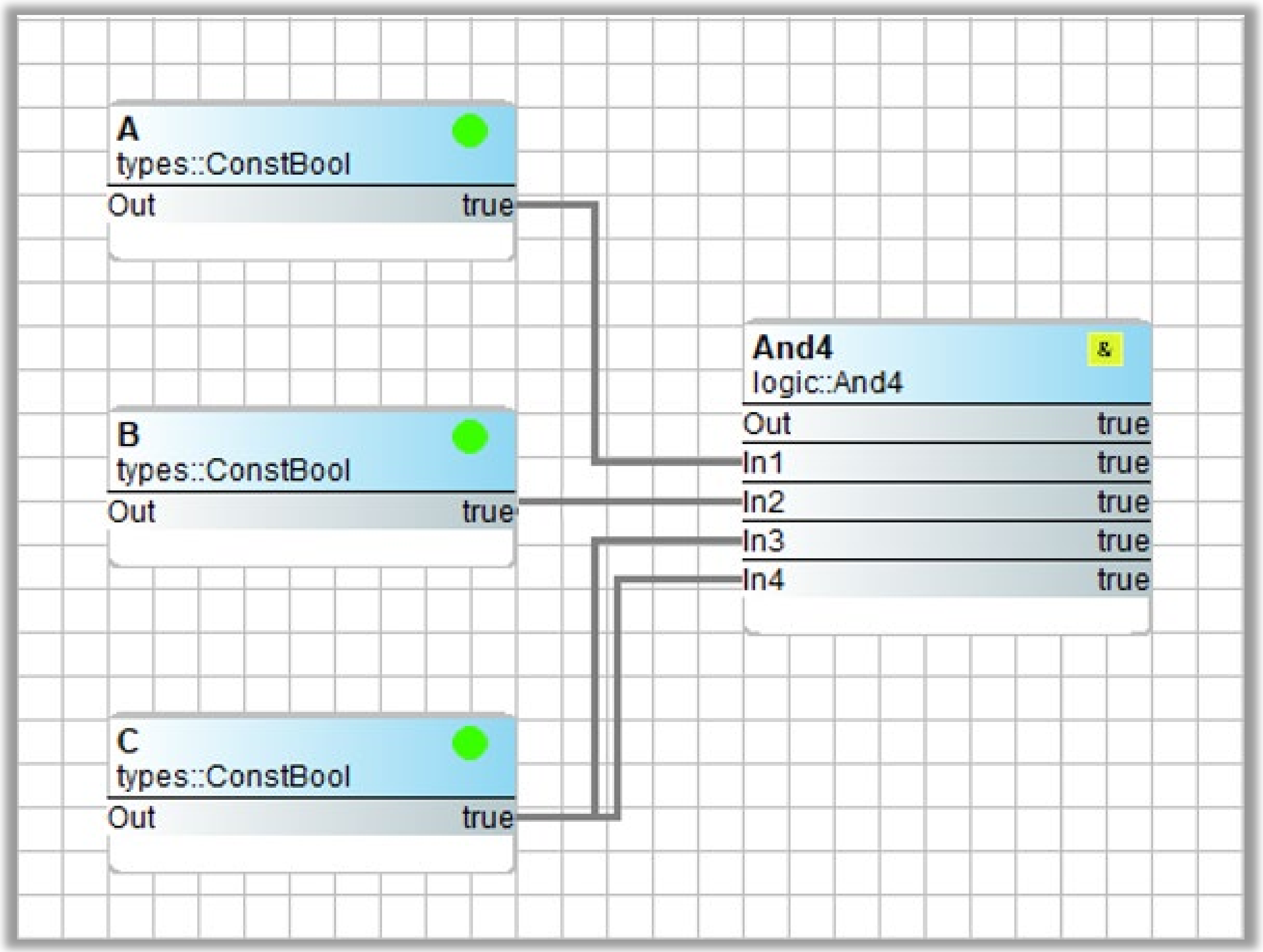
Cascading Logic Blocks and Unused Inputs (continued)



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.

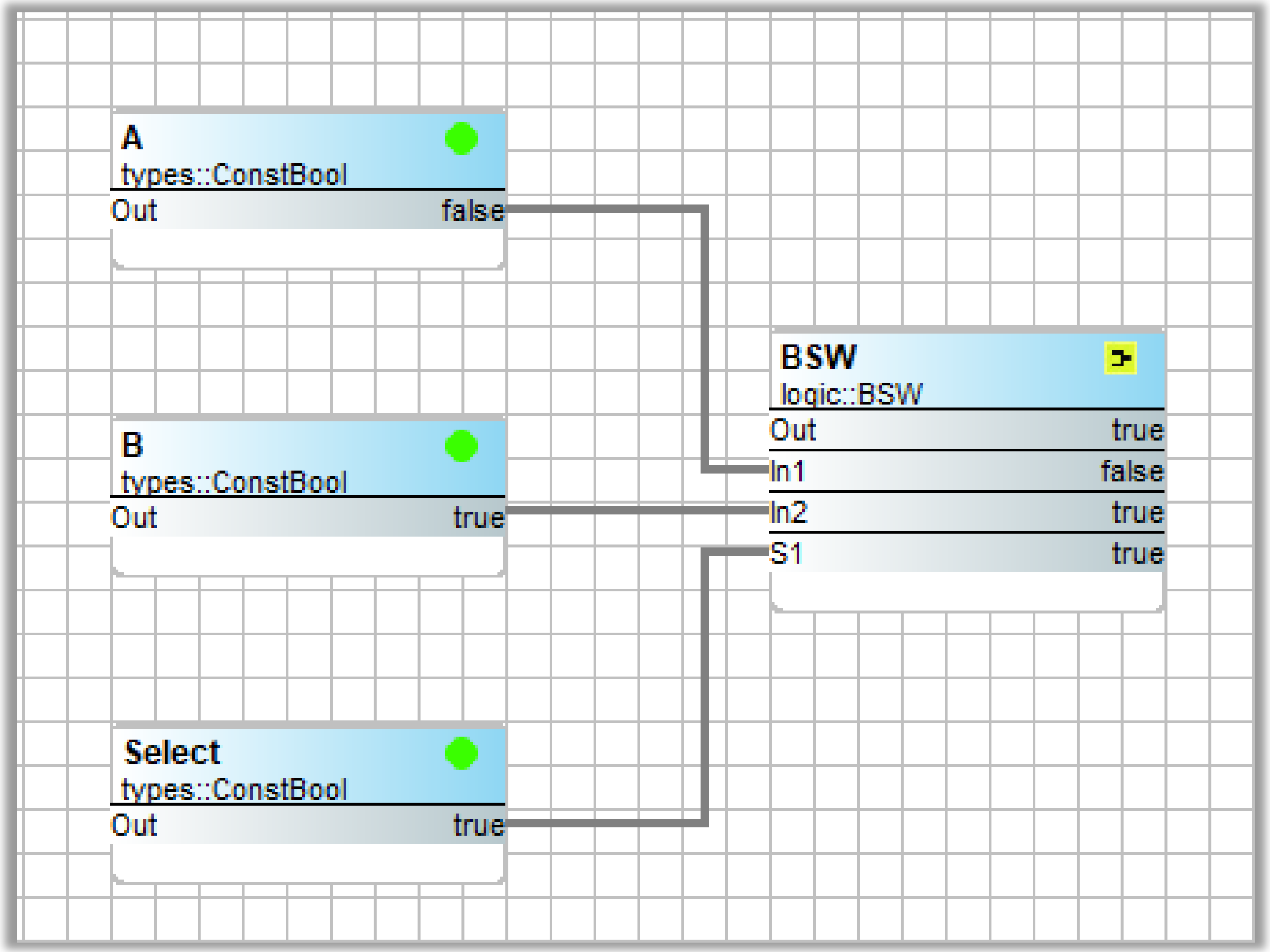
Cascading AND gates are also a possibility when using more than two variables.

Cascading Logic Blocks and Unused Inputs (continued)



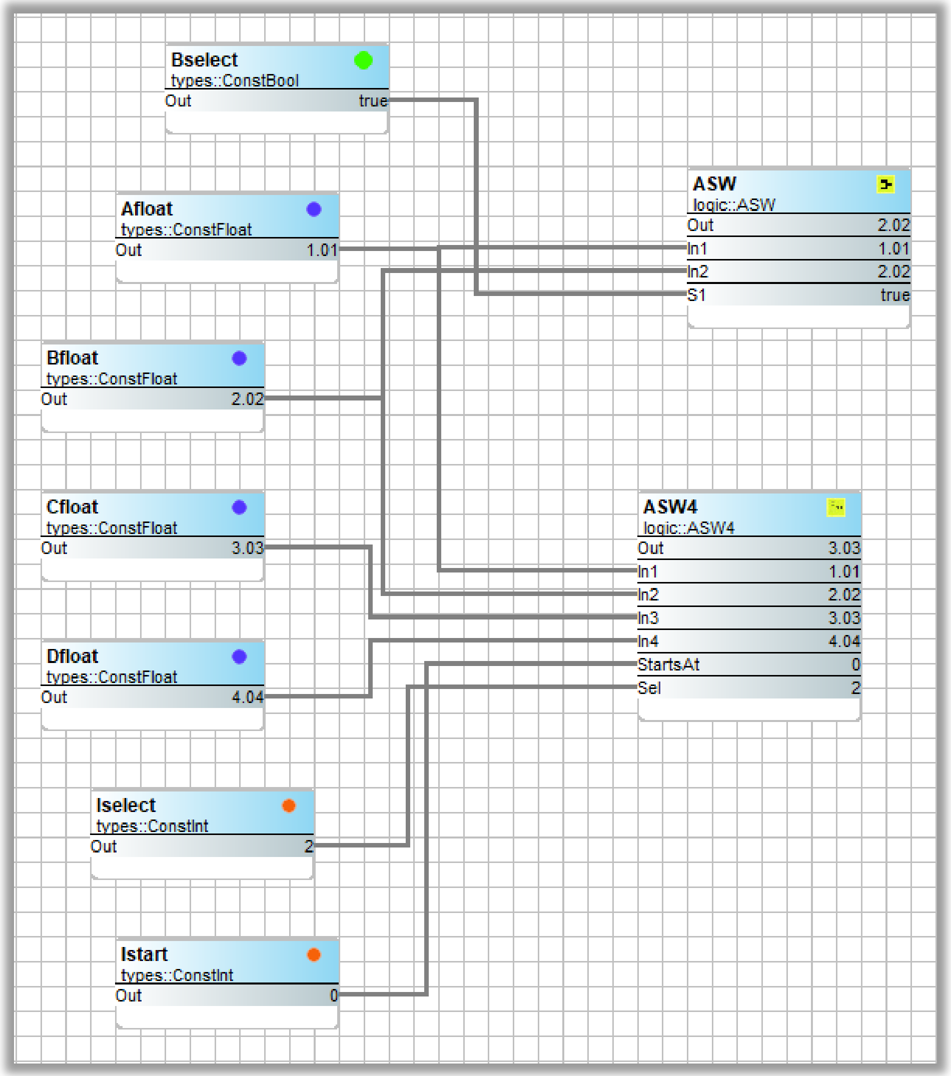
Here is another way of handling an unused input when three variables are connected to a four-input AND gate. Attach the unused input to one of the variables. It does not matter which one is used.

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.

Selecting Boolean, Float or Integer (continued)

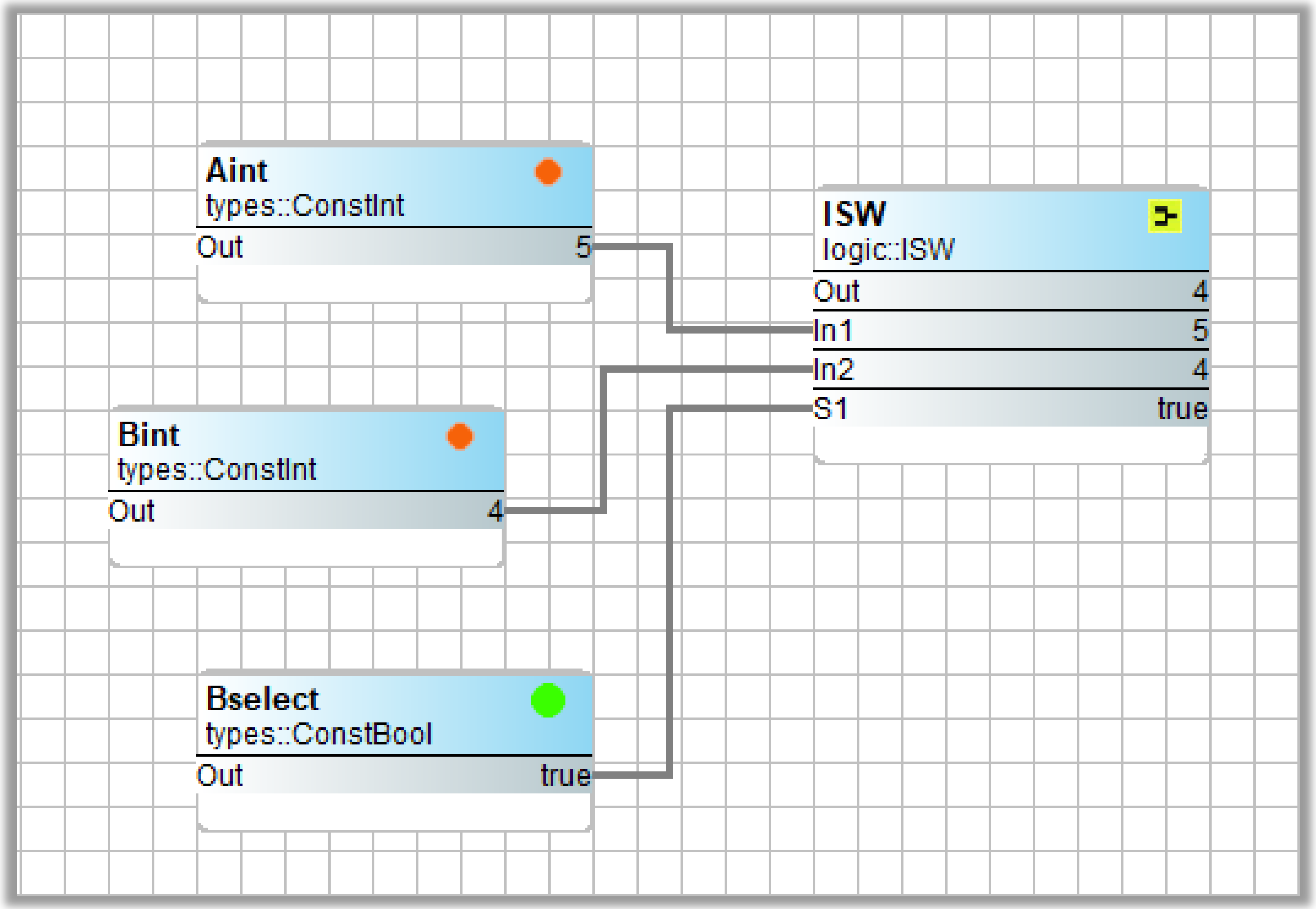


Similar in operation to the binary switch (BSW) is the analog switch (ASW). Instead of Boolean variables, the inputs are floats, but the selector (S1) is a Boolean variable. With S1 set to true, the ASW output selects input 2 (In2), otherwise Input 1 (In1) is selected. Therefore, the ASW selects one 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).

