ObVerse Manual: Standard Processor

ObVerse is North's strategy language, and is available within Commander and ObSys. It allows an engineer to define how equipment should link and interoperate. This manual covers the features available in ObvProcessor within Commander and ObSys – the standard set.

This document relates to ObVerse Processor version 1.1 dated March 2016.

Please read the *Commander Tutorial* or *ObSys Tutorial* alongside this document, available from *www.northbt.com*

Contents

What is ObVerse? 4
- Processors 4
- Properties, Modules, and Comments 4
- Editing ObVerse 4

Quick Start 5
- Edit ObVerse in a Processor 5

Processors 6
- Standard Processors 6
- Advanced Processors 6

Properties 7
- Private Properties 7
- Public Properties 7
- Type Conversion 8
- Reserved References 8

Modules 9
- Inputs 9
- Outputs 9

Comments 10

Property Types in Standard Processors 11
- ENum 12
- Float 13
- NoYes 14
- Num 15
- Obj 16
- OffOn 17
- Text 18

Module Types 19
- Add 20
- Alarm 21
- Average 23
- Bits-To-Byte 24
- Bit-To-Num 26
- Byte-To-Bits 28
- Counter 30
- Delay 32
- Divide 33
- Equal 34
- Feedback 35
- Gate 37
- Greater 38
- Hysteresis 39
- Latch 41
- Lead-Lag 43
- Less 46
- Linearize 47
- Logical-And 50
- Logical-Inverse 51
- Logical-Exclusive-Or 52
- Logical-Or 53

Maximum 54
Minimum 55
Modulus-Remainder 56
Multiple-Add 57
Multiple-Logical-And 58
Multiple-Logical-Or 59
Multiply 60
Num-To-Bit 61
Object-Read 63
Object-Write 65
On-Off-Delay 67
Optimum-Start-Stop 69
Profile 72
Proportional-Integral-Derivative 74
Pulser 77
Raise-Lower 79
Random 81
Rescale 82
Select 83
Square-Root 85
Subtract 86
System-Information 87
Usage-Over-Period 89

ObVerse Standard Processor Versions 91

What is ObVerse?

Sometimes you need to do more than simply transfer a value from one object to another – you need to calculate something, delay something, or perform a more complex function on a value. North provides this flexibility with ObVerse, a cause-and-effect programming language.

ObVerse consists of a range of modules. The engineer selects particular modules and links them together to perform a desired strategy.

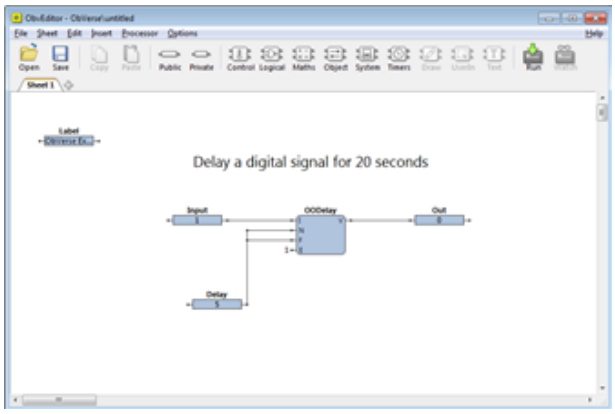


Fig. 1 Example ObVerse

Processors

ObVerse strategy runs in an ObVerse processor within a device.

ObVerse processors come in two types:

- **Standard Processor** – with logic, maths, and control modules
- **Advanced Processor** – with the same features found in a standard processor, plus extended maths and logic, display, application execution, directory and file services, and user-defined modules.

Both Commander and ObServer contain ObVerse standard processors.

ObView, part of the North ObSys software, contains an ObVerse advanced processor. Each ObView instance contains a processor.

This document covers the features found in an ObVerse standard processor.

Properties, Modules, and Comments

ObVerse strategy is made up from properties, modules and comments.

ObVerse properties are containers for storing data values, and carry a value from one module to another or between the processor and other tasks in the system.

ObVerse modules are used to perform an operation on one or more input values, and calculate a value. Properties are linked to the inputs and outputs to store these input and output values.

Comments in ObVerse are short pieces of text used to explain ObVerse strategy and make it easier for others to understand.

Editing ObVerse

You can create and edit ObVerse strategy using North's ObvEditor application, installed as part of the ObSys software. ObvEditor provides drag-and-drop graphical editing of ObVerse, uploading and downloading of ObVerse strategy, and run-time monitoring of the strategy within the processor.

Quick Start

If necessary, install North Engineering software onto your PC. It is available from *www.northbt.com*

Edit ObVerse in a Processor

💻 To edit ObVerse in a processor, follow these steps:

- → Run the **Start Engineering** application
- → If you are using a processor in the local PC, navigate to **ObServer**. If you are using a processor in Commander, navigate to **North IP Devices**, and select the required Commander.
- → Navigate to **Configuration**, and then select an **ObVerse Processor**
- → Select **ObVerse** to start the ObvEditor application.

For more information on using ObvEditor, refer to the section 'Introduction to ObVerse Programming' in the *Commander Tutorial*.

Processors

A processor runs ObVerse. There are two types of ObVerse processor: one supports a standard set of properties and modules, and the other supports a more advanced set.

Standard Processors

ObVerse standard processors perform essential control for a distributed system – and support modules for object reading and writing, maths, logic, and timing, as well as more advanced energy optimisations.

Commander has two ObVerse standard processors. Whenever Commander is powered-on, these processors run their ObVerse strategy continuously.

ObServer, the core of ObSys, has four ObVerse standard processors. When ObServer is running, these processors run their ObVerse strategy continuously.

The engineer can use the ObvEditor application to create and edit ObVerse strategy.

Each standard processor allows 1000 items within its strategy. These items are any combination of modules, properties and comments.

Advanced Processors

ObVerse advanced processors provide all the functions of a standard processor, as well as extra maths, logic, drawing, text, and user input modules. An advanced processor also supports custom modules, which the engineer creates and uses as required.

ObSys includes an application called ObView, and several copies can be run simultaneously. Each copy of ObView is an advanced processor.

Each copy of ObView can run the same or different ObVerse strategy file when it is started – this could be when Windows starts, when some ObVerse decides, or when the user decides.

The engineer may use ObView or the ObvEditor to create the ObVerse strategy files, but ObView loads them from the Windows file system directly when an ObView copy is started.

Properties

ObVerse properties are containers for storing data values. They can carry a value from one module to another, or between the processor and other tasks within the system – similar to a wire in an electronic circuit.

Properties have a data type, to define the type of value they hold – like a number or a text string. The range of types supported depends on the processor. For a complete list of the types supported by a standard processor, refer to the *Property Types in Standard Processors* section below. Properties sometimes hold values passing only between modules in the same processor. In ObVerse, we call these private properties, as their value is private to the processor.

Properties sometimes hold values passing between the processor an external task within the system. A task could be Essential Data, Data Transfer, Time Control, user action, or another ObVerse Processor; either in the same North device or another. In ObVerse, we call these public properties, as their value is publicly available.

A property can be assigned an initial value to use when the processor runs for the first time.

Private Properties

A private property holds a value as it passes between modules. One module assigns a value to it, and one or more modules use this value (Fig. 2). The property's value is not available outside the processor.

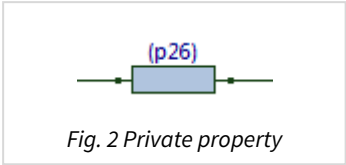


Fig. 2 Private property

A private property's reference must start with a lowercase character. The editor usually assigns the reference automatically, as the character 'p' followed by a number.

The engineer should link a module output to a property's left-side connector: the module will then store its output in that property.

The engineer should link a module input to a property's right-side connector: the module will then use the value stored in the property for its input, when it is performing its operation.

Public Properties

A public property holds a value that both modules and external tasks can access.

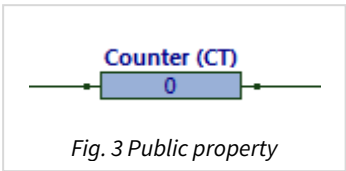


Fig. 3 Public property

A public property must have an reference that starts with an uppercase character. This reference also becomes the object reference used by external tasks to access the value from the ObVerse processor.

External tasks can read the value of a public property, and if adjustable, write the value. A property becomes adjustable by an external task when a module is not assigning a value to it within ObVerse.

The engineer may link a module output to the property's left-side connector: the module will then store its output in that property.

If the left-side connector is not linked to a module, the property becomes adjustable by an external task, and would hold its initial value until a task wrote a new value into it.

The engineer may link a module input to the property's right-side connector, the property will then provide the input value to the module when it is performing its operation.

When inserting a public property, in addition to an initial value and reference, other parameters may be included to help describe the property to an external task. These vary depending on the data type, but could include a label, high and low value limits, read rates, etc.

In the example public property shown above (Fig. 3), the reference has been set to ‘CT’, the label has been set to ‘Counter’, and its initial value is ‘0’. Because both left and right-hand sides are linked (but not fully shown), a module to the left will write a value into the property (overwriting the initial value), and a module (or more) to the right will use the value.

For more information on accessing public properties from an external task, refer to the ObVerse object within the *Commander Manual* document.

Type Conversion

If a module assigns a value of one type into a property of a different type, the value is automatically converted to the type required by the property. Similarly, if a module input reads a value from a property of another type, the value will be automatically converted to that required by the module input.

Reserved References

The processor reserves some property references within ObVerse for special purposes.

Description	Reference	Type
Label Label for ObVerse. If this property is present, the processor returns its value when scanned by the ObView engineering tool. The value is also used as the System field for alarms sent by the Alarm module.	L	Obj\Text: 31 chars; Adjustable
Remote Object Prefix Prefix to add in the front of the object reference used by ObjRead and ObjWrite modules. If a module has its absolute option set, then this prefix is not used.	O	Obj\Obj; Adjustable
Alarm Object If present, the Alarm module routes alarms to the object reference specified. Typically, this object reference ends ‘.ALARM. If this property is not present, then the Alarm module routes alarms to the North device’s ALARM object.	AO	Obj\Obj; Adjustable

Modules

Modules calculate values. They take one or more inputs, and calculate one or more outputs.