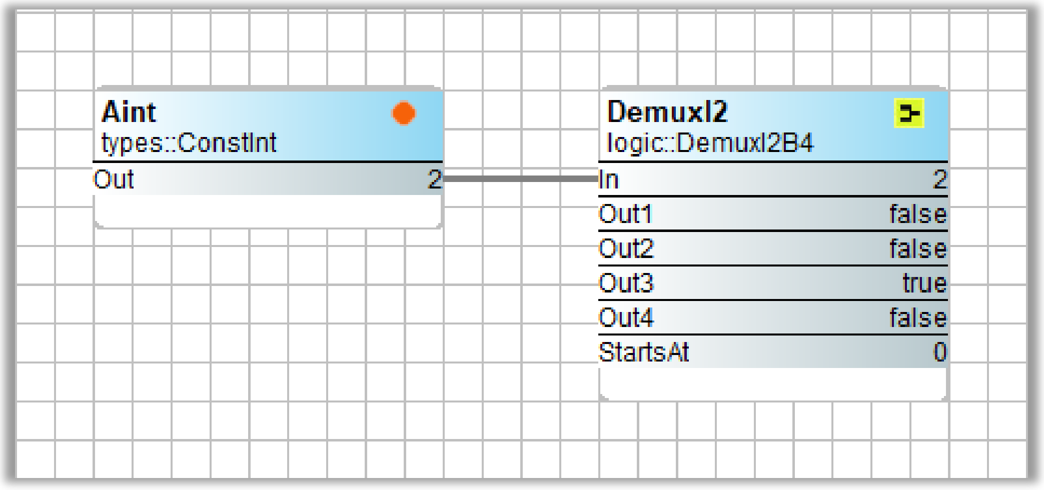
In this example, the StartsAt slot has a value of 0, meaning that input 1 (In1) is selected if Sel is 0. Since Sel is 2, the third input (In3) is selected.

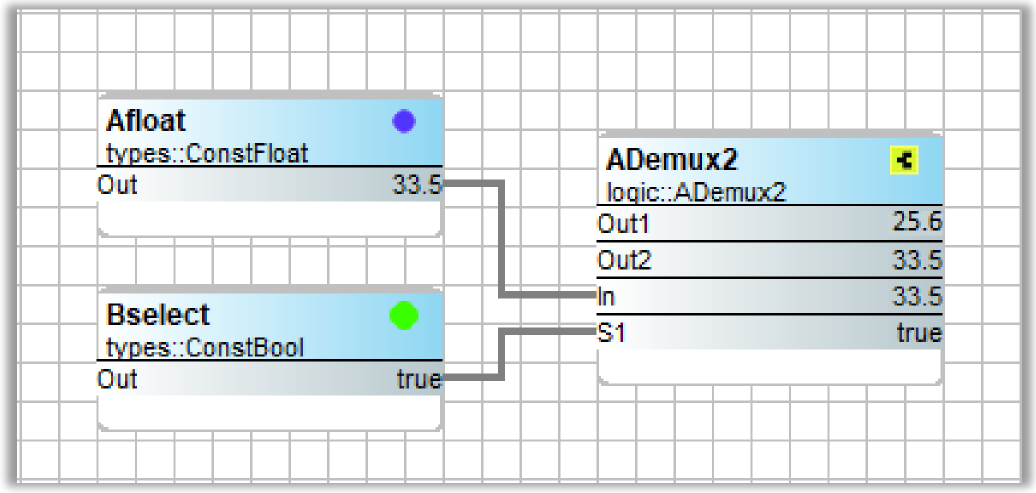
Selecting Boolean, Float or Integer (continued)



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.

De-Multiplexing

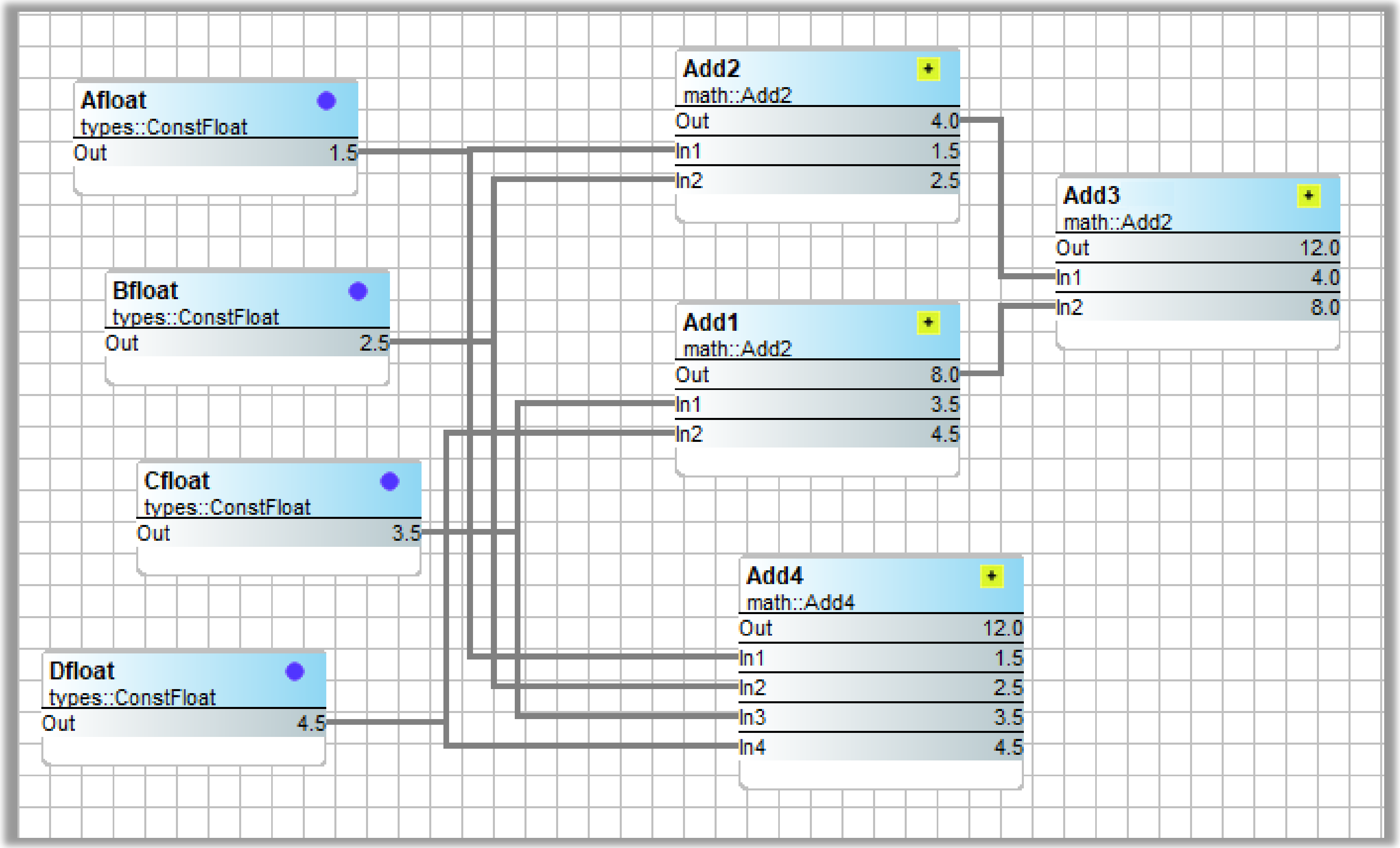




The de-multiplexer (DemuxI2) operates on integer values and provides a linear selection of the outputs based upon the value of the input. If the input is 0, the first output (Out1) is set to true. If the input is 2, the third output (Out3) is set to true.

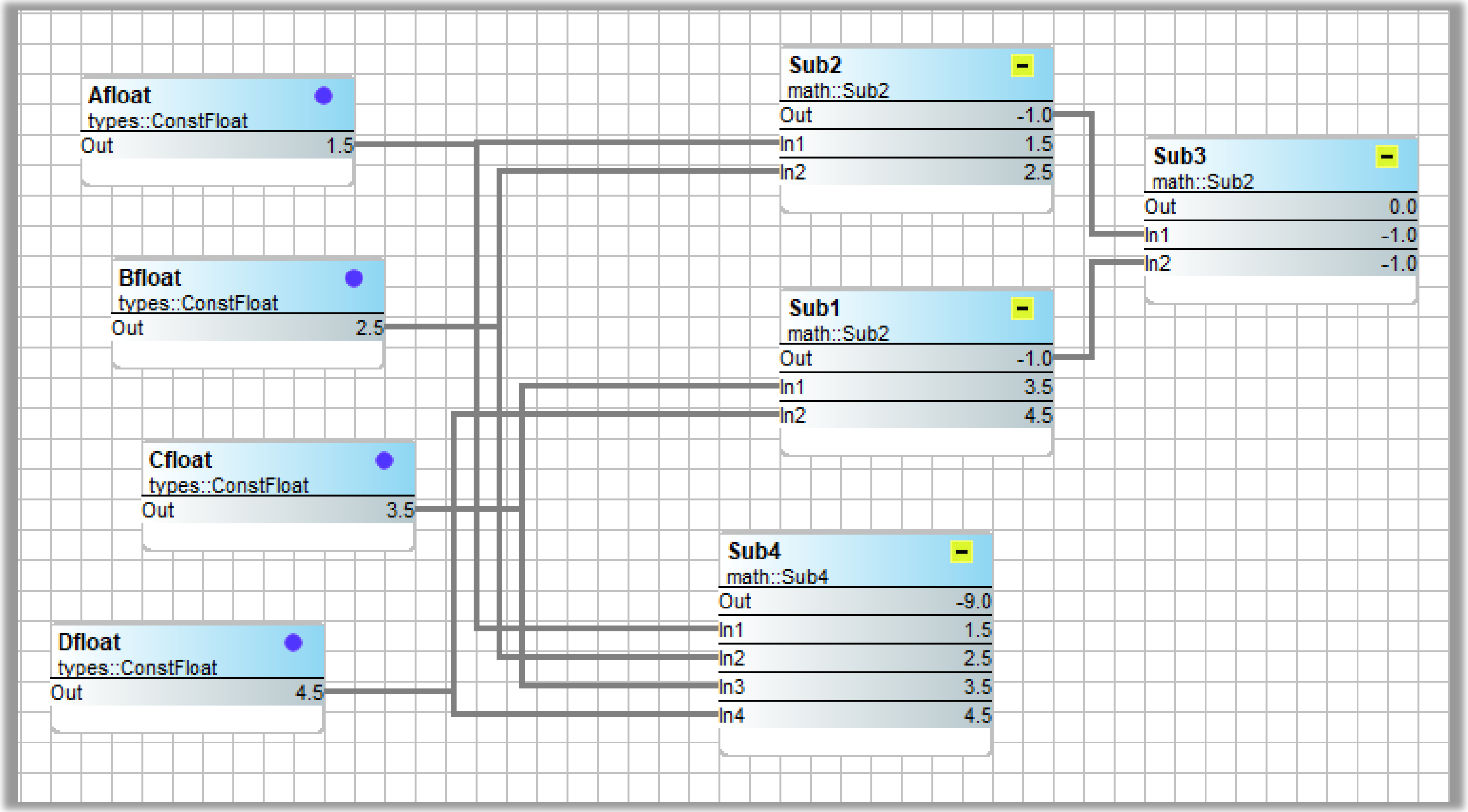
The analog de-multiplexer has only one input (In) one selector (S1), and two outputs. When the S1 is false, Out1 reflects the input. When S1 is true, Out1 will reflect the input just before the selector changed state, and Out2 will reflect the instantaneous value of the input. This component can be treated as a sample-and-hold detector.

Creating Float Addition



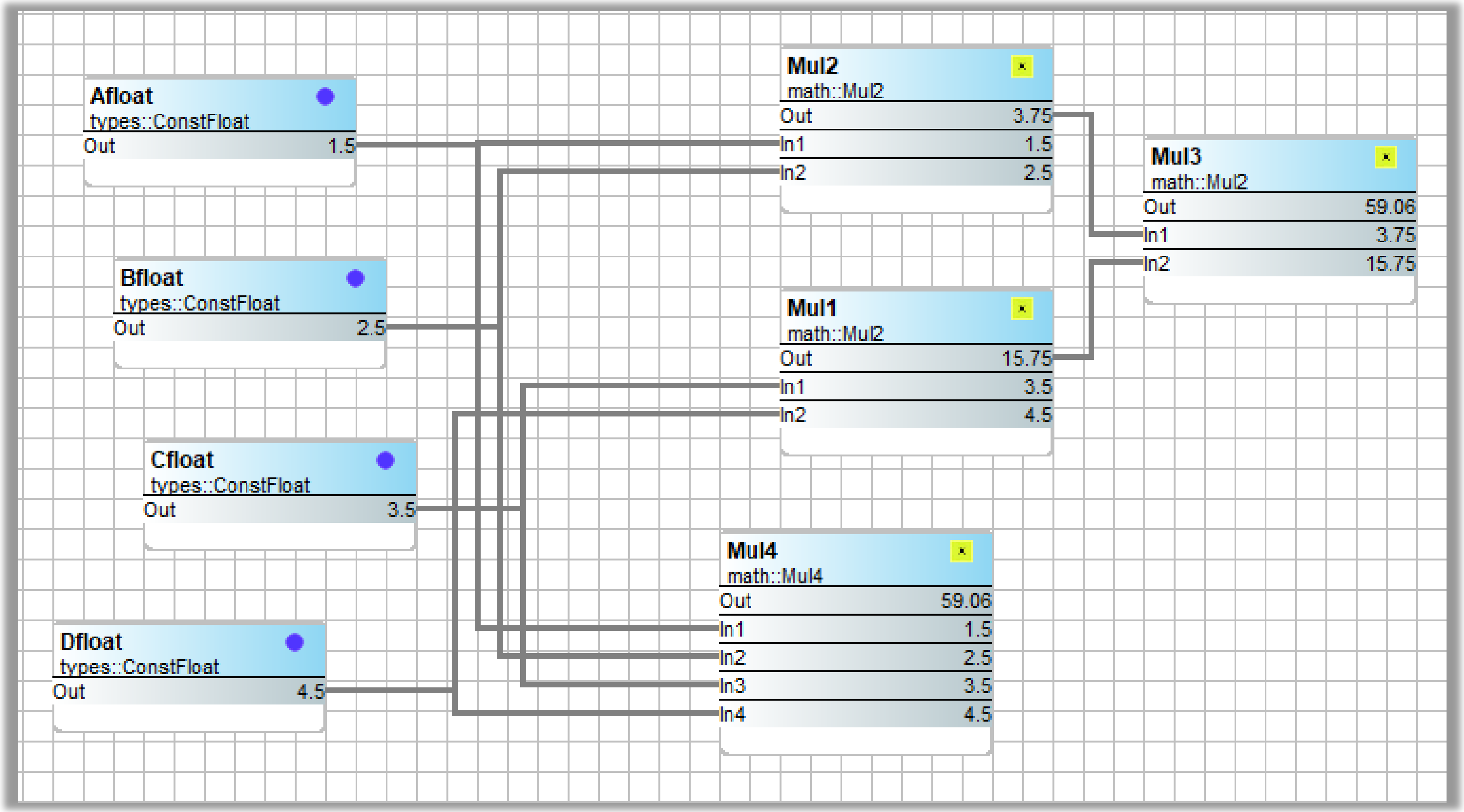
The addition (Add) components are straight forward. You can cascade two-input Add components, or you can use a four-input Add component. All inputs and outputs are floats.