Different modules are available to perform different operations. The range of modules supported depends on the processor. For a complete list of the modules supported by a standard processor, refer to the *Module Types* section below.

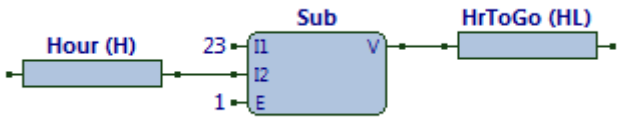


Fig. 4 Module with inputs and outputs

Inputs

Each module has one or more inputs. An input can either be set with a constant value, or linked to a property.

The example strategy above (Fig. 4) shows a subtract module (Sub) with its three inputs – I1, I2, and E. Input I1 has been set to the constant value '23', input I2 has been linked to property H and uses the property's value, and input E has been set to the constant value '1' (the default value).

ObVerse allows several module inputs to be linked to a property, if those modules all need to use its value.

Outputs

Each module has one or more outputs. To use a module's output value, either as an input to another module or a value for an external task, then connect the output to a property. The module will then update the property's value whenever it calculates a new value.

The example strategy above (Fig. 4) shows a subtract module (Sub) with its one output – V. Output V has been linked to property HL, so stores its output value in that property.

Only one module can write to a particular property, otherwise it becomes difficult to understand which value the property holds at any time.

Comments

A comment is a short piece of text added to help understand the design of a piece of strategy (Fig. 5). They are optional, but we recommend they are used – it is surprising how quickly we all forget how our strategy works (or should work).

In ObvEditor, a comment can be set as a title, and shown in large text.

You can also use the different sheets, along with a descriptive title, to organise your ObVerse. Public properties should be labelled too, as this will also aid understanding.

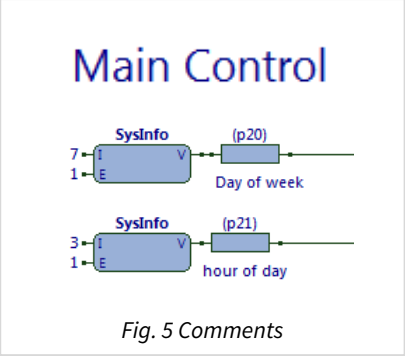


Fig. 5 Comments

Property Types in Standard Processors

A property within a processor must have a unique reference and a type. The ObVerse standard processor supports the following types of properties:

ENum
Float
NoYes
Num
Obj
OffOn
Text

Other value types can be handled by using a Text property.

ENum

Object Type: *[Obj\ENum]*

An ENum property holds a single enumerated value as a positive integer. A list of text strings defines a label for each value. E.g. 'Off, On, Auto' represents values 0, 1, and 2.

The value is in the range 0 to *n*, where *n*+1 is the number of options. Some values may not be valid. Display systems convert the value from a number to text using the Alternatives parameter.

An ENum property contains the following parameters:

Description	Reference	Type
Label Label of parameter	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\ENum: 0…20
Alternatives List of alternatives, separated by commas, where 0 is the first value, 1 is the second, and so on	A	Obj\Text: 64 chars
Read Rate (s) Indicate how often an external task should read the property's value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0..1; Where: 0=Writable; 1=Write Inhibited

Float

Object Type: *[Obj\Float]*

A Float property holds a single floating-point value – a number with a decimal point. E.g. 623.5 or -3.14

Floating-point numbers are held in IEEE 754 format, which is accurate to approximately 6 ½ significant figures, in the range 9999999×10^{90} to -9999999×10^{90}

A Float property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\Float
Value High Highest value allowed in the property	VH	Obj\Float
Value Low Lowest value allowed in the property	VL	Obj\Float
Decimal Places Number of decimal places to display	D	Obj\Num: 0..4
Read Rate (s) Indicate how often an external task should read the property's value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0…1; Where: 0=Writable; 1=Write Inhibited

NoYes

Object Type: *[Obj\NoYes]*

A NoYes property holds a single binary state, no or yes, as a number.

The value '0' (zero) represents the **no** state, and the value '1' represents the **yes** state. Display systems will convert the number to the text 'No' or 'Yes'.

A NoYes property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\NoYes
Read Rate (s) Indicate how often an external task should read the property's value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0..1; Where: 0=Writable; 1=Write Inhibited

Num

Object Type: *[Obj\Num]*

A Num property holds a single integer value – a whole number with no decimal. E.g. 623 or -3

The value is in the range -2,147,483,648 to +2,147,483,647.

A Num property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\Num: -2147483648… 2147483647
Value High Highest value allowed in the property	VH	Obj\Num: -2147483648… 2147483647
Value Low Lowest value allowed in the property	VL	Obj\Num: -2147483648… 2147483647
Read Rate (s) Indicate how often an external task should read the property's value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0…1; Where: 0=Writable; 1=Write Inhibited

Obj

Object Type: *[Obj\Obj]*

An Obj property holds an object reference as a text string. E.g. 'PL' or 'S1.M5.DI2.S'.

An Obj property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\Obj
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0…1; Where: 0=Writable; 1=Write Inhibited

OffOn

Object Type: *[Obj\OffOn]*

An OffOn property holds a single binary state, off or on, as a number.

The value '0' (zero) represents the **off** state, and the value '1' represents the **on** state. Display systems will convert the number to the text 'Off' or 'On'.

An OffOn property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property's value to use when ObVerse is first run. Typically, the property's value is preserved during restarts.	IV	Obj\OffOn
Read Rate (s) Indicate how often an external task should read the property's value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0…1; Where: 0=Writable; 1=Write Inhibited

Text

Object Type: *[Obj\Text]*

A Text property holds a single text string value. E.g. ‘Label’

A Text property contains the following parameters:

Description	Reference	Type
Label Label of property	L	Obj\Text: 20 chars
Initial Value Property’s value to use when ObVerse is first run. Typically, the property’s value is preserved during restarts.	IV	Obj\Text
Maximum Length Maximum characters to be stored for value	ML	Obj\Num: 1..31
Read Rate (s) Indicate how often an external task should read the property’s value	R	Obj\Num
Write Inhibit Indicates whether external tasks can adjust the value. ObvEditor generates this automatically.	WI	Obj\ENum: 0…1; Where: 0=Writable; 1=Write Inhibited

Module Types

The ObVerse standard processor supports the following modules:

Maths

Add
Subtract
Multiply
Divide
Modulus-Remainder
Multiple-Add
Average
Minimum
Maximum
Byte-To-Bits
Bits-To-Byte
Num-To-Bit
Bit-To-Num
Random
Rescale
Square-Root
Linearize
Usage-Over-Period

Logic

Logical-And
Logical-Or
Logical-Inverse
Equal
Gate
Greater
Less
Logical-Exclusive-Or
Multiple-Logical-And
Multiple-Logical-Or
Select

Control

Feedback
Hysteresis
Lead-Lag
Optimum-Start-Stop
Proportional-Integral-Derivative
Raise-Lower

Timers

Counter
Delay
Latch
On-Off-Delay
Profile
Pulser

System

System-Information

Object

Alarm
Object-Read
Object-Write

Add

Object Type: *[Obv\Add]*

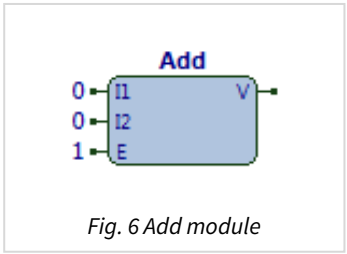


Fig. 6 Add module

The Add module (Fig. 6) performs the maths operation to add two numbers together.

When enabled, the formula is:

```
V = I1 + I2
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

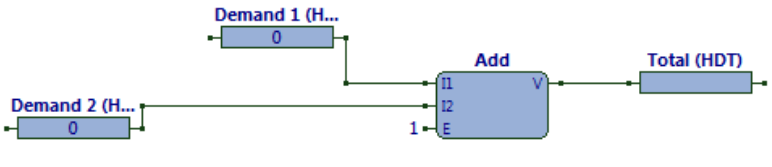


Fig. 7 Add module example

The ObVerse strategy (Fig. 7) shows the adding together of two demands to determine the total demand.

The Add module's two input values, I1 and I2, are provided by the linked properties. The output, V, passes the result to the linked property HDT. Enable is set to '1' (Yes).

Some external task will write values in to Demand 1 and 2. If the input values were set to '1.7' and '2.9', the output value would be '4.6'.

Related Modules

Subtract, *Multiple-Add*

Alarm

Object Type: *[Obv\Alarm]*

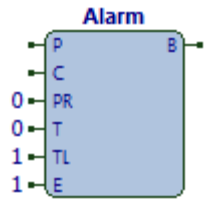


Fig. 8 Alarm module

The Alarm module (Fig. 8) performs the remote object operation to generate and send a North-format alarm message when triggered.

When enabled, the operation is:

```
if T == TL then
   B = Alarm(P, C, PR)
```

North-format alarms contain six text fields. The Alarm module places the following information into these fields:

System – from property L within the ObVerse, see *Reserved References* above

Point – set from input **P**

Condition – set from input **C**

Priority – set from input **PR**

Date & Time – from North device running the ObVerse processor

The module sends the alarm to the device's alarm processing, which can deliver it to one or more destinations. If the property AO is present within the ObVerse, then the alarm is delivered to this destination instead (see *Reserved References* above).

The module contains the following objects:

Description	Reference	Type
Point Text used for the point field in the alarm message	P	Obj\Text; Adjustable; Max chars: 30; Default value: ''
Condition Text used for the condition field in the alarm message	C	Obj\Text; Adjustable; Max chars: 30; Default value: ''
Priority Importance of the alarm notification, where: 1=Life safety, 2=Property safety; 3=Major,.. 9=Information only; 0=No Alarm	PR	Obj\Num; Adjustable; Range 0, 1...9; Default value: 0
Trigger The trigger input, to cause the alarm to be sent when it changes to the Trigger Level (TL)	T	Obj\NoYes; Adjustable; Default value: 0 (No)
Trigger Level The state to which the Trigger input (T) must be equal, to cause the alarm to be sent	TL	Obj\NoYes; Adjustable; Default value: 0 (No)
Enable Enables the module's operation. If set to '0' (No), then no operation occurs	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Busy Set to 1 (Yes) when the alarm is in the process of being sent	B	Obj\NoYes

Example

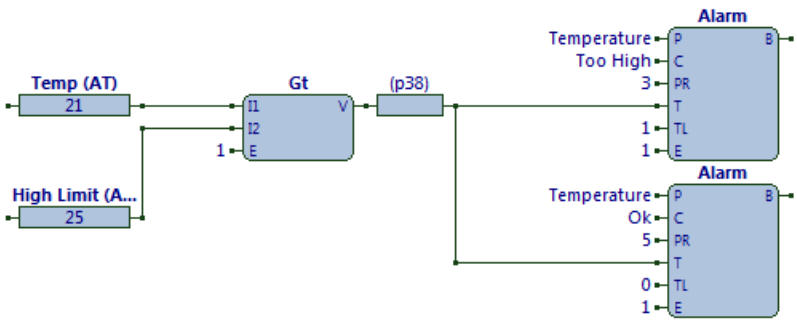


Fig. 9 Alarm module example

This ObVerse strategy (Fig. 9) sends an alarm message when a temperature exceeds a high limit, and when it returns below the limit. In both Alarm modules, notice how the trigger level (TL), priority (PR), and condition (C) inputs all differ.

Enable is set to the constant value '1' (Yes) on both Alarm modules.

Some external task will write the current temperature to the Temp property, AT. Some external task may write a different value into the High Limit property.

When the current temperature in Temp property is greater than that in the High Limit property, the greater module (Gt) will output the value '1'. This will trigger the upper Alarm module to send its 'Too High' alarm message. If the temperature falls below this value, the Greater (Gt) module will output the value '0'. This will trigger the lower Alarm module to send its 'Ok' alarm.

If there is an object with reference AO in the ObVerse strategy, the module will send alarm messages to the object specified within it; otherwise it will send alarm message to the object ALARM, which routes the messages to the Alarm Delivery task.

An alarm will be sent once by an Alarm module, for each time its trigger input changes to equals its trigger level input.

Average

Object Type: *[Obv\Ave]*

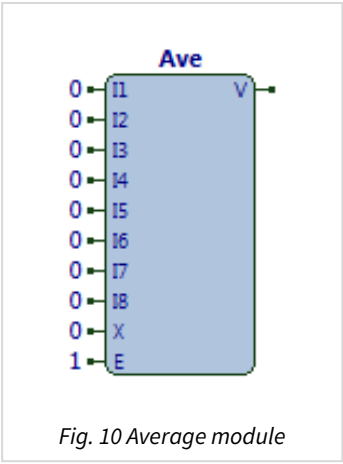


Fig. 10 Average module

The Ave module (Fig. 10) performs the maths operation to find the average of up to eight numbers.

When enabled, the formula is:

$$V = (I1 + I2 + I3 + \ldots + I_x) / X$$

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of inputs (starting from I1) to include in calculation	X	Obj\Num; Range: 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

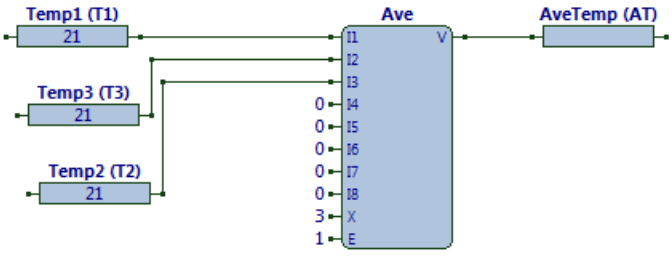


Fig. 11 Average module example

The ObVerse strategy (Fig. 11) shows a simple averaging of three temperature sensors.

The Ave module's three input values, I1 to I3, are provided by the linked properties. Input X is set to '3'. The output, V, passes the result to the linked property AT. Enable is set to '1' (Yes).

Some external task will write values into each of the Temp properties. With the input values set to '21', '22', and '23', the output value will be '22'.

Remember it is possible to set Zip inputs so that they give an override value whenever a sensor is out-of-limits.

Related Modules

Minimum, *Maximum*

Bits-To-Byte

Object Type: *[Obv\BitsToByte]*

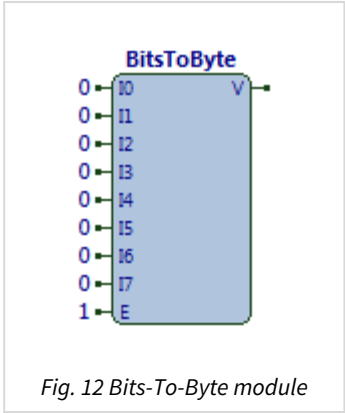


Fig. 12 Bits-To-Byte module

The BitsToByte module (Fig. 12) performs the maths operation to build a byte value from a set of eight individual bits.

When enabled, the formula is:

```
V = 0
if I0 then
   V = V + 1
if I1 then
   V = V + 2
if I2 then
   V = V + 4
if I3 then
   V = V + 8
if I4 then
   V = V + 16
if I5 then
   V = V + 32
if I6 then
   V = V + 64
if I7 then
   V = V + 128
```

The module contains the following objects:

Description	Reference	Type
Bit *x* The bit number to set. Input to include in calculation, where *x* is in the range 0..7. Input 0 is the least significant bit, 8 the most significant.	I*x*	Obj\OffOn; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Num: 0…255

Example

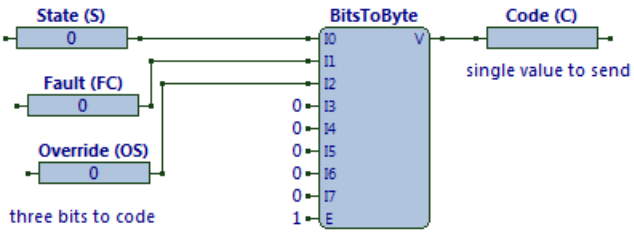


Fig. 13 Bits-To-Byte module example

The ObVerse strategy (Fig. 13) shows three binary states converted to a byte value for easy transmission to another device (where the value will be decoded back to the three states).

The BitsToByte module's three bit values, I0 to I2, are provided by the linked properties. The output, V, passes the result to the linked property C. Enable is set to the constant value '1' (Yes).

Some external task will write the State, Fault, and Override values. If State and Override properties are set to ‘1’, and property Fault is cleared to ‘0’, then input bits 0 and 2 are set to ‘1’, and the output value will be ‘5’.

Related Modules

Byte-To-Bits, *Num-To-Bit*, *Bit-To-Num*

Availability

Available in standard and advanced processor versions dated September 2012 and later.

Bit-To-Num

Object Type: *[Obv\BitToNum]*

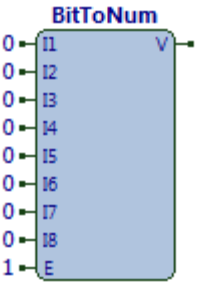


Fig. 14 Bit-To-Num module

The BitToNum module (Fig. 14) performs the maths operation to output the highest input number set, from up to eight input states. The output is set to '0' when no inputs are set.

When enabled, the formula is:

```
if I8 then
   V = 8
else if I7 then
   V = 7
else if I6 then
   V = 6
else if I5 then
   V = 5
else if I4 then
   V = 4
else if I3 then
   V = 3
else if I2 then
   V = 2
else if I1 then
   V = 1
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input Bit *x* Input to include in calculation, where *x* is in the range 1..8	I*x*	Obj\NoYes; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Highest Bit Set The last calculated value	V	Obj\Num: 0…8

Example

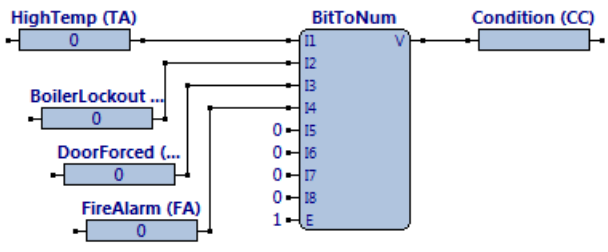


Fig. 15 Bit-To-Num module example

The ObVerse strategy (Fig. 15) shows the BitToNum module used to monitor a number of alarm statuses and output the highest priority – in this case a fire alarm is condition 4, whilst a boiler lockout is condition 2.

The BitToNum module's four inputs, I1 to I4, are provided by the linked properties. The output, V, passes the result to the linked property CC. Enable is set to the constant value '1' (Yes).

Some external task writes the states of the four input properties HighTemp, BoilerLockout, DoorForced, and FireAlarm. When input 4, from property FireAlarm, is set to '1', the output value will be '4'. When input 1, from property HighTemp is set to '1' and all other inputs are '0', the output value will be '1'.

Related Modules

Num-To-Bit, *Byte-To-Bits*, *Bits-To-Byte*

Availability

Available in standard and advanced processor versions dated September 2012 and later.

Byte-To-Bits

Object Type: *[Obv\ByteToBits]*

The ByteToBits module (Fig. 16) performs the maths operation to separate a byte value into eight individual bit states.

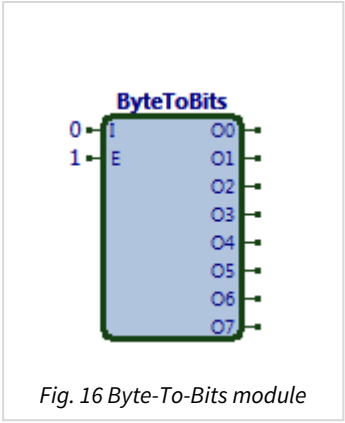


Fig. 16 Byte-To-Bits module

When enabled, the formula is:

```
if I is odd then
   O0 = 1
if (I/2) is odd then
   O1 = 1
if (I/4) is odd then
   O2 = 1
if (I/8) is odd then
   O3 = 1
if (I/16) is odd then
   O4 = 1
if (I/32) is odd then
   O5 = 1
if (I/64) is odd then
   O6 = 1
if (I/128) is odd then
   O7 = 1
```

The module contains the following objects:

Description	Reference	Type
Input Input byte value to convert to bits	I	Obj\Num; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs are left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Bit *x* The last calculated value *x* for each bit of the byte, where *x* is the in the range 0..7	O*x*	Obj\OffOn

Example

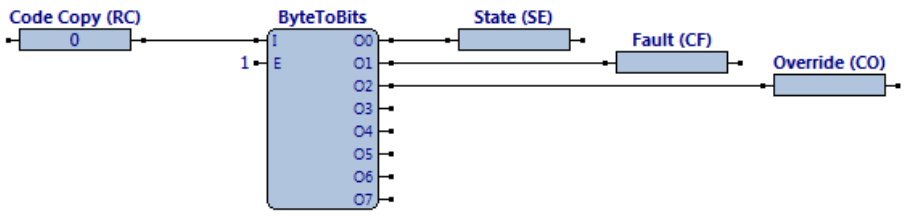


Fig. 17 Byte-To-Bits module example

The ObVerse strategy (Fig. 17) shows a byte value converted to three binary states.