Creating Float Subtraction



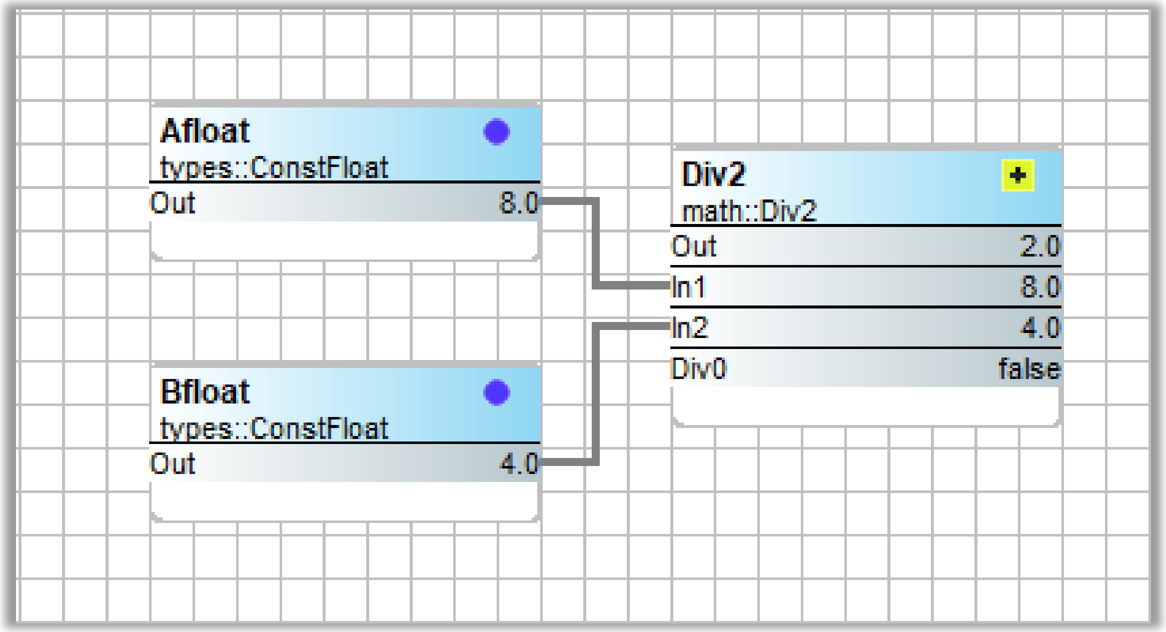
The subtract (Sub) components are also easy to work with but notice that cascading components do not yield the same results. The first input (In1) is the minuend, and all other inputs (In2, In3, In4) are subtrahends leading to outputs which represent the difference.

Creating Float Multiplication



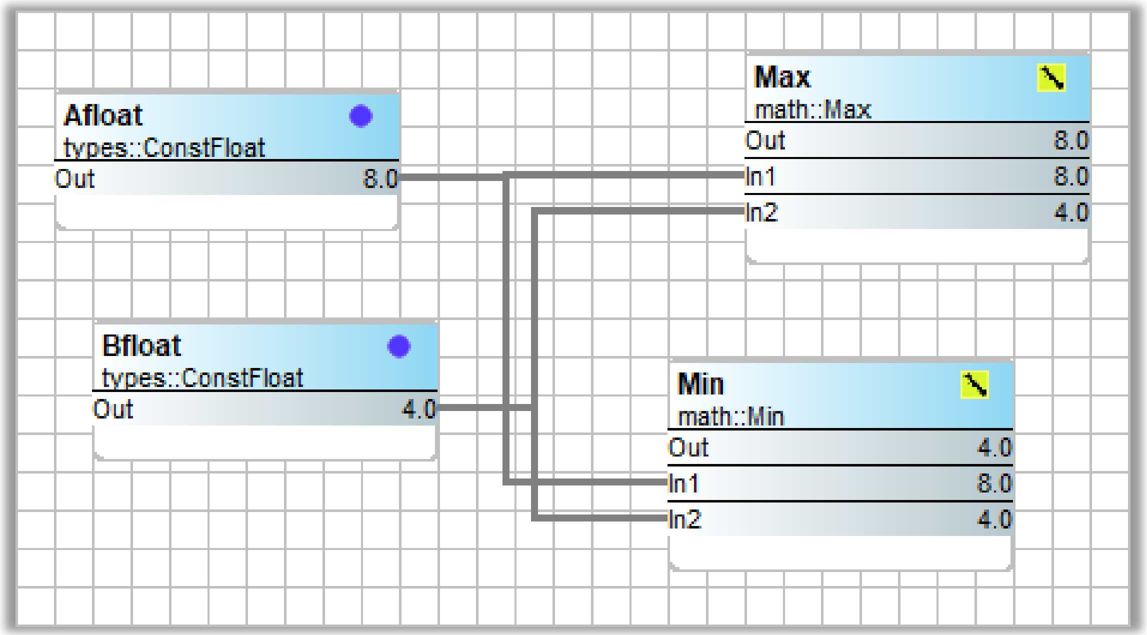
Similar to addition, float variables can be multiplied either by cascading multiply (Mul) components or by using a single larger multiplier component. All inputs and outputs are floats.

Creating Float Division



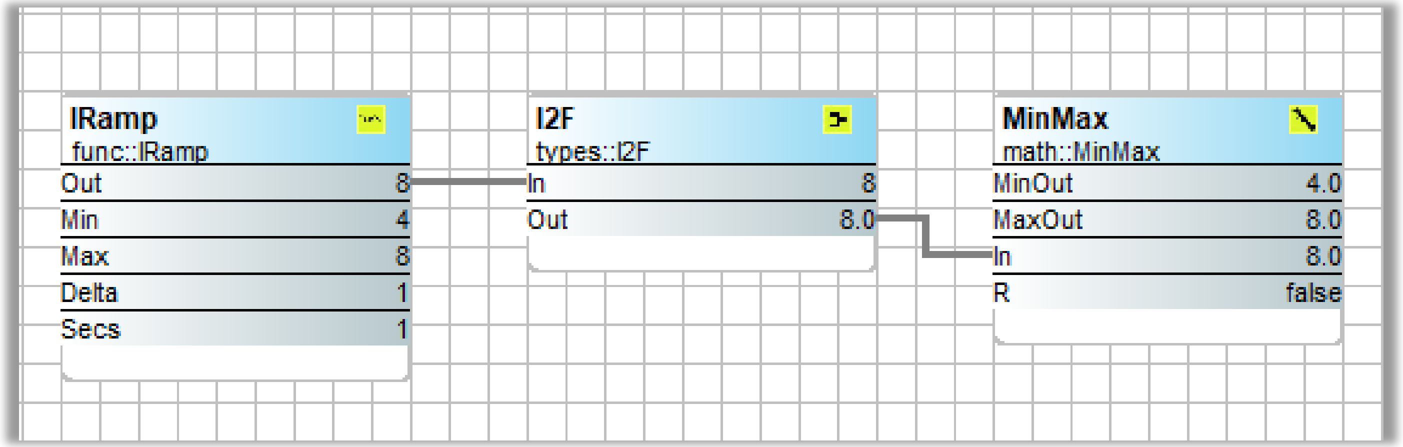
Division is also straight forward. Input 1 (In1) is the dividend, input 2 (In2) is the divisor, and the output (Out) is the quotient. Dividing by zero will result in the pin Div0 being set to true.

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.

Finding Minimums and Maximums (continued)



IRamp
func::IRamp

Out	8
Min	4
Max	8
Delta	1
Secs	1

I2F
types::I2F

In	8
Out	8.0

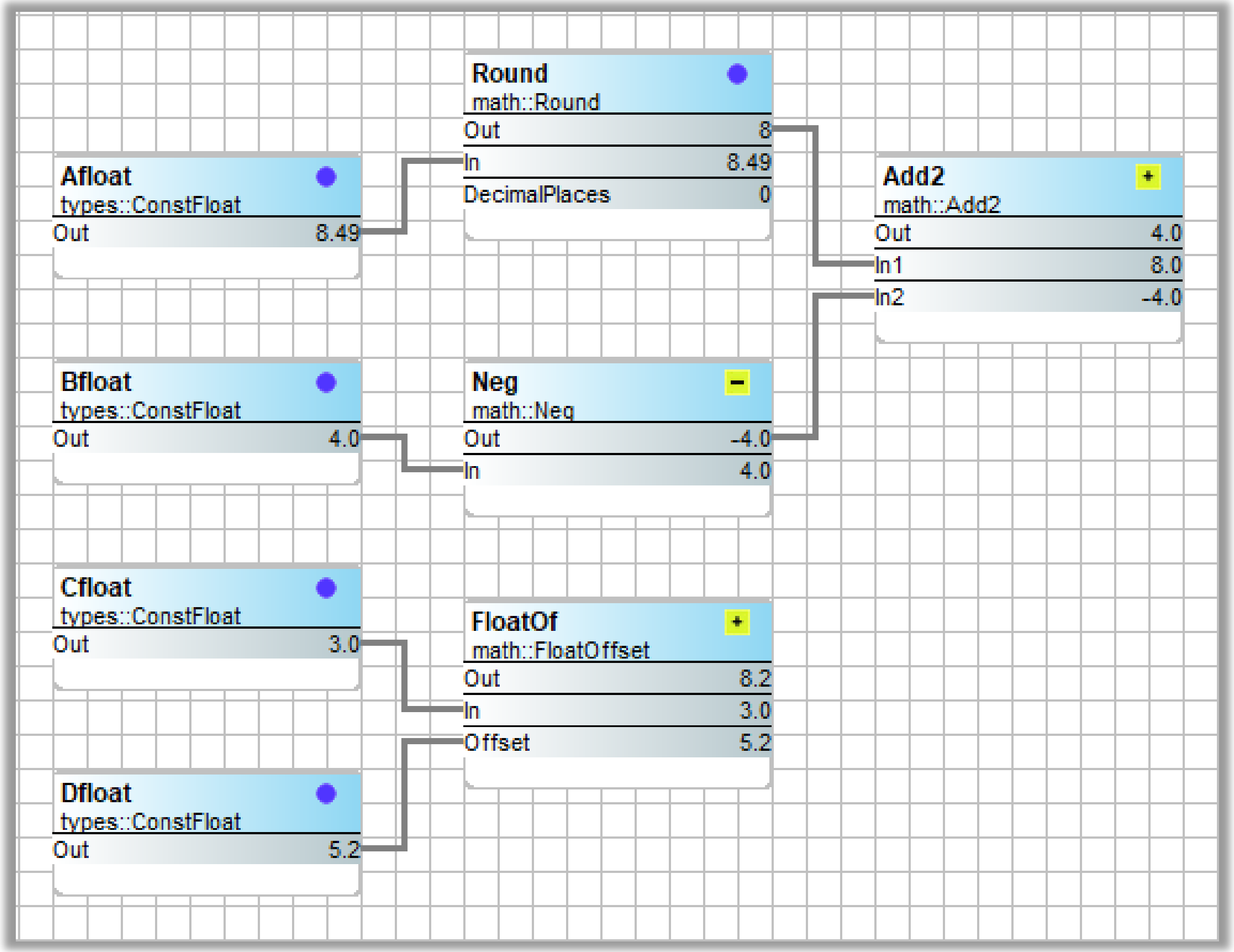
MinMax
math::MinMax

MinOut	4.0
MaxOut	8.0
In	8.0
R	false

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.

Rounding Off Floats

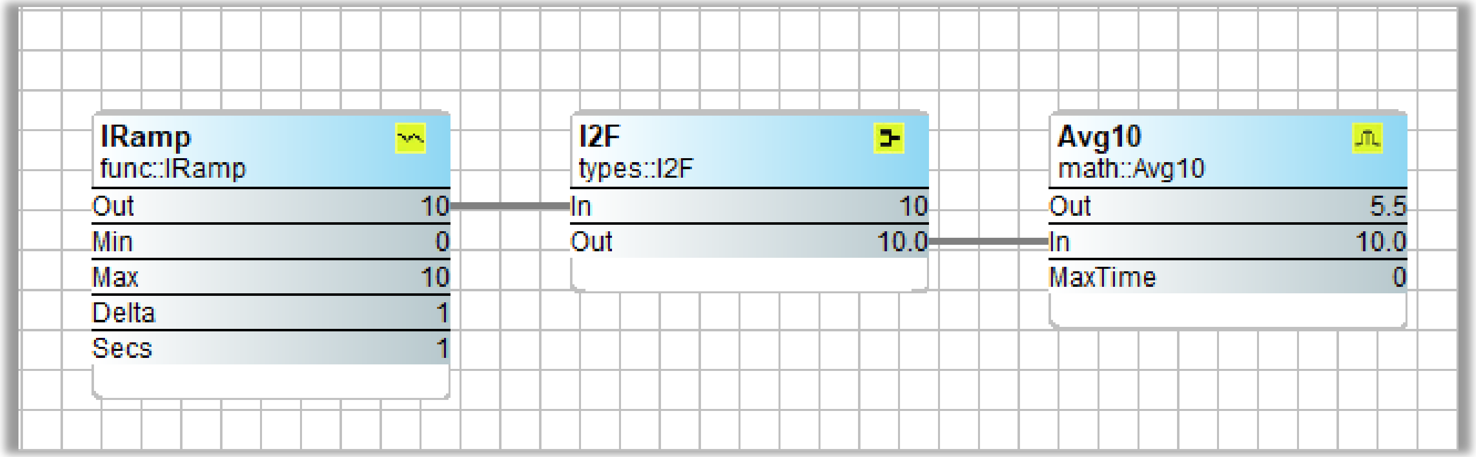


Using the Round component, you can round-off the value of a float to the closest integer value, but the output will remain a float.

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.

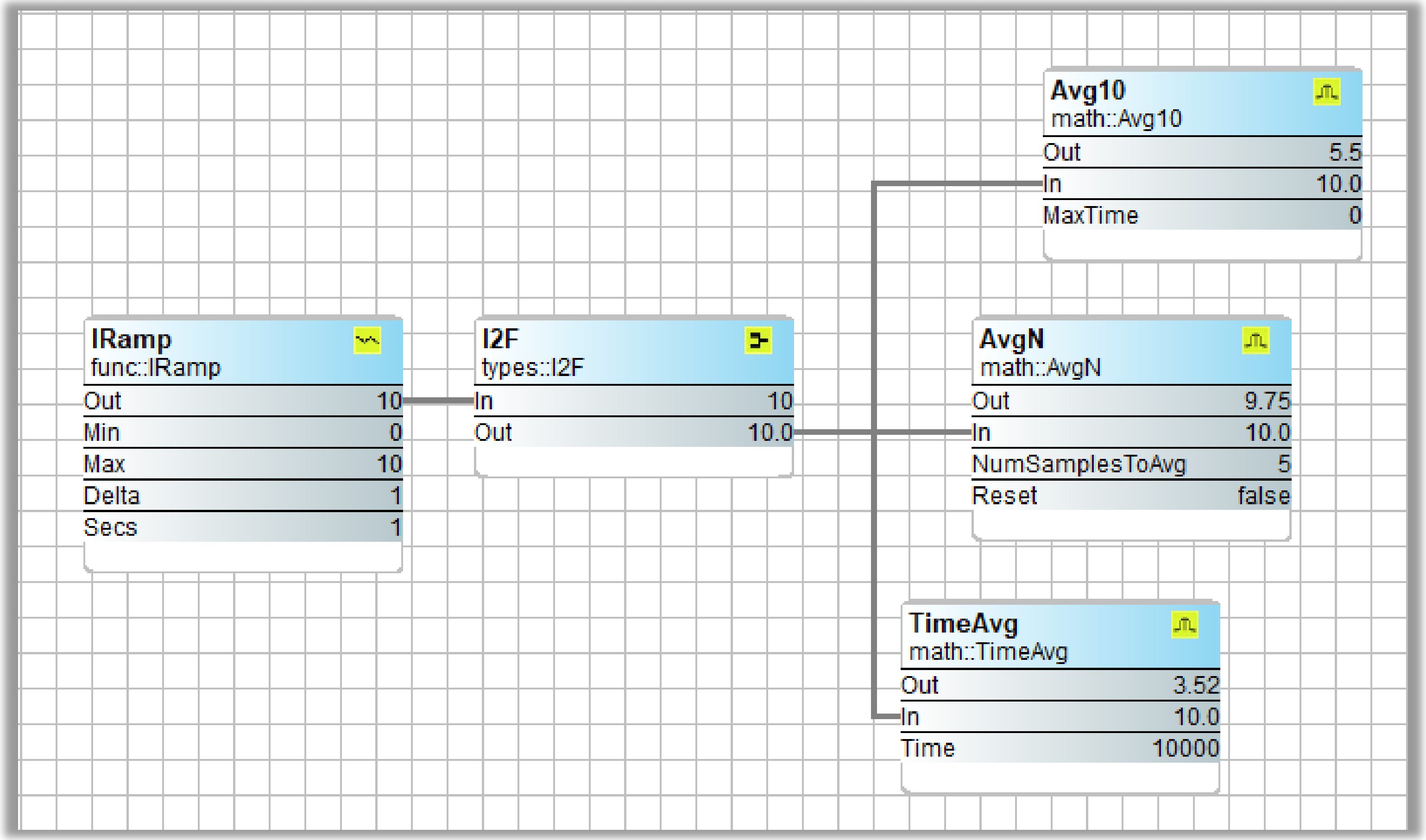
Averaging Successive Readings



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.

Averaging Successive Readings (continued)

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.



IRamp
func::IRamp

Out	10
Min	0
Max	10
Delta	1
Secs	1

I2F
types::I2F

In	10
Out	10.0

Avg10
math::Avg10

Out	5.5
In	10.0
MaxTime	0

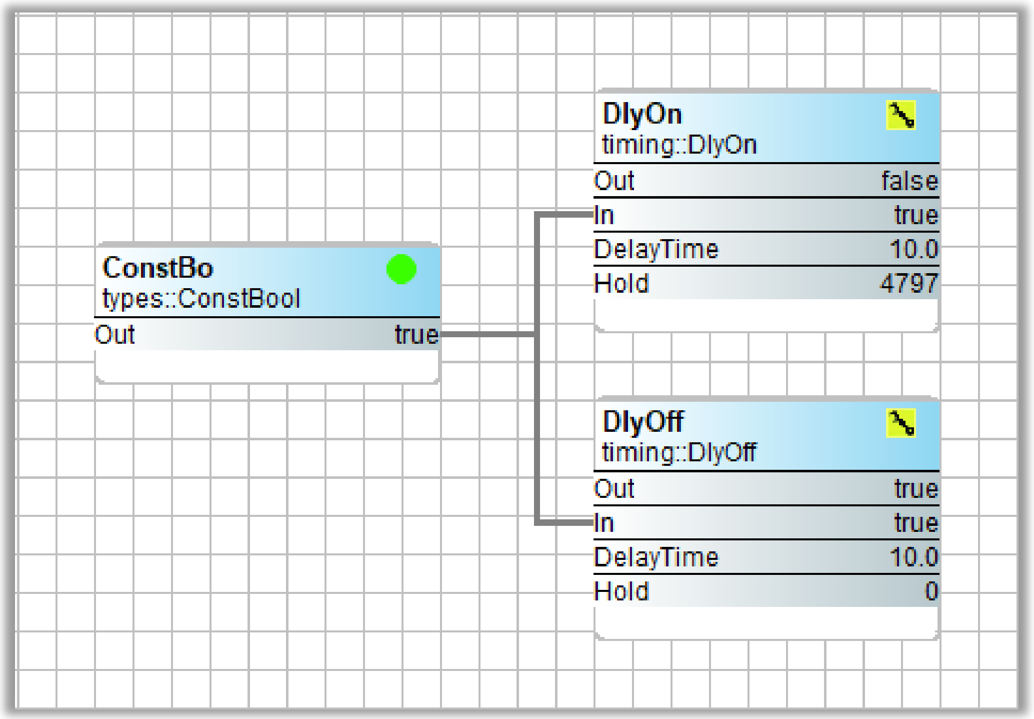
AvgN
math::AvgN

Out	9.75
In	10.0
NumSamplesToAvg	5
Reset	false

TimeAvg
math::TimeAvg

Out	3.52
In	10.0
Time	10000

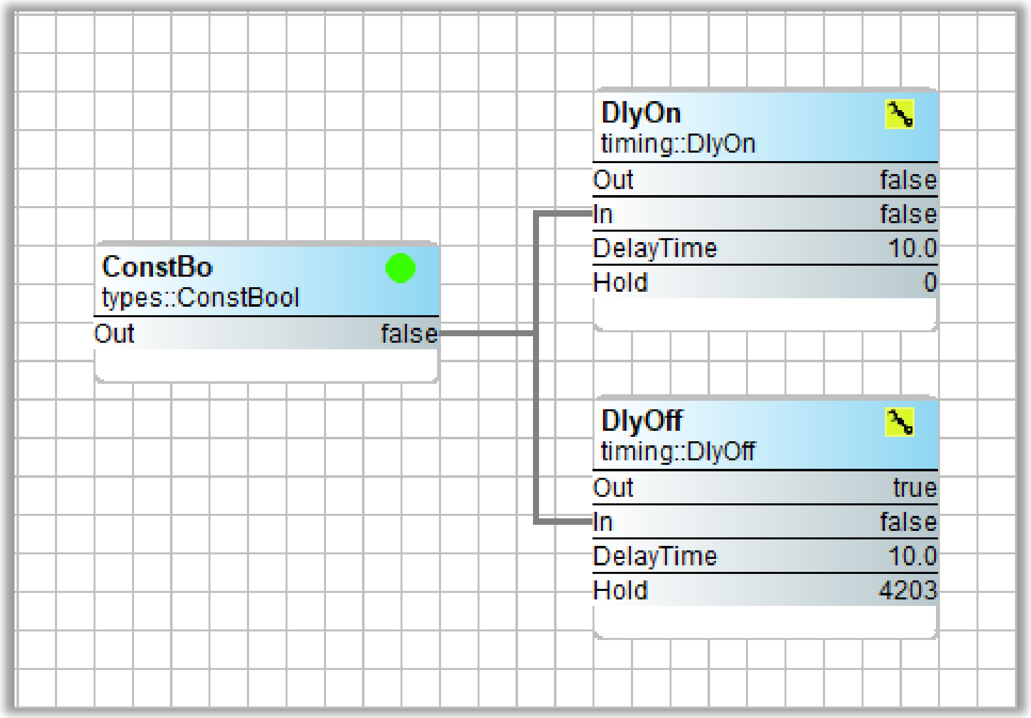
Creating On-Delays and Off-Delays



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.

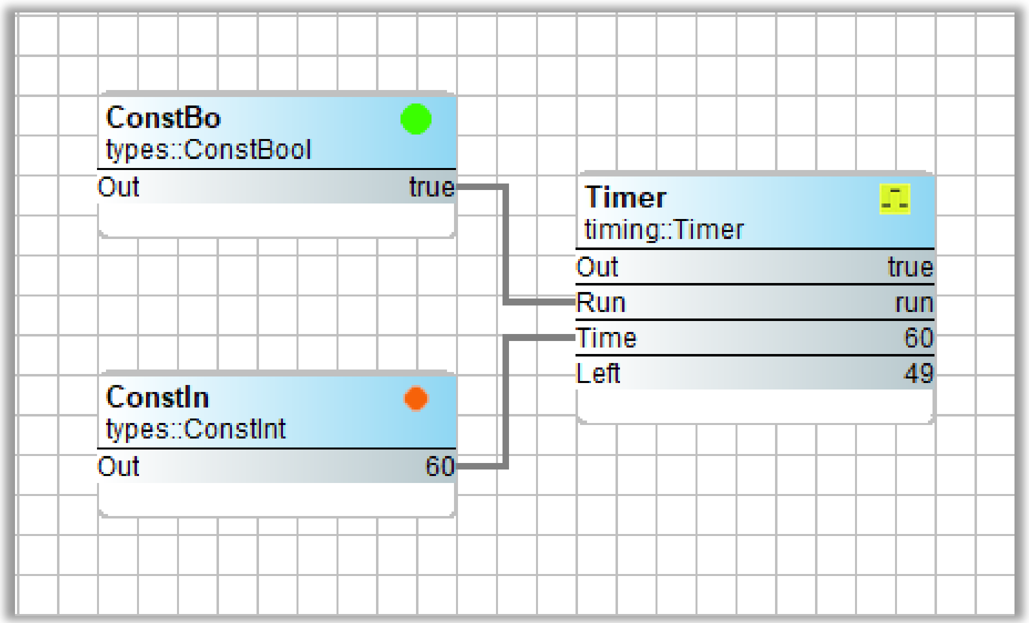
The Dlyoff component operates the same except it is triggered by a true to false transition of the input.

Creating On-Delays and Off-Delays (continued)



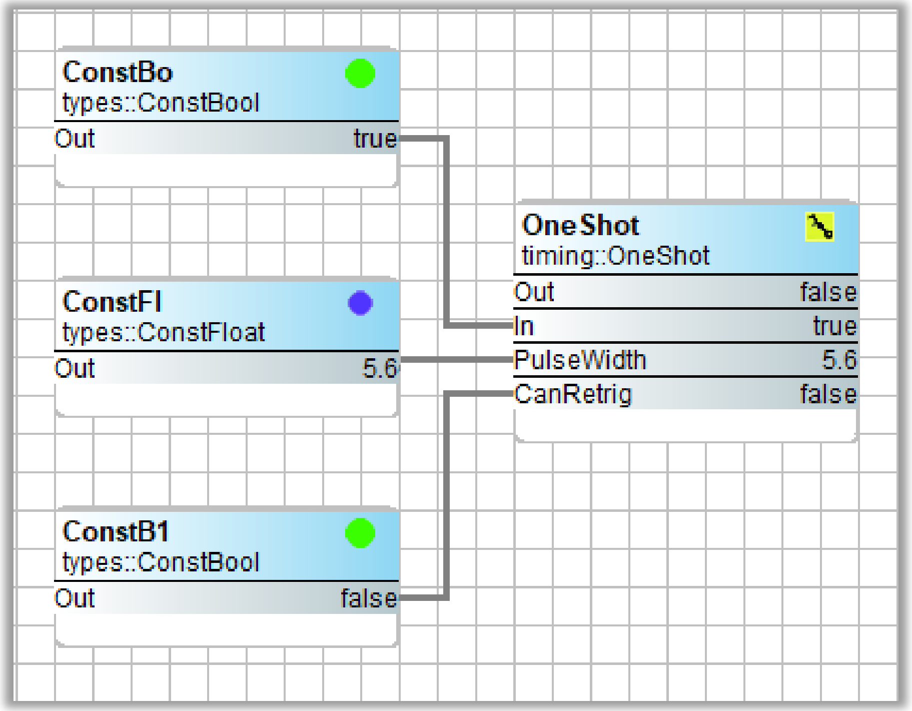
In this example, the input to the two timers made a true to false transition six seconds ago. The DlyOn components had immediately transitioned from true to false with the input, but the DlyOff timer is still timing. In another four seconds its output will become false.

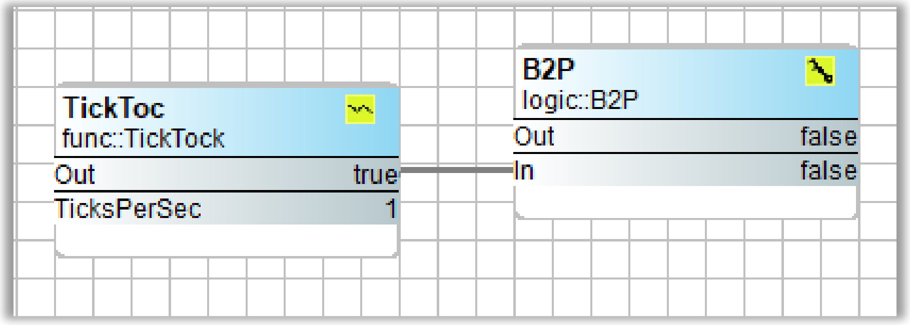
Using the Timer



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.