The ByteToBits module's input value, I, is provided by the linked property. The bit outputs, O0 to O2, pass the result to the linked properties. Enable is set to '1' (Yes).

Some external task writes a value to the Code Copy property. When the value written is '5', the outputs will set the output properties to State = '1', Fault = '0', and Override = '1'.

Related Modules

Bits-To-Byte, *Bit-To-Num*, *Num-To-Bit*

Availability

Available in standard and advanced processor versions dated September 2012 and later.

Counter

Object Type: *[Obv\Counter]*

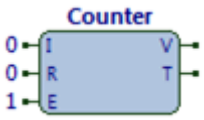


Fig. 18 Counter module

The Counter module (Fig. 18) performs the timer operation to increment a counter value when the input state transitions from off to on. The module also outputs the total time the input has been on.

An input allows the counter and timer outputs to be reset.

When enabled, the operation is:

```
if I then
   V = V + 1
   T = T + Time(I)
if R then
   V = 0
   T = 0
```

The module contains the following objects:

Description	Reference	Type
Input Input to count and time	I	Obj\OffOn; Adjustable; Default value: 0 (Off)
Reset Set to 1 (Yes) to reset both Value and Timer outputs. Set to 0 (No) to enable counting	R	Obj\NoYes; Adjustable; Default value: 0 (No)
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs are left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Count The count of times the Input has changed from 0-to-1 (Off-to-On)	V	Obj\Num: 0…2147483647
Run Time (s) The number of seconds the Input has been set to 1 (On)	T	Obj\Num: 0…2147483647

Example

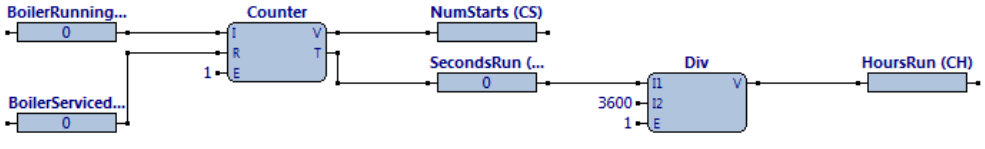


Fig. 19 Counter module example

The ObVerse strategy (Fig. 19) shows a Counter module counting the number of boiler starts, and timing how long the boiler runs in hours. Additional strategy could compare the number of starts and hours run to determine when to service the boiler.

The Counter module's input value, I, is provided by the BoilerRunning property. The output, V, passes the number of starts to the property NumStarts. The output, T, passes the time run period via a Divide module to the property HoursRun for total hours run. Enable is set to the constant value '1' (Yes).

Some external task writes values into properties BoilerRunning and BoilerServiced. When the BoilerServiced property is set to ‘1’, the Counter module resets the properties NumStarts and SecondsRun to ‘0’. When BoilerServiced is set to ‘0’, the module will start its counting and timing function. When BoilerRun is set to ‘1’, the module will increment the NumStarts property, and every complete second the BoilerRun property is still ‘1’, the module will increment the SecondsRun property.

Delay

Object Type: *[Obv\Delay]*

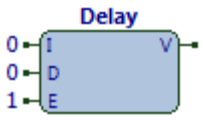


Fig. 20 Delay module

This module is for legacy applications only. Use the *On-Off-Delay* module which provides both an on and off delay period.

The Delay module (Fig. 20) performs the timer operation to delay a digital input state, before setting the output value to the same state. The delay time is specified in seconds.

When enabled, the operation is:

```
onChange(I) then
   if I != V for D seconds
      V = I
```

The module contains the following objects:

Description	Reference	Type
Input Input state to delay	I	Obj\OffOn; Adjustable; Default value: 0 (Off)
Delay (s) Number of seconds to delay the output value	D	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (No)
Output Value The delayed input state	V	Obj\OffOn

Example

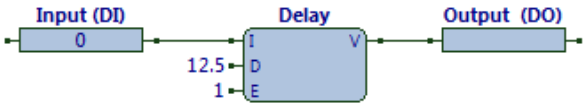


Fig. 21 Delay module example

The ObVerse strategy (Fig. 21) shows a Delay module delaying an input state for 12.5 seconds, before outputting this same state.

The Delay module's input value, I, is provided by the linked property DI. The delay input, D, is set to '12.5'. The output, V, passes the result to the linked property DO. Enable is set to the constant value '1' (Yes).

Some external task writes a state to property Input. When the value is set to '1', the output will be set to '1' after 12.5 seconds. When the value is set to '0', the output will be set to '0' after 12.5 seconds.

Related Modules

On-Off-Delay

Divide

Object Type: *[Obv\Div]*

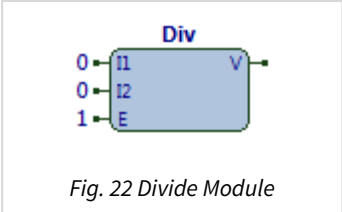


Fig. 22 Divide Module

The Div module (Fig. 22) performs the maths operation to divide two numbers.

When enabled, the formula is:

```
V = I1 / I2
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..2 Input 1 is the dividend, Input 2 is the divisor	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value, the quotient	V	Obj\Float

Example

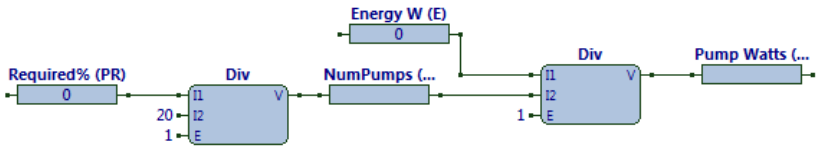


Fig. 23 Divide module example

The ObVerse strategy (Fig. 23) shows two Div modules. The first divides a Required% parameter by a constant, 20, to determine the number of pumps to run – i.e. 100% is equivalent to 5 pumps. The NumPumps property is type Num, and so holds only whole numbers – in this case 0..5. This property could be used to run a particular number of pumps.

The second Div module divides the total energy being used by the number of pumps running, to determine the average Watts used by each pump.

Some external task writes a percentage to the Required% property, and some task writes a total energy value to property Energy W. When the Require% is set to '66' and Energy W set to '620', the NumPumps will be calculated as '3' (the Div modules output, V, is '3.3' and converted from a type float to num) and Pump Watts as '206.667'.

Related Modules

Multiply, *Modulus-Remainder*

Equal

Object Type: *[Obv\Equal]*

The Equal module (Fig. 24) performs the logic operation to compare two inputs for equality.

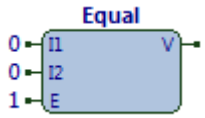


Fig. 24 Equal module

When enabled, the formula is:

```
if I1 == I2 then
   V = 1
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1
Value The last calculated value	V	Obj\NoYes

Example

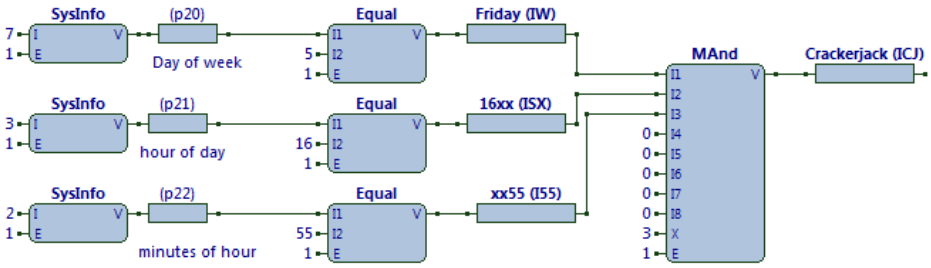


Fig. 25 Equal module example

The ObVerse strategy (Fig. 25) shows three SysInfo modules feeding into three Equal modules. SysInfo modules are passing the day-of-week, hour-of-day, and minute-of-hour into the Equal modules. These check to see whether the day-of-week is currently '5' (i.e. Friday), the current hour is '16' (i.e. 4pm), and the current minute is set to '55'. When all three of these are equal (i.e. its Friday at 16:55), then the Crackerjack property is set '1'. After 1 minute, the minute-of-hour will increment, and the lower Equal will set its output to '0', the MAnd module will set Crackerjack to '0'.