Using One-Shots — Mono-Stable Multivibrators

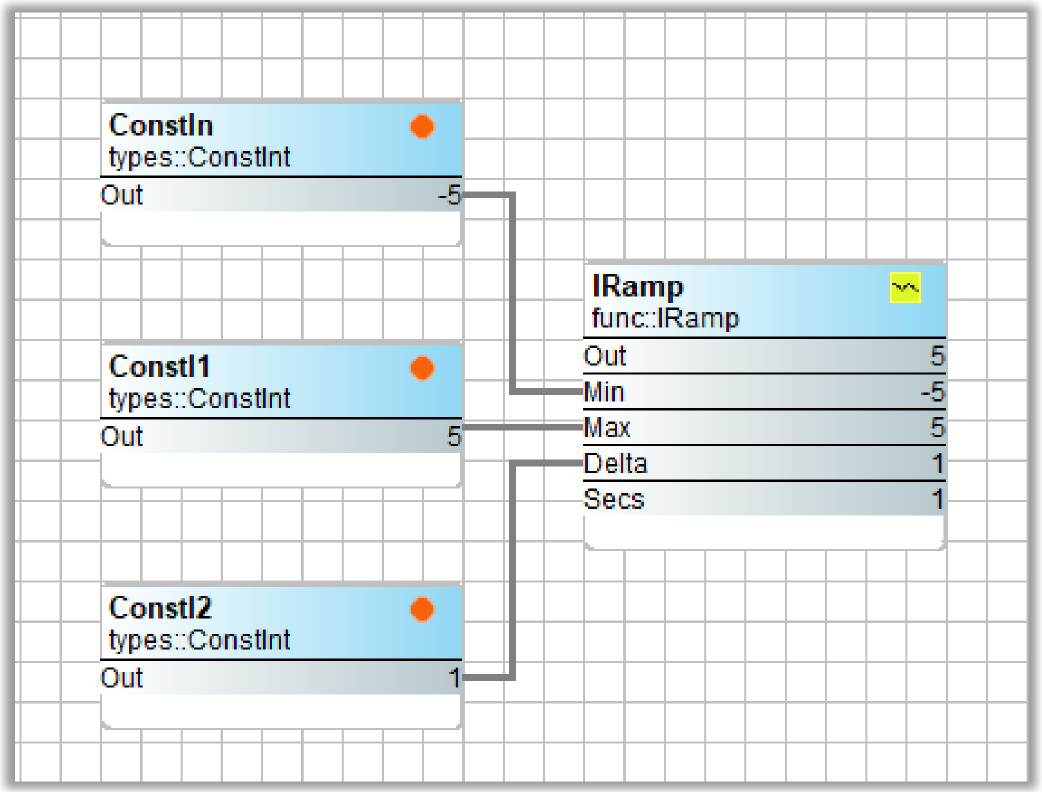




The OneShot provides a single pulse of determined value upon the false to true transition of the input signal. The output immediately goes true on its input's false to true transition, and then returns to false when timing is complete. The PulseWidth can be configured or externally programmed as shown. 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.

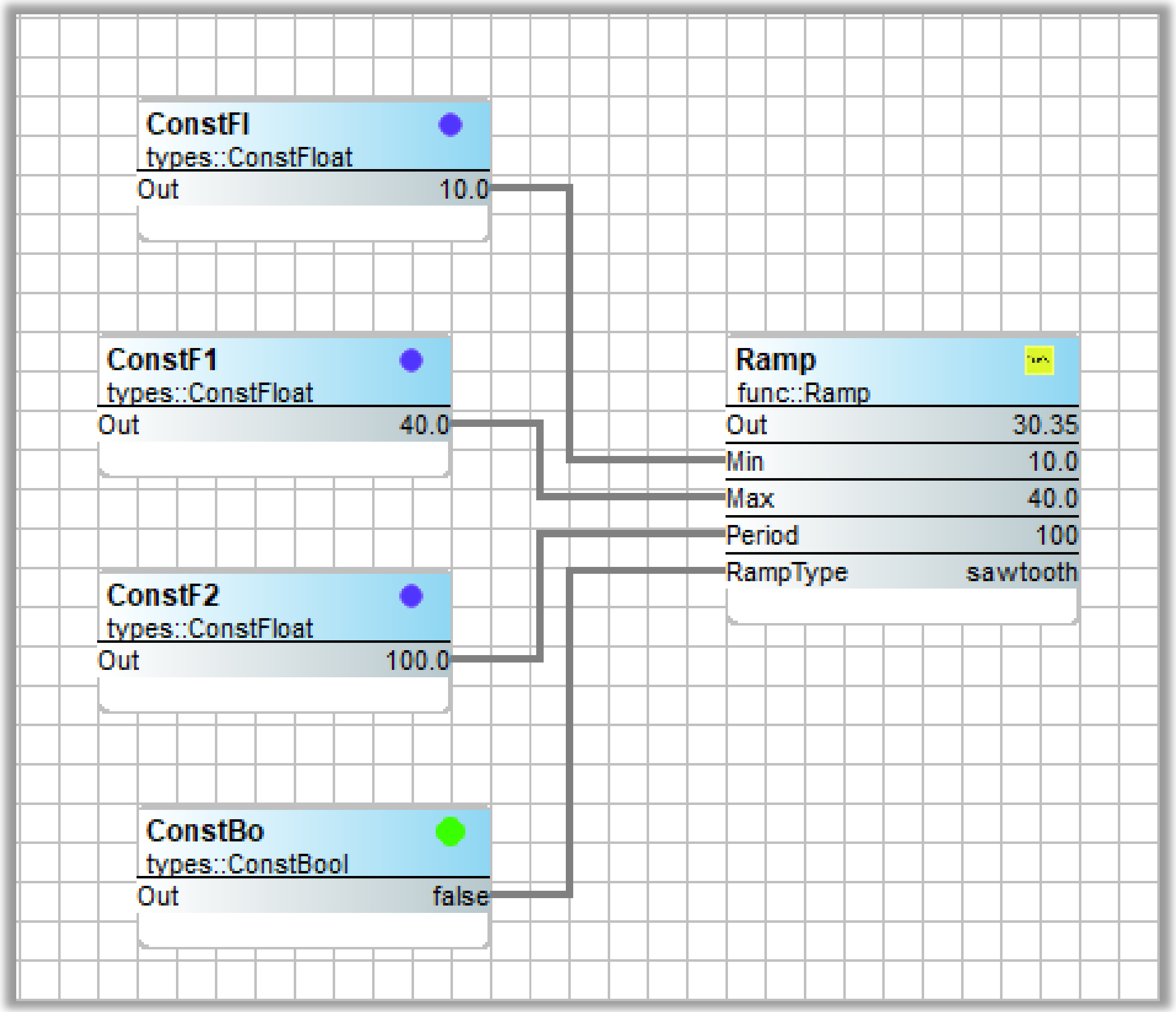
The Boolean-to-Pulse (B2P) converter is actually a very simple single-shot in that it outputs a true for only one scan time when its input goes from false to true. There are no time settings. It is used when a pulse is required after detection of an event instead of a logic level.

Creating Ramps — A-Stable Multivibrators



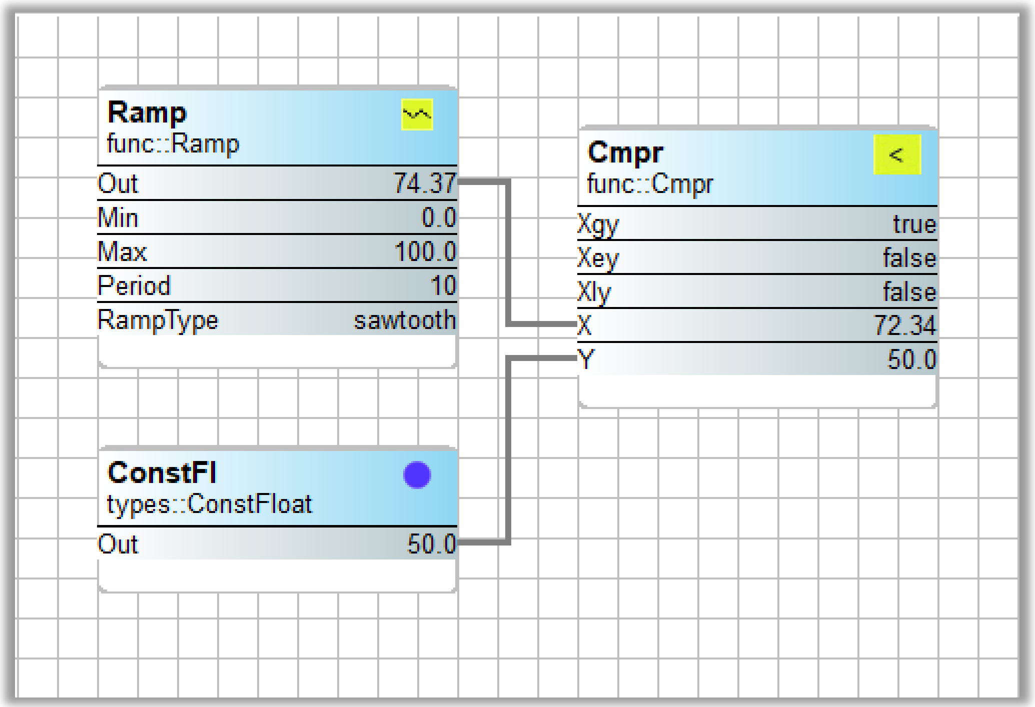
The IRamp provides a triangular output ranging from Min to Max with increments of Delta. These paraments can be configured or programmed as shown. The time increment must be configured having the units of seconds. Notice that all the configurable settings are integers, as is the output.

Creating Ramps — A-Stable Multivibrators (continued)



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.

Comparing Two Floats



Ramp
func::Ramp

Out	74.37
Min	0.0
Max	100.0
Period	10
RampType	sawtooth

ConstFl
types::ConstFloat

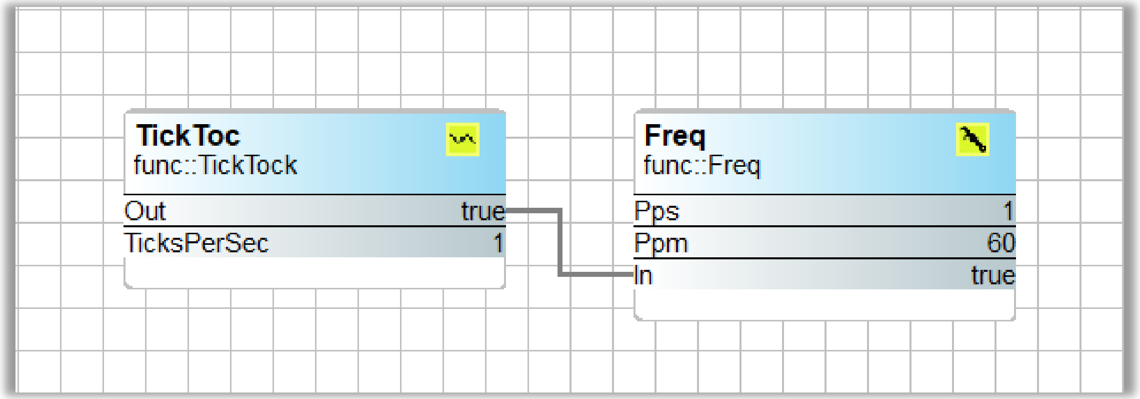
Out	50.0

Cmpr
func::Cmpr

Xgy	true
Xey	false
Xly	false
X	72.34
Y	50.0

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.

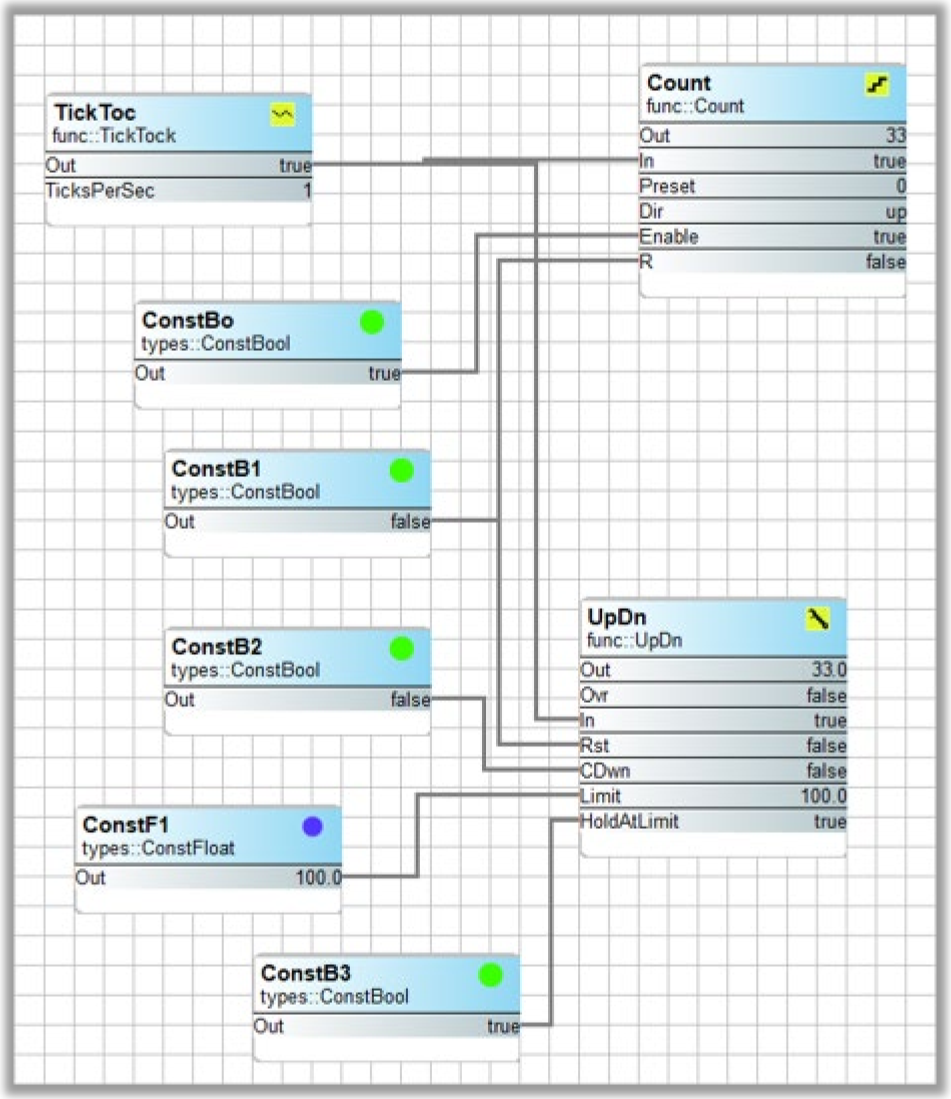
Creating a Simple Clock — the Tick Toc



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.

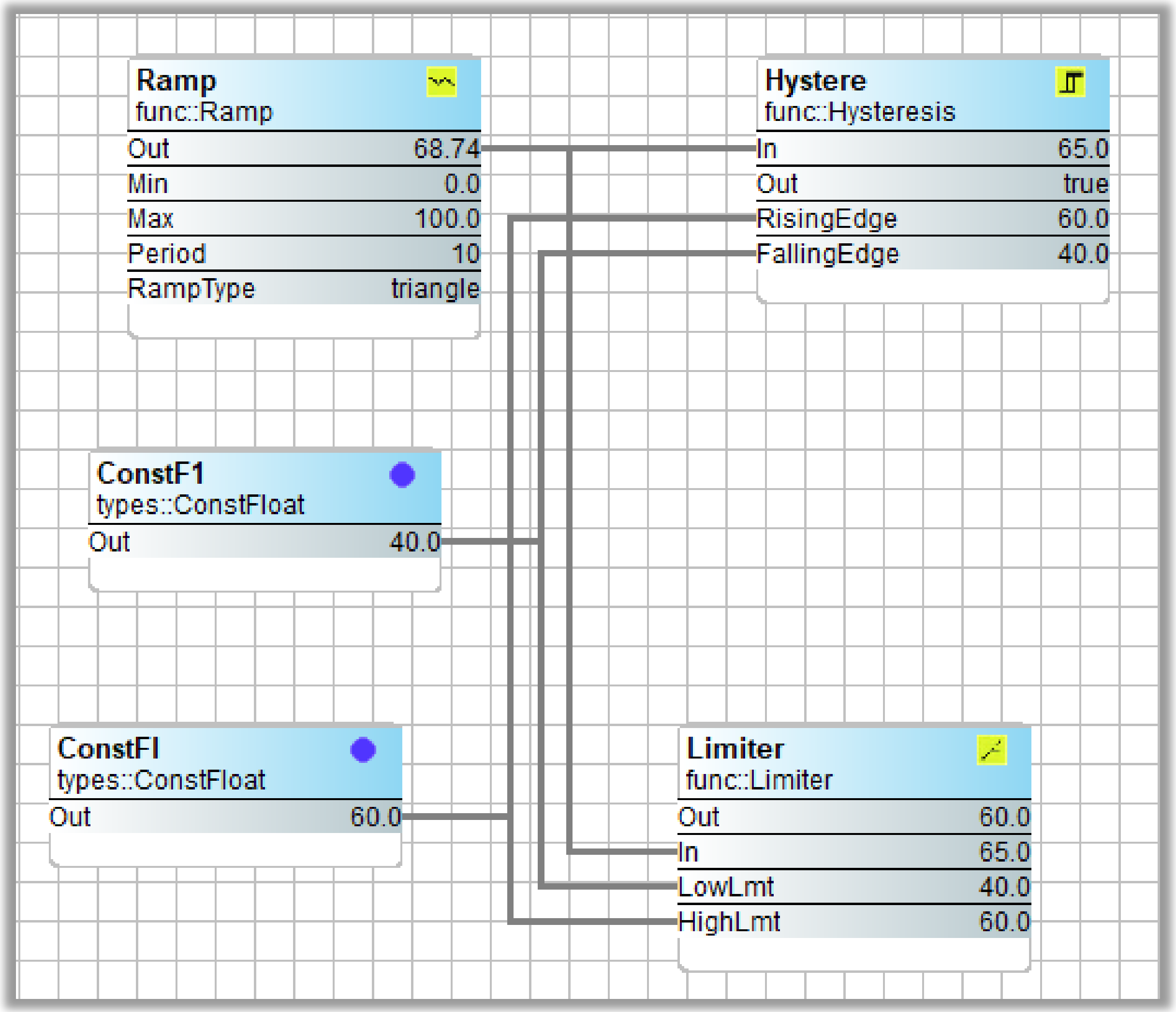
Introducing Counters



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.

Operating on Real-World Signals — Hysteresis and Limiting



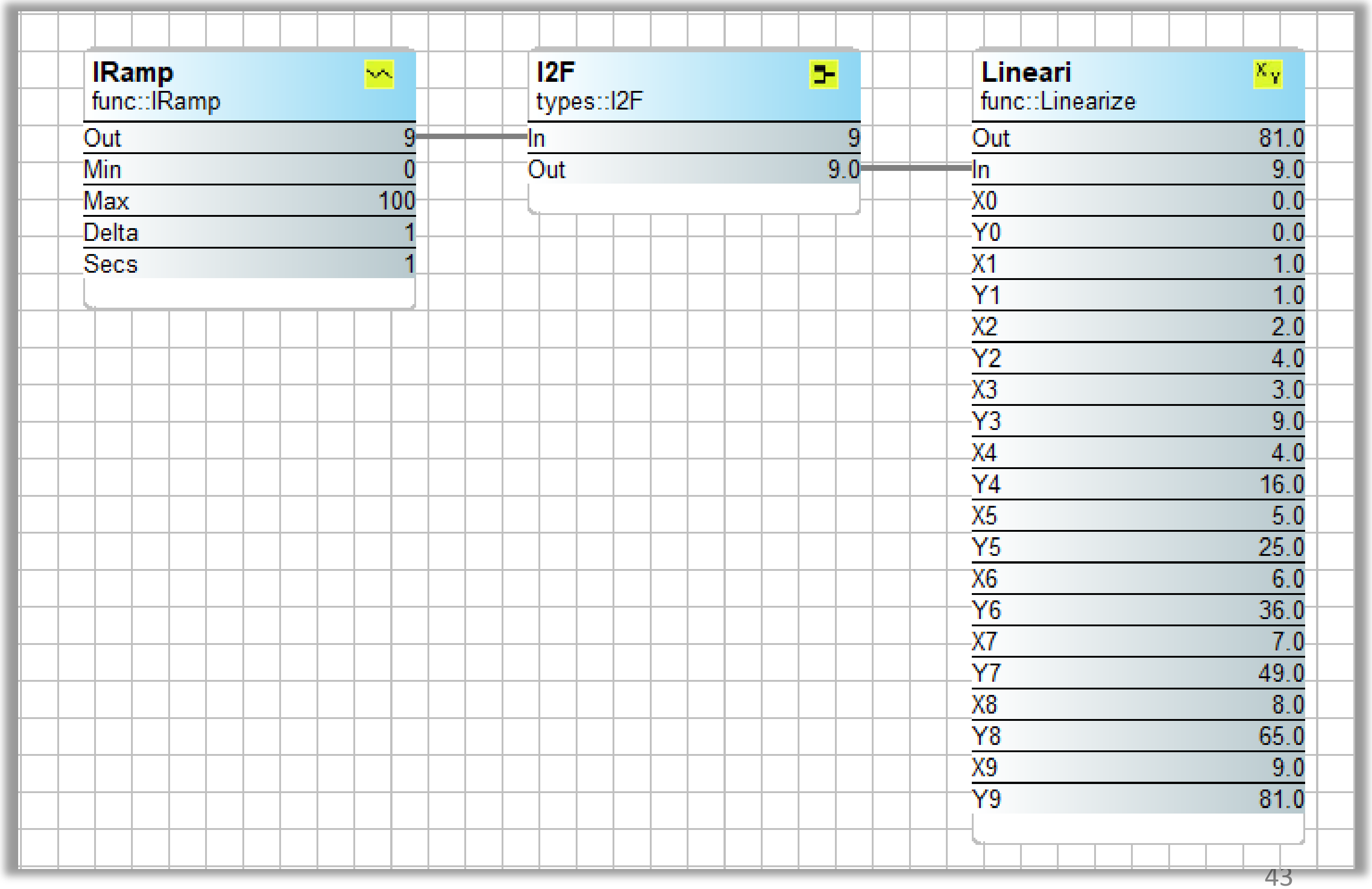
The hysteresis component (Hystere) has separate rising-edge and falling-edge trip points when setting a trigger on a float variable. It is ideal for creating a digital event from a real-world analog input. Its output is Boolean.

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.

Handling Non-Linear Signals

The 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.



IRamp
func::IRamp

Out	9
Min	0
Max	100
Delta	1
Secs	1

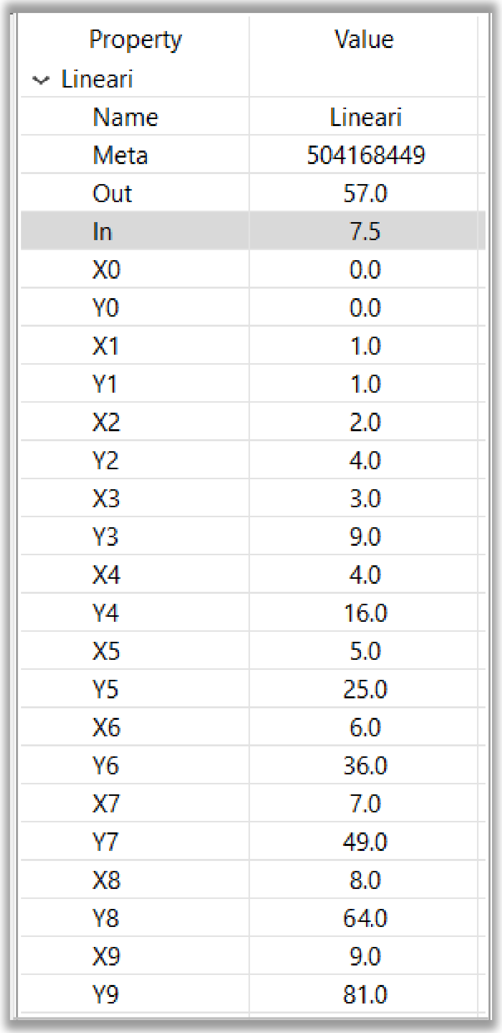
I2F
types::I2F

In	9
Out	9.0

Lineari
func::Linearize

Out	81.0
In	9.0
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	65.0
X9	9.0
Y9	81.0

Handling Non-Linear Signals (continued)

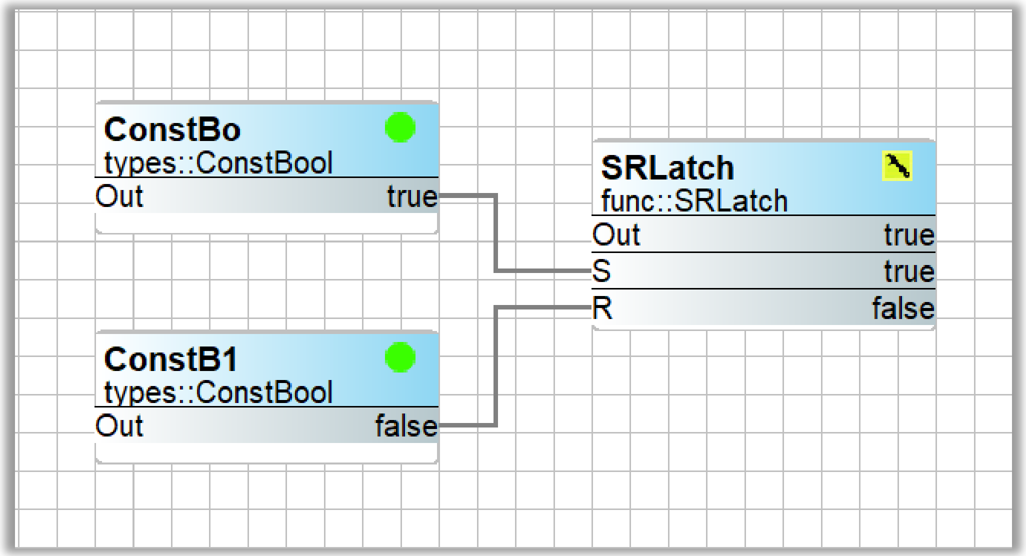


Property	Value
Lineari	
Name	Lineari
Meta	504168449
Out	57.0
In	7.5
X0	0.0
Y0	0.0
X1	1.0
Y1	1.0
X2	2.0
Y2	4.0
X3	3.0
Y3	9.0
X4	4.0
Y4	16.0
X5	5.0
Y5	25.0
X6	6.0
Y6	36.0
X7	7.0
Y7	49.0
X8	8.0
Y8	64.0
X9	9.0
Y9	81.0

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.

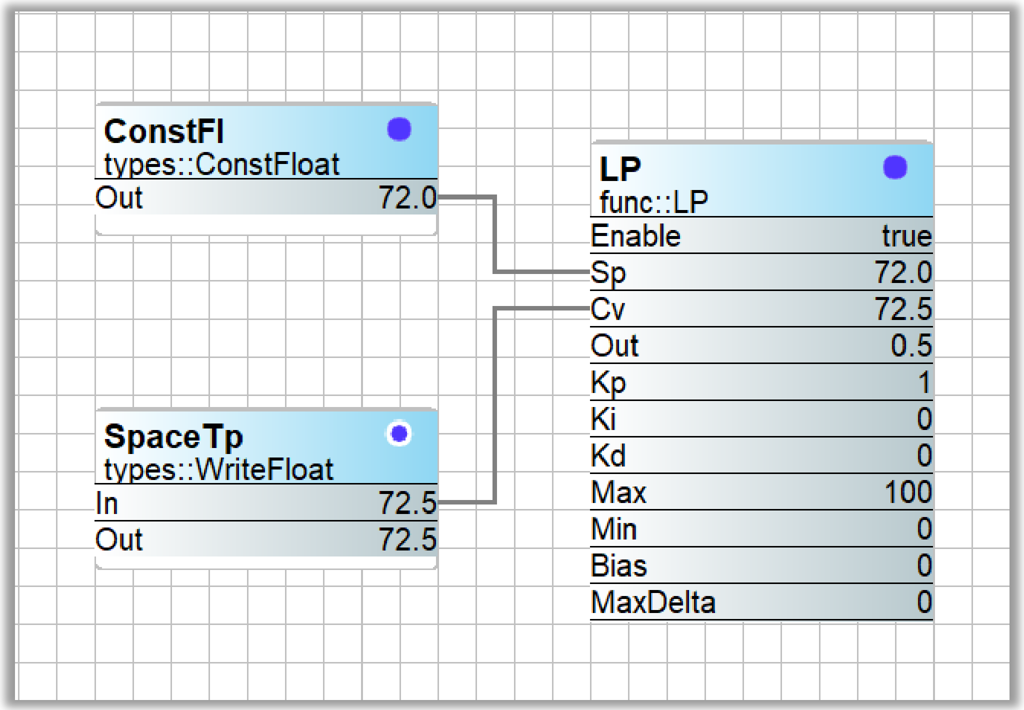
Creating a Simple Set-Reset Flip Flop — Bi-Stable Multivibrator



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.

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.

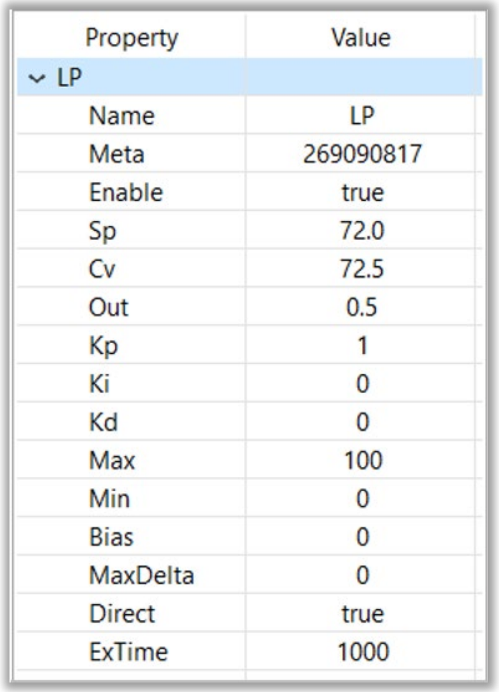
Creating the Loop Component — Basic Analog Controller



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

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.