Related Modules

Greater, *Less*

Feedback

Object Type: *[Obv\Feedback]*

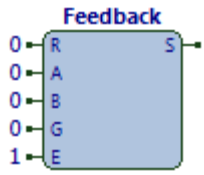


Fig. 26 Feedback module

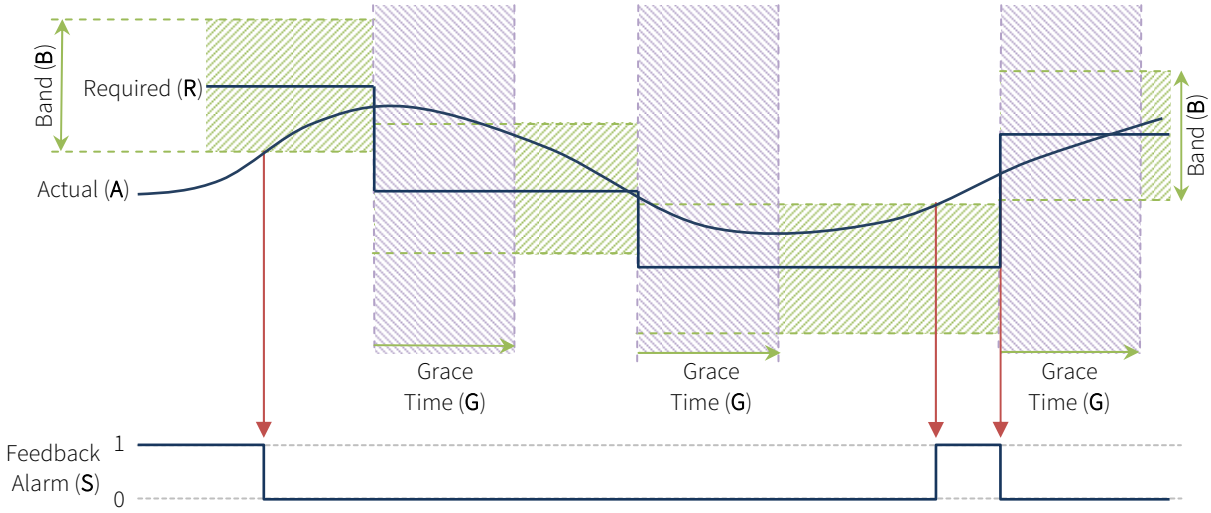
The Feedback module (Fig. 26) performs the control operation to determine whether an input is under control with respect to a required value.

It has a band of variance, which specifies how much the input can vary around the required value. It also has a grace time (in seconds), which specifies the grace time allowed after the required value changes before the input must return within the variance band.

If the input is outside of the band, excepting for grace periods, the status is set to '1'. Specify the required value, actual value and band in the same units.

When enabled, the operation is:

```
S = Feedback(R, A, B, G)
```



The module contains the following objects:

Description	Reference	Type
Required The required value of the thing being controlled	R	Obj\Float; Adjustable; Default value: 0
Actual The actual value of the thing being controlled	A	Obj\Float; Adjustable; Default value: 0
Band The amount of variability in the Actual value that is acceptable. This is distributed evenly around the Required value, 50% above and 50% below	B	Obj\Float; Adjustable; Default value: 0
Grace Time (s) The time after a change in the Required value during which the Actual is allowed to exceed the Band	G	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and S is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)

Description	Reference	Type
Feedback Alarm Set if the Actual value is outside the required band, and outside the Grace Time	S	Obj\NoYes

Example

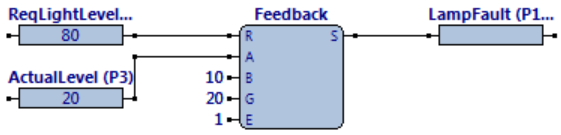


Fig. 27 Feedback module example

The ObVerse strategy (Fig. 27) shows feedback used to monitor the controlled light level against a required level. A band is specified to allow a variance around the required level. The grace time provides a delay during which time the actual level can reach the required level.

The Feedback module's required and actual values are taken from properties ReqLightLevel and ActualLevel. The band input, B, is set to a constant '10' and grace timer input, G, set to 20 seconds. Enable is set to '1' (Yes).

Some external task writes a value into ReqLightLevel property. Some external task writes the current light level into the property ActualLevel. When the required value is changed to '80', and band to '10', then the feedback module allows waits for the grace timer (20 seconds) before checking that the ActualLevel is in the range 75…85. If, after the grace timer, the actual value is outside this value then output S will be set '1' (Yes).

Gate

Object Type: *[Obv\Gate]*

The Gate module (Fig. 28) performs the logic operation to allow one of two possible input values through to the output, based on a switch input.

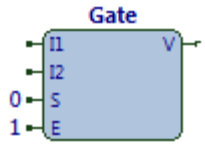


Fig. 28 Gate module

When enabled, the operation is:

```
if S then
   V = I2
else
   V = I1
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to pass to output, dependent on Switch, where *x* is in the range 1..2	I*x*	Obj\Text; Adjustable; Default value: ‘’
Switch If set to ‘0’, then Input 1 is passed to the output. If set to a non-zero number, then Input 2 is passed to the output	S	Obj\ENum; Adjustable; Values: 0=Input 1, 1=Input 2 Default value: 0
Enable Enables the module’s operation. If set to ‘0’ (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last passed value	V	Obj\Text

Example

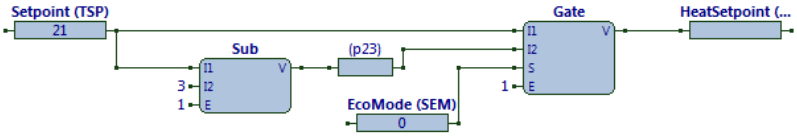


Fig. 29 Gate module example

The ObVerse strategy (Fig. 29) shows a heating setpoint calculated depending on a specified setpoint and EcoMode flag.

The Gate module’s two input values, I1 and I2, are provided by properties: one is linked directly to the Setpoint property; the other is calculated by the Sub module, as Setpoint-3. The switching is provided by the EcoMode property. The Gate module output, V, writes the resultant value to the HeatSetpoint property. Enable is set to the constant value ‘1’ (Yes).

Some external task writes the required setpoint to the Setpoint property. If Setpoint is set to ‘21’, private property p23 will be set to ’18’ (21-3). If some external task sets EcoMode to ‘0’, then the Gate output will be set to ‘21’ from input I1. If EcoMode is set to ‘1’, then the output will be set to ‘18’ from I1.

Related Modules

Select

Greater

Object Type: *[Obv\Gt]*

The Gt module (Fig. 30) performs the logic operation to determine whether one input is greater than the other input.

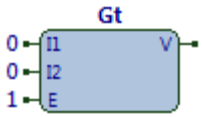


Fig. 30 Greater module

When enabled, the formula is:

```
if I1 > I2 then
   V = 1
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input x to compare, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '1' (Yes) if Input 1 is greater than Input 2	V	Obj\NoYes

Example

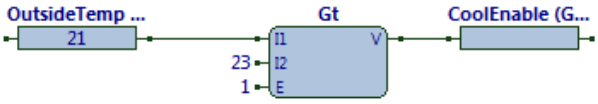


Fig. 31 Greater module example

The ObVerse strategy (Fig. 31) shows the outside air temperature used to enable cooling. If the temperature is greater than 23, then cooling is enabled.

The Gt module's input, I1, is provided by a linked property. Input, I2, is set to a constant '23'. The output passes the result to property CoolEnable. Enable is set to the constant value '1' (Yes).

Some external task writes a value to the OutsideTemp property. If OutsideTemp set to '21', the Gt module output value will be '0', and will be written to the CoolEnable property.

Related Modules

Less, *Equal*

Hysteresis

Object Type: *[Obv\Hyst]*

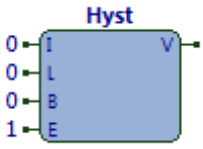


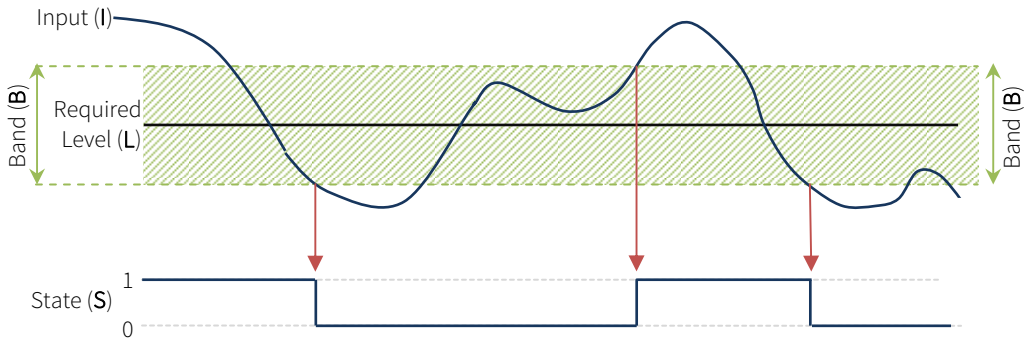
Fig. 32 Hysteresis module

The Hyst module (Fig. 32) performs the hysteresis control operation to determine an output depending on a required level, a band of variance, and the current state of the output.

It has a band of variance, which specifies how much the input can vary around the required level value. If the input is above the specified band, the value is set to '1'. If the input is below the specified band, the value is set to '0'. If the input is within the specified band, the output is not changed.

When enabled, the operation is:

```
V = Hyst(I, L, B)
```



The module contains the following objects:

Description	Reference	Type
Input The actual value of the thing being controlled	I	Obj\Float; Adjustable; Default value: 0
Required Level The required value of the thing being controlled	L	Obj\Float; Adjustable; Default value: 0
Band The amount of variability in the Input value that is acceptable, in which no output switching is done. The Band is distributed evenly around the Required Level, 50% above and 50% below.	B	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
State Set to '1' (On) if the Input value is above the required band, and set to '0' (Off) if the Input is below the required band. If the Input is within the band, this state is left unchanged	V	Obj\OffOn

Example

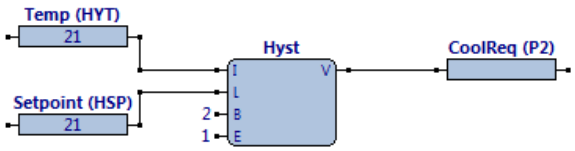


Fig. 33 Hysteresis module example

The ObVerse strategy (Fig. 33) shows hysteresis used to calculate when to enable cooling. A band is specified to allow a variance around the setpoint. When the temperature exceeds this band, the cooling output is enabled until the temperature falls below the lower band.

The Hyst module's I and L levels are provided by the Temp and Setpoint properties. The band input, B, is set to '2'. Enable is set to '1' (Yes).

Some external task writes a setpoint to the Setpoint input. Some external task writes the current temperature to the Temp property. If the Setpoint is set to '21', and band to '2', then the hysteresis module allows an input value in the range 20…22 without affecting the output. However, if the input value rises above '22', the output will write '1' (On) to the CoolReq property. The CoolReq property will only be set to '0' when the Temp property falls below '20'.

If the application were for heating rather than cooling, invert the output using the Logical-Inverse module.