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

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

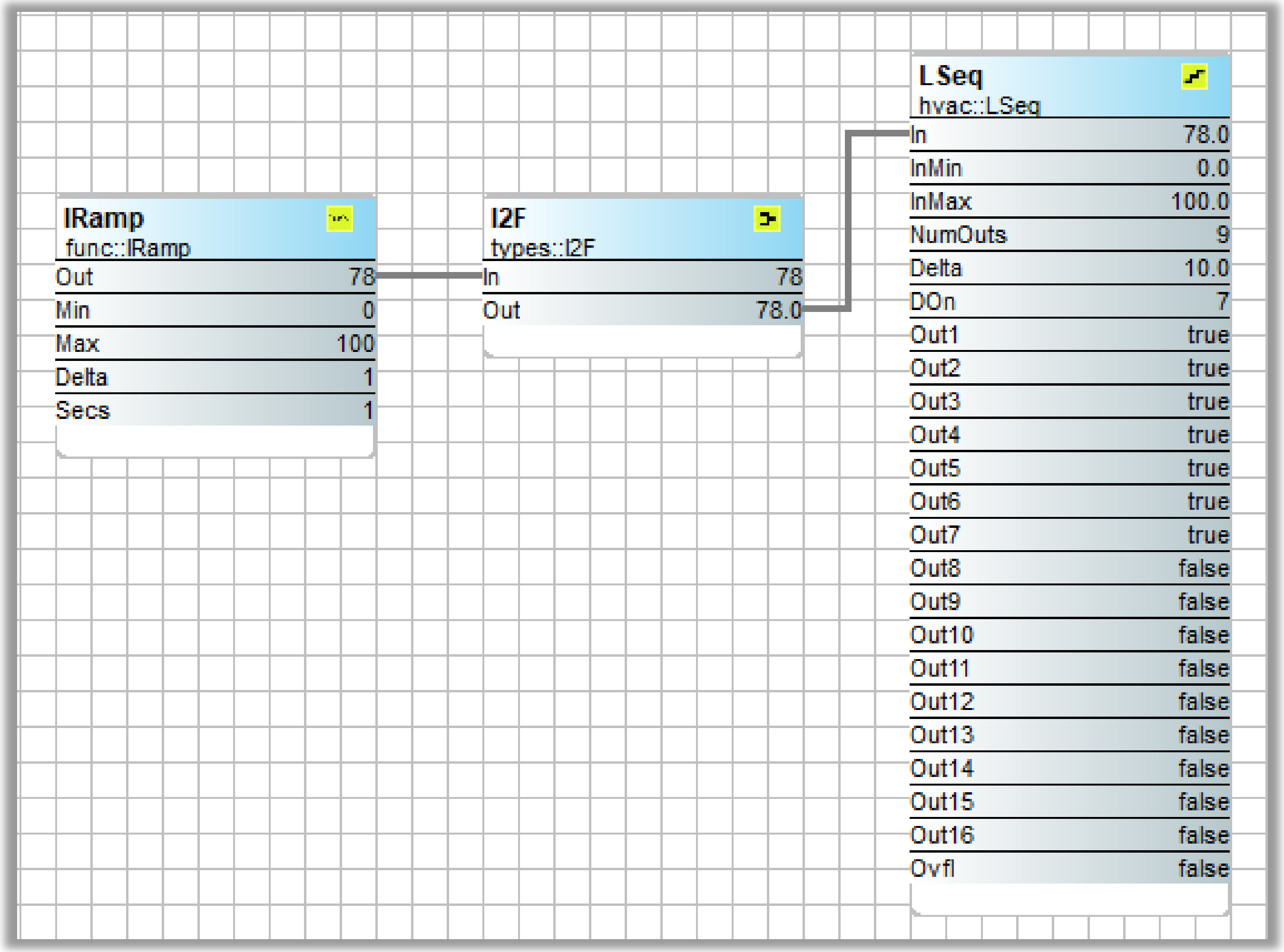
Enable must be configured true, otherwise there is no control.

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.

Creating a Linear Sequencer — Bar-Graph Representation of a Float



IRamp
func::IRamp

Out	78
Min	0
Max	100
Delta	1
Secs	1

I2F
types::I2F

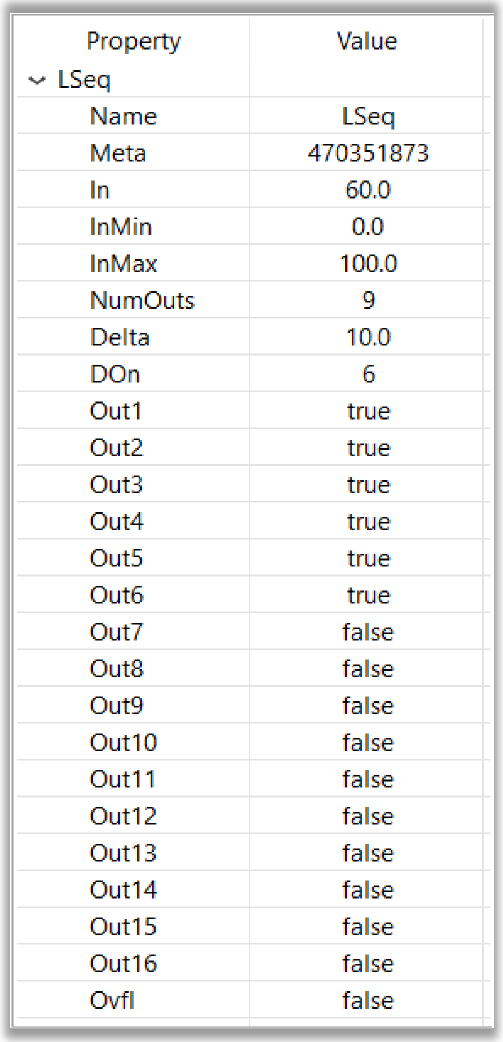
In	78
Out	78.0

LSeq
hvac::LSeq

In	78.0
InMin	0.0
InMax	100.0
NumOuts	9
Delta	10.0
DOn	7
Out1	true
Out2	true
Out3	true
Out4	true
Out5	true
Out6	true
Out7	true
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.

Creating a Linear Sequencer — Bar-Graph Representation of a Float (continued)

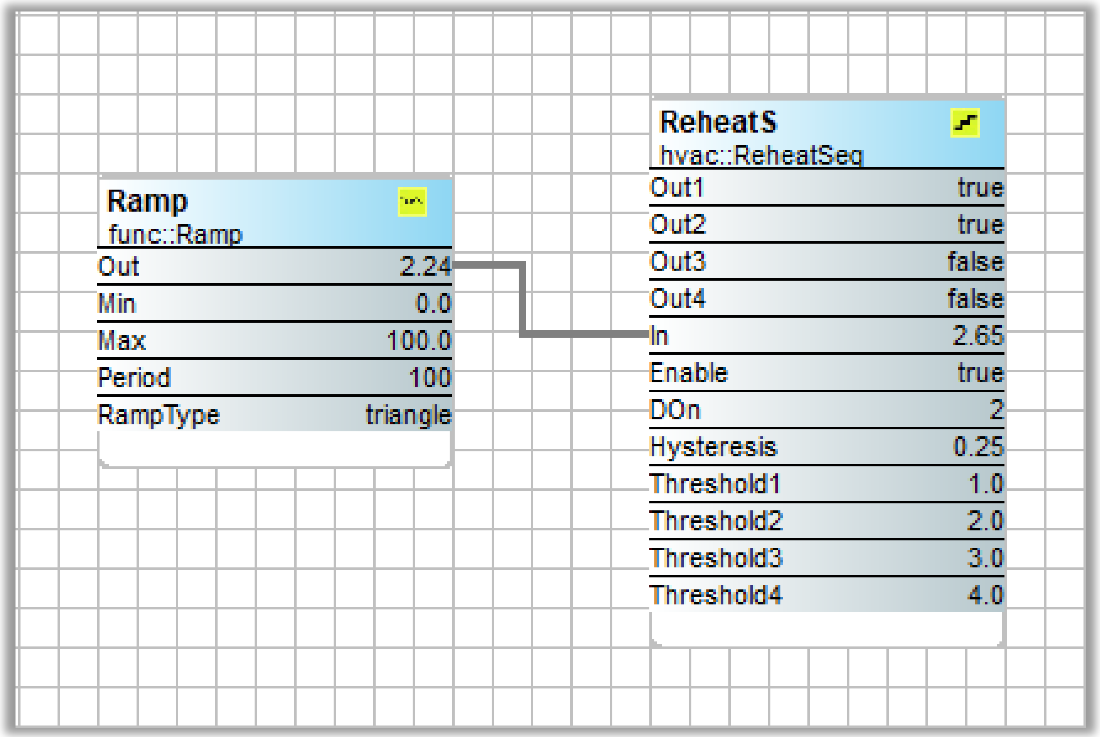


Property	Value
⌄ LSeq	
Name	LSeq
Meta	470351873
In	60.0
InMin	0.0
InMax	100.0
NumOuts	9
Delta	10.0
DOn	6
Out1	true
Out2	true
Out3	true
Out4	true
Out5	true
Out6	true
Out7	false
Out8	false
Out9	false
Out10	false
Out11	false
Out12	false
Out13	false
Out14	false
Out15	false
Out16	false
Ovfl	false

The 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.

Creating a Reheat Sequencer — Four Staged Outputs from a Float Input

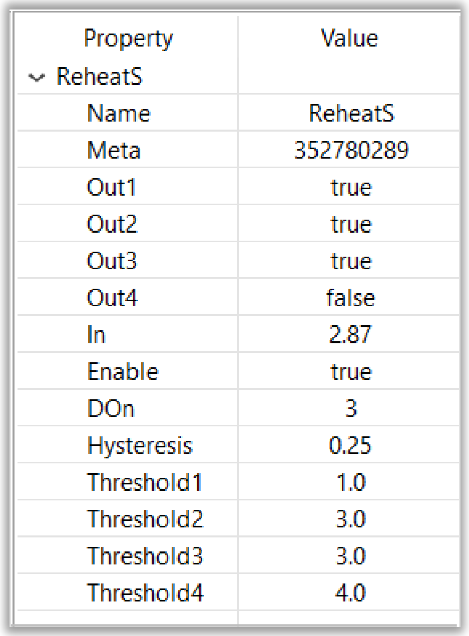


Ramp	func::Ramp
Out	2.24
Min	0.0
Max	100.0
Period	100
RampType	triangle

ReheatS	hvac::ReheatSeq
Out1	true
Out2	true
Out3	false
Out4	false
In	2.65
Enable	true
DOn	2
Hysteresis	0.25
Threshold1	1.0
Threshold2	2.0
Threshold3	3.0
Threshold4	4.0

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.

Creating a Reheat Sequencer — Four Staged Outputs from a Float Input (continued)



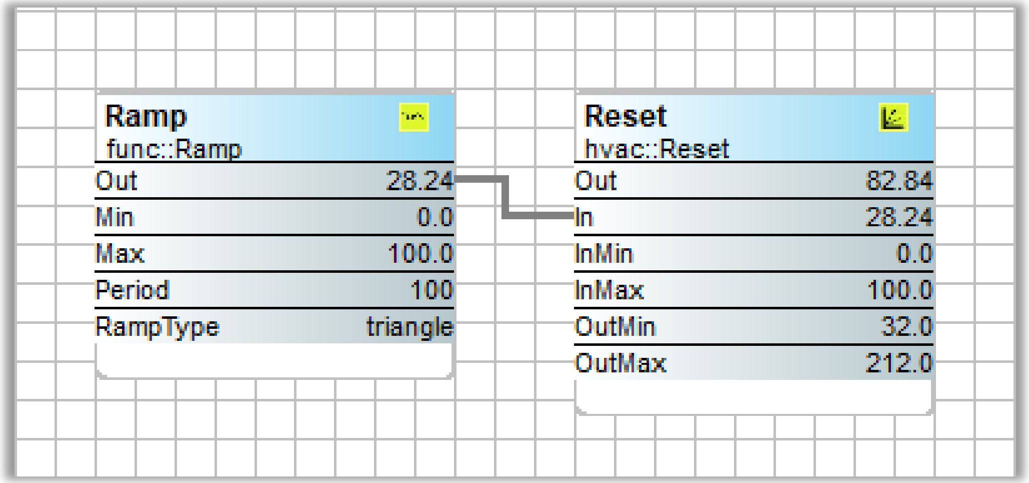
Property	Value
⌄ ReheatS	
Name	ReheatS
Meta	352780289
Out1	true
Out2	true
Out3	true
Out4	false
In	2.87
Enable	true
DOn	3
Hysteresis	0.25
Threshold1	1.0
Threshold2	3.0
Threshold3	3.0
Threshold4	4.0

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.

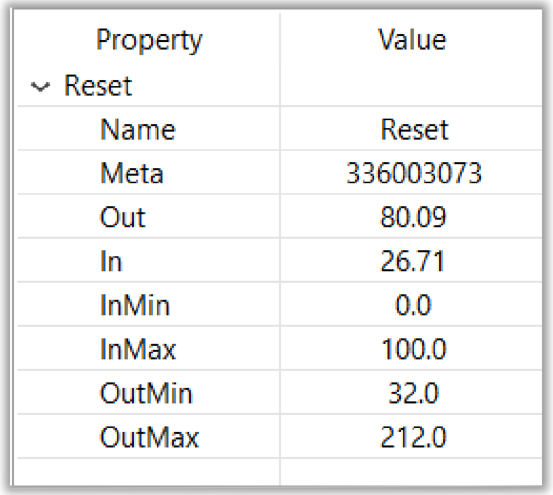
Reset — Scaling a Float Input Between Two Limits



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

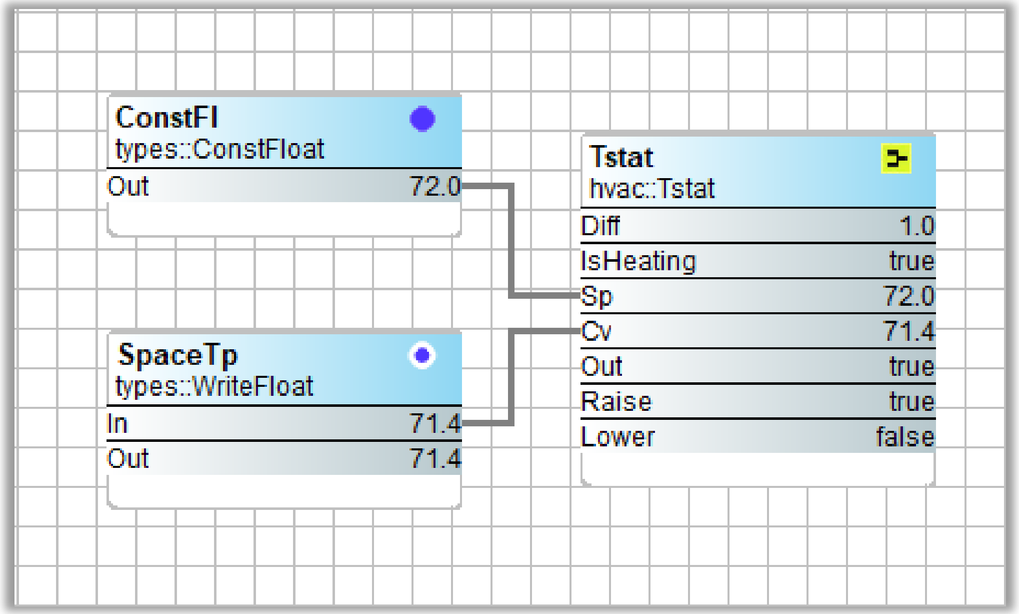
The Reset component (Reset) will scale the output linearly between two limits. The input ranges must be configured by setting InMin and InMax. The corresponding output for those two points must be configured as OutMin and OutMax. If the input signal exceeds the defined input range, the output will be 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