Related Modules

Greater, *Less*

Latch

Object Type: *[Obv\Latch]*

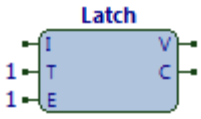


Fig. 34 Latch module

The Latch module (Fig. 34) performs the timer operation to latch an input value, and to indicate when the value has changed.

When enabled, the operation is:

```
if (T == 1) and (V != I) then
   C = 1
else
   C = 0

if T == 1 then
   V = I
```

The module contains the following objects:

Description	Reference	Type
Input Input value to latch	I	Obj\Text; Adjustable; Default value: ‘’
Trigger Enables passing the Input value to the output Value. Typically, this is linked to a pulsed value. Set to ‘1’ (Yes), to pass the Input value to the output Value, and enable value Changed notifications. Set to ‘0’ (No) to stop passing the Input value, latching the output Value.	T	Obj\NoYes; Default value: 1 (Yes)
Enable Enables the module’s operation. If set to ‘0’ (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to the input value when Trigger is set ‘1’ (Yes)	V	Obj\Text
Changed Pulsed output – set to ‘1’ (Yes) when the Value has changed, then resets to ‘0’ (No) after one processor cycle	C	Obj\NoYes

Example

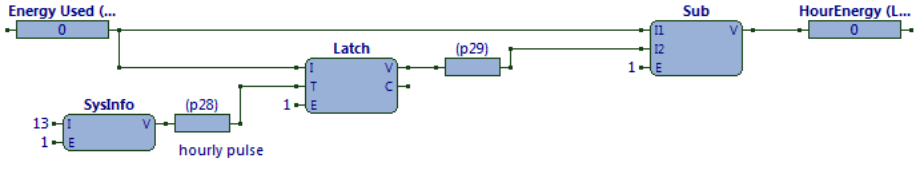


Fig. 35 Latch module example

The ObVerse strategy (Fig. 35) calculates the amount of energy used in the current hour. It latches the total energy used on the hour, and subtracts that latched value from the current total energy, to give the amount of energy used since the latch occurred.

The Latch module’s input values, I and T, are provided by the Energy Used property and a private property that is set to ‘1’ every hour. The output, V, is stored in a private property, which is then subtracted from the current Energy Used to determine the usage. Enable is set to ‘1’ (Yes).

Some external task writes the current totalised energy use (say kWh reading from a meter) to the Energy Used property. As the first hourly pulse occurs, the Latch module captures the reading at that point. From then on, the Subtract module subtracts the latched value from the latest value, and stores this value in the HourEnergy property. When the next hour pulse occurs, the Latch module captures the current value again. The process repeats every hour.

Related Modules

Usage-Over-Period

Availability

Available in standard and advanced processor versions dated April 2015 and later.

Lead-Lag

Object Type: *[Obv\LeadLag]*

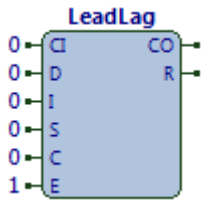


Fig. 36 Lead-Lag module

The LeadLag module (Fig. 36) performs a control operation to assist with enabling multiple items of plant that run in sequence – also known as a lead-lag configuration.

Each item of plant has its own LeadLag module, which is given an index. Link these modules together to form a circular chain. The modules are all passed a common start index (the lead item), and count of items to run.

The module determines when its item of plant should run based on its position in the chain of LeadLag modules. A module can be set to maintenance mode (e.g. on a fault condition), disabling the plant item's operation and passing operation to the next module in the chain.

When enabled, the operation is:

```
if S == I then
   if C then
      if D == 0 then
         R = 1
         CO = C - 1
      else
         R = 0
         CO = C
   else
      R = 0
      CO = 0
else
   if CI then
      if D == 0 then
         R = 1
         CO = CI - 1
      else
         R = 0
         CO = CI
   else
      R = 0
      CO = 0
```

The module contains the following objects:

Description	Reference	Type
Chain In Link this, via a property, to Chain Out from the previous LeadLag module. All LeadLag modules controlling a group of plant must be linked in a cyclic chain.	CI	Obj\Num; Adjustable; Default value: 0
Maintenance Temporarily disables the plant linked to this module from running. Can be used as fault (trip) or maintenance input	D	Obj\NoYes; Adjustable; Default value: 0 (No)
Index Index number of this LeadLag module, in the chain of modules. Index modules sequentially, starting from 0, in the order they will be operated.	I	Obj\Num; Adjustable; Default value: 0

Description	Reference	Type
Start The index number of the lead item of plant to run. Set this value periodically to ensure even wear of the plant	S	Obj\Num; Adjustable; Default value: 0
Count The number of plant items to run. Set this based on demand	C	Obj\Num; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs are left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Chain Out Output to next LeadLag module in the loop	CO	Obj\Num
Run Output set to '1' (Yes) when the device linked to this module needs to run	R	Obj\NoYes

Example

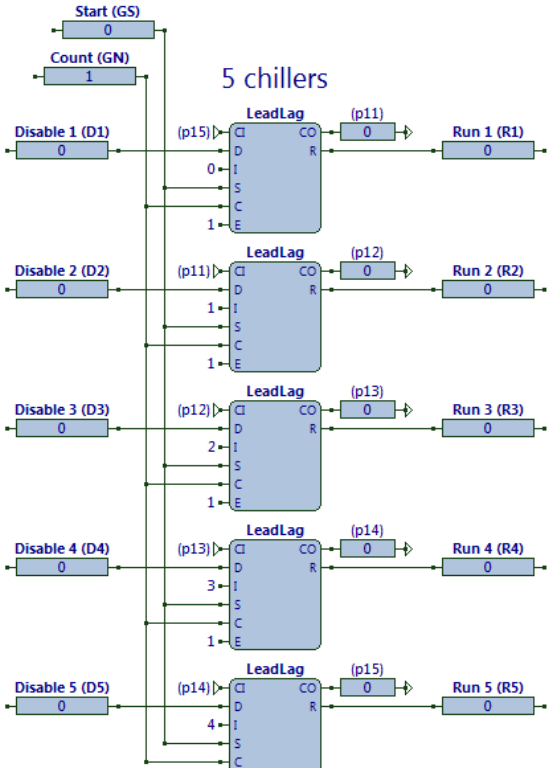


Fig. 37 Lead-Lag module example

The ObVerse strategy (Fig. 37) shows the lead-lag configuration of five chillers. The lead chiller is specified along with a count of chillers to run. Collectively, the LeadLag modules decide which chillers are run, and which are placed in standby. If a running chiller is disabled, maybe due to a fault, then it is stopped and the next chiller in the chain is started.

Each LeadLag module's CO output is linked to the next module's CI input via a private property, forming a circular chain. The modules are numbered sequentially, set using input I, using constants in the range 0…4. The Start and Count properties are linked to each LeadLag module's S and C inputs. Each LeadLag module has its input D is connected to a Disable property. Each Enable is set to '1' (Yes).

Some external task determines the start chiller (sometimes called the Lead) and the number of chillers needed, and writes these into the Start and Count properties. Some external task determines the operational (or fault) state of each chiller, and writes these into the Disable X properties.

If the Start property is set to '0', only the LeadLag module with its I input set to '0' starts the sequence calculation. The module determines whether its chiller can run, and if so, sets its output R to '1', decrements the count of chillers needed to determine the 'number of chillers still needed', and passes this value to the next LeadLag module via the CO-CI link. If its module cannot run, because its D input is '0', it passes the count of chillers needed to the next LeadLag module.

A LeadLag module with an I input that doesn't match the Start property works instead from its CI input, determines whether its chiller is needed and available, decrements the count if necessary, and passes this value to the next LeadLag module in the chain. This continues until the value passed becomes '0' or works all the way around the chain.

Availability

Available in standard and advanced processor versions dated April 2015 and later.

Less

Object Type: *[Obv\Less]*

The Less module (Fig. 38) performs the logic operation to determine whether one input is less than the other.

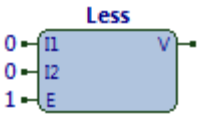


Fig. 38 Less module

When enabled, the formula is:

```
if I1 < I2 then
   V = 1
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input x to compare, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1
Value Set to '1' (Yes) if Input 1 is less than Input 2	V	Obj\NoYes

Example

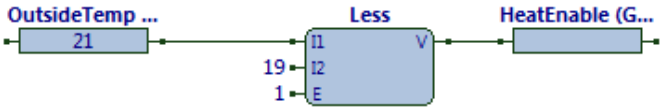


Fig. 39 Less module example

The ObVerse strategy (Fig. 39) shows the outside air temperature used to enable heating. If the temperature is less than 19, then heating is enabled.

The Less module's input I1, is provided by a linked property. Input I2, is set to a constant '19'. The output writes the result to property HeatEnable. Enable is set to '1' (Yes).

Some external task sets the OutsideTemp value. If the OutsideTemp is set to '21', the HeatEnable value will be '0' (No). If the OutsideTemp value is '16', the HeatEnable value will be '1'.

Related Modules

Greater, *Equal, Minimum*

Linearize

Object Type: *[Obv\Linearize]*

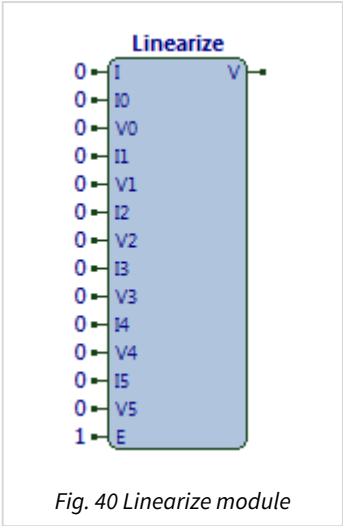


Fig. 40 Linearize module

The Linearize module (Fig. 40) performs the maths operation to linearize a non-linear system using a set of linear ranges.

Using a list of input-to-value points, the module determines which pair of input points the input lies between, and uses the corresponding value points to determine an approximate output value.

If the input is below Input Point 0 (I0), or equal to or above the Input Point 5 (I5), the output from this module is set to '0'. This allows several Linearize modules to be summed, to produce linearization over more than six input-to-value points.

When enabled, the operation is:

```
if (I >= I0) and (I < I1) then
   V = V1 – (V1 - V0) x (I1 – I) / (I1 – I0)
else if (I >= I1) and (I < I2) then
   V = V2 – (V2 – V1) x (I2 – I) / (I2 – I1)
else if (I >= I2) and (I < I3) then
   V = V3 – (V3 – V2) x (I3 – I) / (I3 – I2)
else if (I >= I3) and (I < I4) then
   V = V4 – (V4 – V3) x (I4 – I) / (I4 – I3)
else if (I >= I4) and (I < I5) then
   V = V5 – (V5 – V4) x (I5 – I) / (I5 – I4)
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input Input to convert	I	Obj\Float; Adjustable; Default value: 0
Input Point *x* Input value for input-to-value point *x*, where *x* is in the range 0..5	I*x*	Obj\Float; Adjustable Default value: 0
Value Point *x* Value output for input-to-value point *x*, where *x* is in the range 0..5	V*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to the result of the conversion	V	Obj\Float

Example

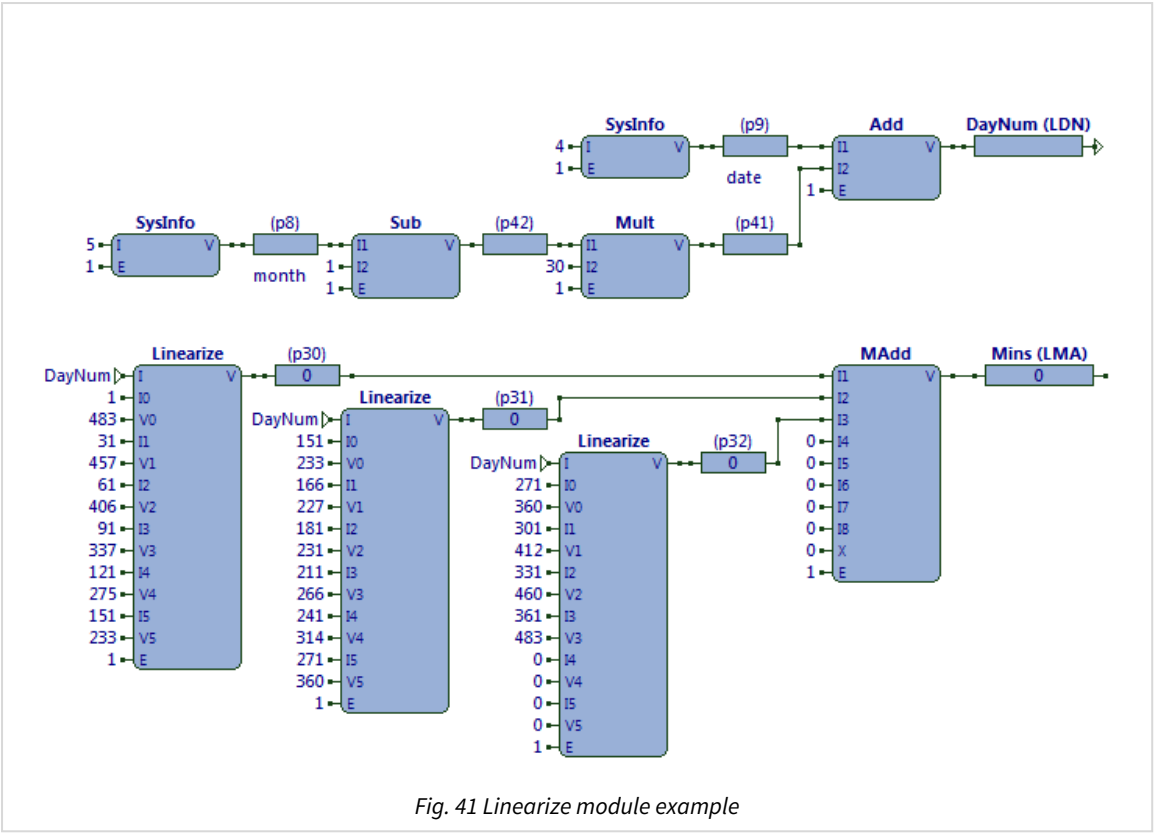


Fig. 41 Linearize module example

The ObVerse strategy (Fig. 41) calculates today's approximate sunrise time (in minutes). The strategy first calculates an approximate day-of-year (because it assumes months have 30 days). Sunrise changes non-linearly depending on the position of the sun and the location's latitude. The table below shows the sunrise time sampled every 30 days throughout the year in London. The strategy uses this data to approximate the sunrise time between these days, for the current date.

Day of Year	Sunrise Time (mins)
1	483
31	457
61	406
91	337
121	275
151	233
166	227
181	231
211	266
241	314
271	360
301	412
331	460
365	483

The Linearize modules input points, I*x*, are set with the day of year, and the value points, V*x*, are set with the sunrise time from the table above. The input I, is provided by the DayNum property. Multiple Linearize modules are used: only one will output a non-zero value, so their outputs are added together.

The strategy is approximate.

Related Modules

Rescale

Availability

Available in standard and advanced processor versions dated September 2012 and later.

Logical-And

Object Type: *[Obv\LAnd]*

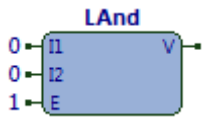


Fig. 42 Logical-And module

The LAnd module (Fig. 42) performs the logic operation to determine whether both of its inputs are set to a non-zero value. It performs the logical and operation (AND).

When enabled, the formula is:

```
if I1 and I2 then
  V = 1
else
  V = 0
```

I1	I2	V
0	0	0
0	1	0
1	0	0
1	1	1

The module contains the following objects:

Description	Reference	Type
Input *x* Inputs to AND together, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set '1' (Yes) when both Input 1 and Input 2 are non-zero, otherwise set '0' (No)	V	Obj\NoYes

Example

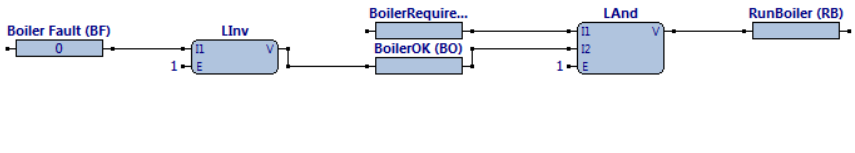


Fig. 43 Logical-And module example

The ObVerse strategy (Fig. 43) determines whether the boiler is healthy (by inverting a boiler fault), before logically AND'ing this with the boiler required state, to determine whether to run the boiler.

Some external task will write the current state of the boiler fault signal into Boiler Fault property, which is inverted before the LAnd module performs its calculation. The module writes the output value '1' if both inputs are '1'; otherwise it sets the value to '0'.

Related Modules

Multiple-Logical-And, Logical-Or, *Logical-Exclusive-Or*

Logical-Inverse

Object Type: *[Obv\LInv]*

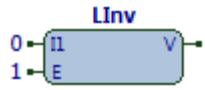


Fig. 44 Logical-Inverse module

The LInv module (Fig. 44) performs the logic operation to invert its input – a zero '0' value becomes '1' (Yes), and a non-zero value becomes '0' (No).

When enabled, the formula is:

```
if I1 then
   V = 0
else
   V = 1
```

The module contains the following objects:

Description	Reference	Type
Input Input to invert	I1	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '0' when Input is non-zero, and '0' when Input is zero	V	Obj\NoYes

Example

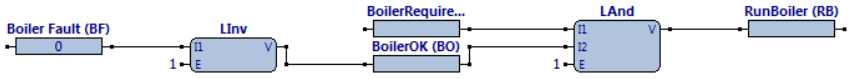


Fig. 45 Logical-Inverse module example

The ObVerse strategy (Fig. 43) determines whether the boiler is healthy (by inverting a boiler fault), before logically AND'ing this with the boiler required state, to determine whether to run the boiler.

Some external task will write the current state of the boiler fault signal into Boiler Fault property, which is inverted by the LInv module, before passing the value to the BoilerOk property. If Boiler Fault is '0' the LInv module sets the BoilerOk property to '1', otherwise it sets the property to '0'. The LAnd module writes the output value '1' if both inputs are '1'; otherwise it sets the value to '0'.

Logical-Exclusive-Or

Object Type: *[Obv\LXor]*

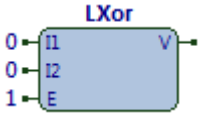


Fig. 46 Logical-Exclusive-Or module

The LXor module (Fig. 46) performs the logic operation to determine whether its inputs differ – i.e. one input is non-zero, and one input is zero. It performs the logical exclusive-or operation (XOR).

When enabled, the formula is:

```
if ((I1 == 0) and I2) or (I1 and (I2 == 0)) then
   V = 1
else
   V = 0
```

I1	I2	V
0	0	0
0	1	1
1	0	1
1	1	0

The module contains the following objects:

Description	Reference	Type
Input *x* Inputs to XOR together, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '1' if one Input is non-zero and one input is zero	V	Obj\NoYes

Example

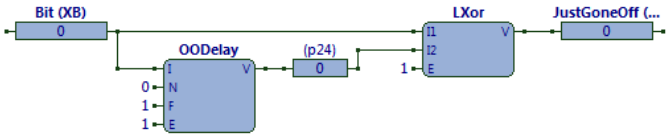


Fig. 47 Logical-Exclusive-Or module example

The ObVerse strategy (Fig. 47) determines when a bit value changes from non-zero to zero. The JustGoneOff property will stay set for one second – the off delay of the OODelay module.

After being '0' for some period, an external task sets the Bit property to '1'. The OODelay module will output '1' immediately, so both inputs to the LXor module are the same and the LXor module writes '0' to JustGoneOff property. However, when the Bit property value changes back to '0', the OODelay output remains at '0' for 1 second: during this 1 second period, the inputs to the LXor are different, so it writes '1' to the JustGoneOff property.

Related Modules

Multiple-Logical-Or, Logical-And

Logical-Or

Object Type: *[Obv\LOr]*

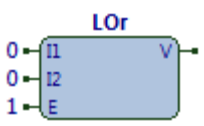


Fig. 48 Logical-Or module

The LOr module (Fig. 48) performs the logic operation to determine whether either of its specified inputs are set non-zero. It performs the logical or operation (OR).