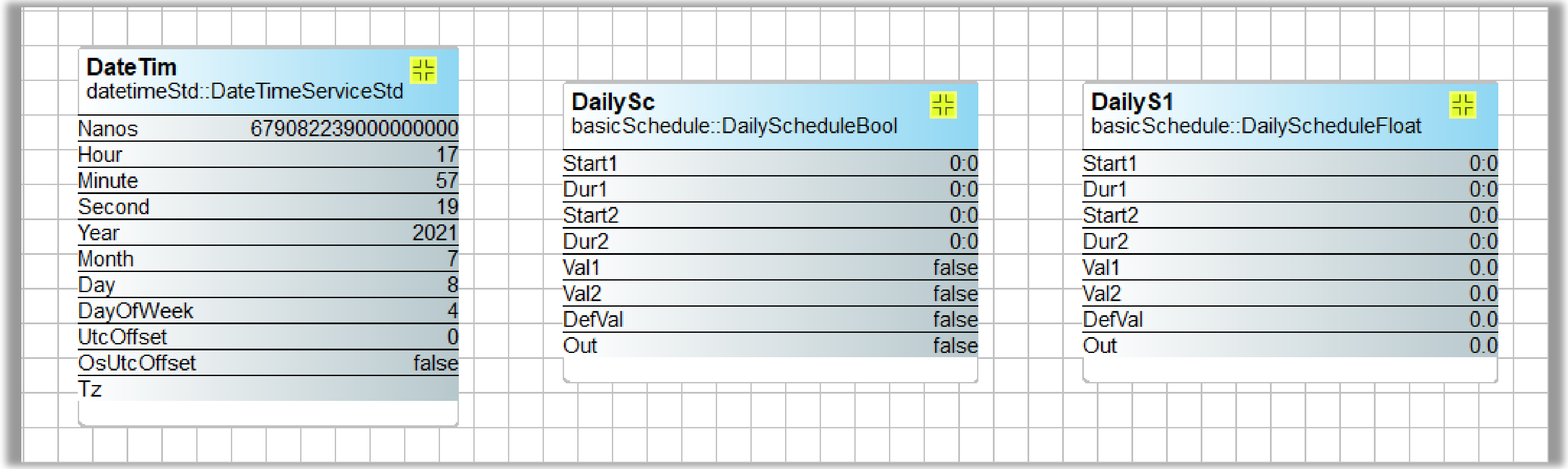
In this example, we are converting degrees Celsius to degrees Fahrenheit within the 0 - 100-degree Celsius range. Therefore, we set OutMin an OutMax to the corresponding Fahrenheit values. All Celsius input values between these two limits will be interpolated thereby providing the correct Fahrenheit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

Setting Tstat — Basic On/Off Temperature Controller



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.

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

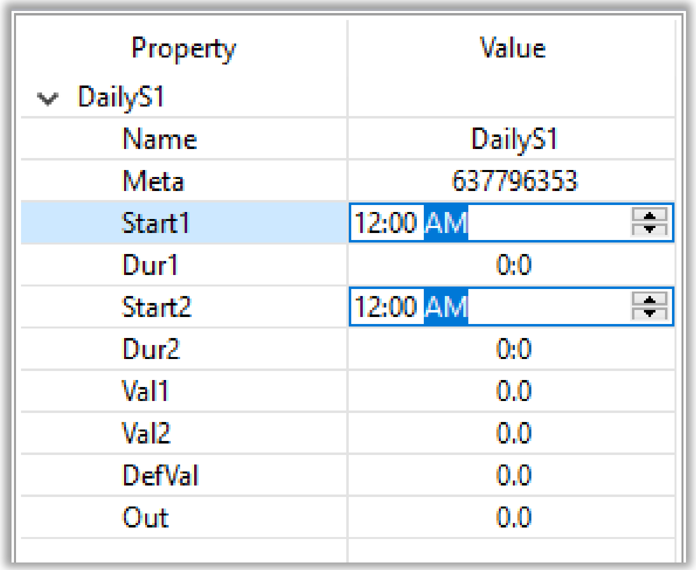
Start1	0:0
Dur1	0:0
Start2	0:0
Dur2	0:0
Val1	0.0
Val2	0.0
DefVal	0.0
Out	0.0

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.

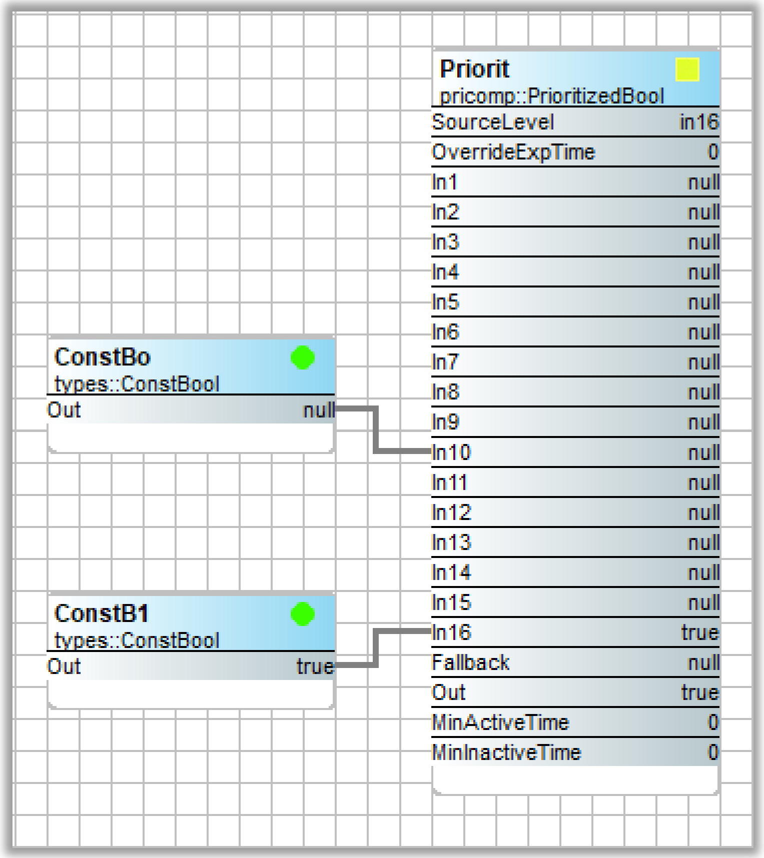
Setting the Real-Time Clock and Scheduling (continued)



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.

Creating Priority Arrays



ConstBo
types::ConstBool

Out	null

ConstB1
types::ConstBool

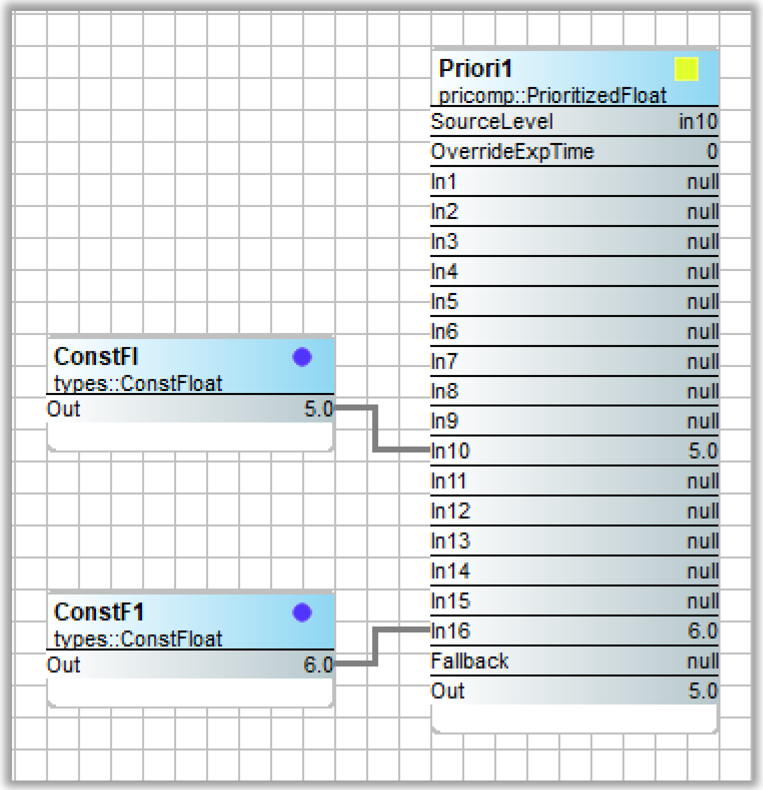
Out	true

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

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.

Creating Priority Arrays (continued)



ConstFl
types::ConstFloat

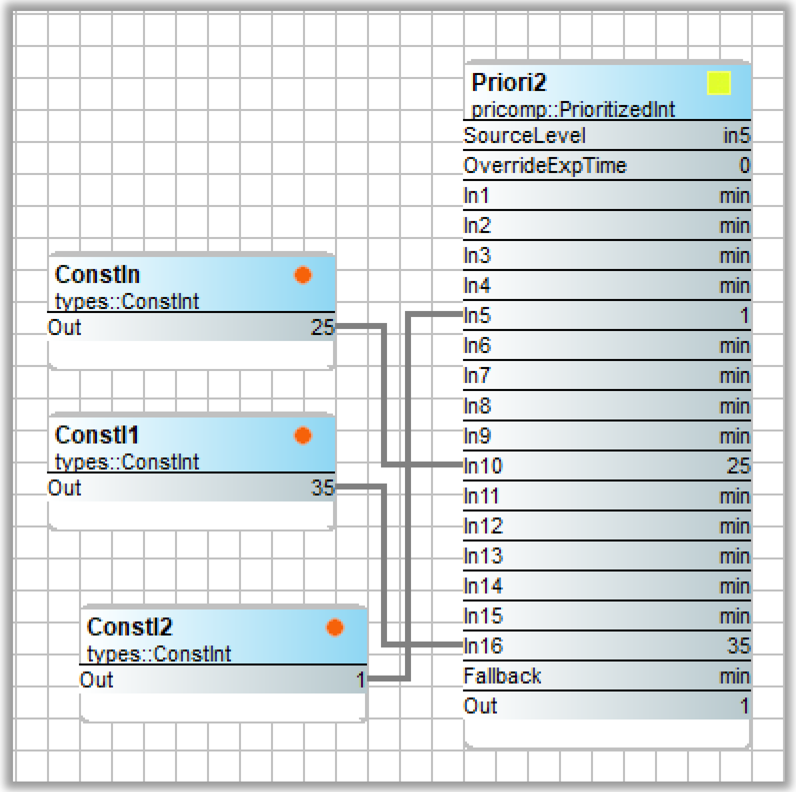
Out	5.0

ConstF1
types::ConstFloat

Out	6.0

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



ConstIn
types::ConstInt

Out	25

ConstI1
types::ConstInt

Out	35

ConstI2
types::ConstInt

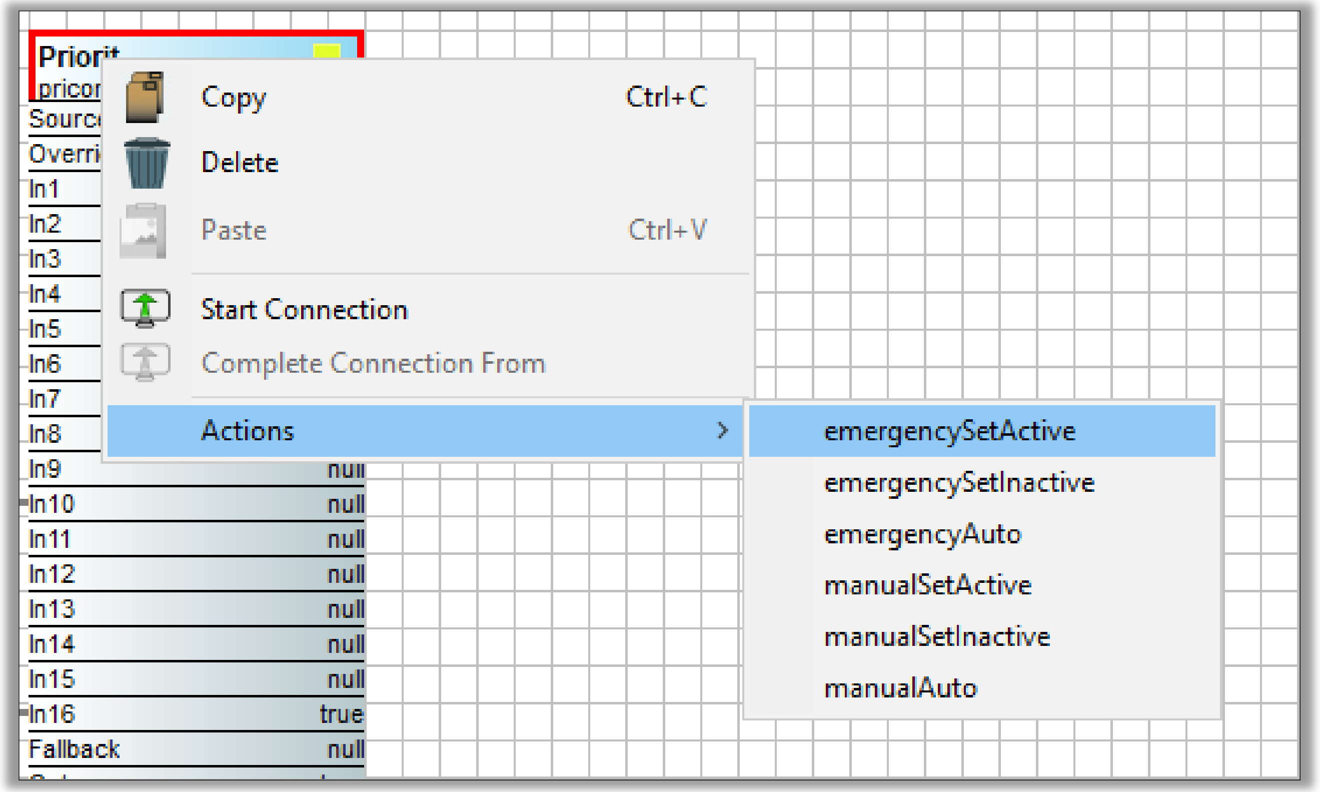
Out	1

Priori2
pricomp::PrioritizedInt

SourceLevel	in5
OverrideExpTime	0
In1	min
In2	min
In3	min
In4	min
In5	1
In6	min
In7	min
In8	min
In9	min
In10	25
In11	min
In12	min
In13	min
In14	min
In15	min
In16	35
Fallback	min
Out	1

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.

Creating Priority Arrays (continued)



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.

Thank You

Learn more at www.ccontrols.com