When enabled, the formula is:

```
if I1 or I2 then
  V = 1
else
  V = 0
```

I1	I2	V
0	0	0
0	1	1
1	0	1
1	1	1

The module contains the following objects:

Description	Reference	Type
Input *x* Input x to OR together, where *x* is in the range 1..2	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '1' if either Input 1 or Input 2 is non-zero, otherwise set '0'	V	Obj\NoYes

Example

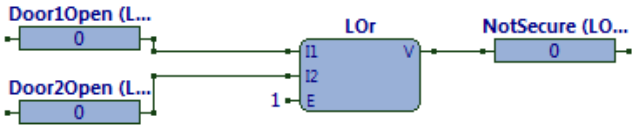


Fig. 49 Logical-Or module example

The ObVerse strategy (Fig. 49) examines the open-states of two doors to the same area – if either is open, the room is not secure. If both are open, the room is not secure.

Related Modules

Multiple-Logical-Or, Logical-And, *Logical-Exclusive-Or*

Maximum

Object Type: *[Obv\Max]*

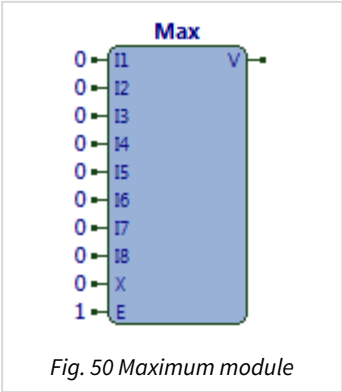


Fig. 50 Maximum module

The Max module (Fig. 50) performs the maths operation to find the maximum of up to eight numbers.

When enabled, the operation is:

V = Maximum(**I1, I2, I3,** … I_X)

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of Inputs (starting from I1) to include in calculation	X	Obj\Num; Range 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

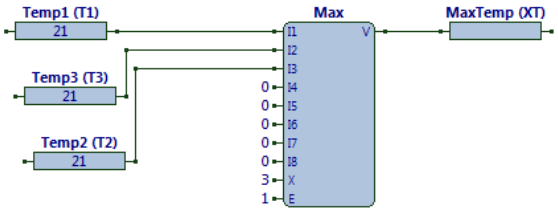


Fig. 51 Maximum module example

The ObVerse strategy (Fig. 51) determines the maximum temperature from three sensor values

The Max module's three input values, I1 to I3, are read from the properties Temp1, 2, and 3. Input X is set to a constant of '3'. The output V is written to the MaxTemp property. Enable is set to '1' (Yes).

Some external task writes into the Temp properties. If the input values were set to '21', '22', and '23' the module would write the value '23' to MaxTemp.

Remember it is possible to set Zip inputs so that they give an override value whenever a sensor is out-of-limits.

Related Modules

Minimum, *Average*

Minimum

Object Type: *[Obv\Min]*

The Min module (Fig. 52) performs the maths operation to find the minimum of up to eight values.

When enabled, the operation is:

V = Minimum(**I1, I2, I3,** … I_X)

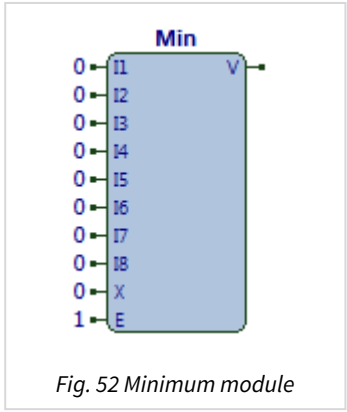


Fig. 52 Minimum module

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of Inputs (starting from I1) to include in calculation	X	Obj\Num; Range 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

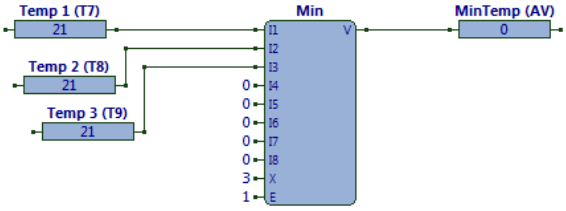


Fig. 53 Minimum module example

The ObVerse strategy (Fig. 53) determines the minimum temperature from three sensor values

The Min module's three input values, I1 to I3, are read from the properties Temp 1, 2, and 3. Input X is set to '3'. The module calculates the minimum of the values, and its output, V, writes the result to the MinTemp property. Enable is set to '1' (Yes).

Some external task write values into the Temp1, 2, and 3 properties. If the input values were set to '21', '22', and '23', the modules would write the value '21' to the MinTemp property.

Remember it is possible to set Zip inputs so that they give an override value whenever a sensor is out-of-limits.

Related Modules

Maximum, *Average*

Modulus-Remainder

Object Type: *[Obv\Mod]*

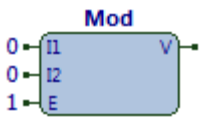


Fig. 54 Modulus-Remainder module

The Mod module (Fig. 54) performs the maths operation to divide one input by the other to calculate the remainder, or modulus.

When enabled, the formula is:

```
V = Remainder of (I1 / I2)
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..2 Input 1 is the dividend, Input 2 is the divisor	I*x*	Obj\Num; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value, the remainder	V	Obj\Num

Example

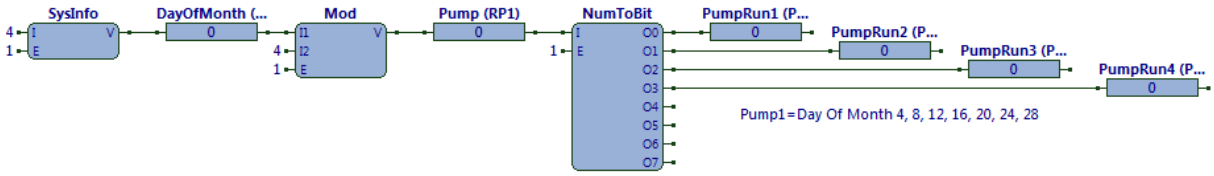


Fig. 55 Modulus-Remainder module example

The ObVerse strategy (Fig. 55) determines the day-of-month, and uses that to decide which of four pumps to run.

The SysInfo module write the day-of-month into a property. The Mod module divides the day-of-month by 4; the remainder value in the range 0…3 is used to enable one of four pumps.

PumpRun1 is set to '1' on day-of-month 4, 8, 12, 16, 20, 24, and 28. PumpRun2 is set to '1' on day-of-month 1, 5, 9, 13, 17, 21, 25, and 29. PumpRun3 is set to '1' on day-of-month 2, 6, 10, 14, 18, 22, 26, and 30. PumpRun4 is set to '1' on day-of-month 3, 7, 11, 15, 19, 23, 27, and 31.

Related Modules

Multiply, *Divide*

Multiple-Add

Object Type: *[Obv\MAdd]*

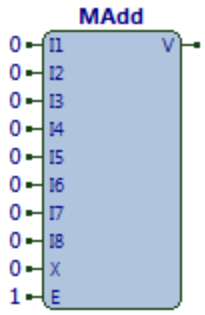


Fig. 56 Multiple-Add module

The MAdd module (Fig. 56) performs the maths operation to add up to eight numbers together.

When enabled, the formula is:

$$V = (I1 + I2 + I3 + \ldots + I_X)$$

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of Inputs (starting from Input 1) to include in calculation	X	Obj\Num; Range: 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

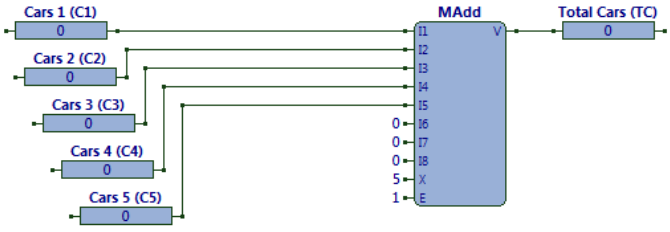


Fig. 57 Multiple-Add module example

The ObVerse strategy (Fig. 57) adds together five Cars properties, to determine the Total Cars parked.

The MAdd module's five inputs, I1 to I5, are read from properties Cars 1, 2, 3, 4, and 5. The MAdd modules adds these together (its X input is set to the number of inputs to add). The output V writes the result to the property TotalCars. Enable is set to '1' (Yes).

Some external task writes values into the Cars properties. If the five values are set to '0', '56', '23', '108', and '12', the value written to Total Cars will be '199'.

Related Modules

Add, *Subtract*

Multiple-Logical-And

Object Type: *[Obv\MAnd]*

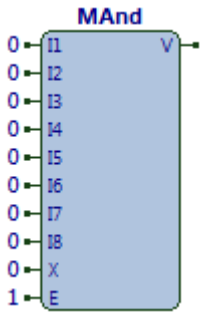


Fig. 58 Multiple-Logical-And module

The MAnd module (Fig. 58) performs the logic operation to determine whether all of its specified inputs are set to a non-zero value. It performs the logical and operation (AND).

When enabled, the formula is:

```
if (I1 and I2 and I3 .. and Ix) then
   V = 1
else
   V = 0
```

I1	I2	I3	I4	I5	I6	I7	I8	V
0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0
1	0	1	0	1	0	0	1	0
1	1	1	1	1	1	1	1	1

The module contains the following objects:

Description	Reference	Type
Input *x* Input to AND together, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of Inputs (starting from Input 1) to include in calculation	X	Obj\Num; Range 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '1' (Yes) when all required inputs are all non-zero, otherwise set to '0' (No)	V	Obj\NoYes

Example

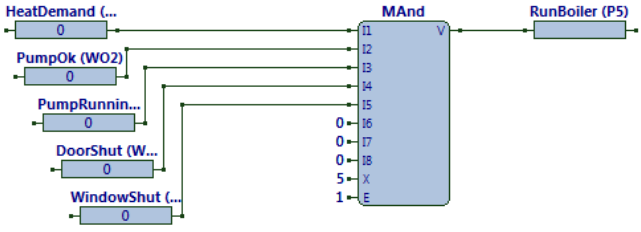


Fig. 59 Multiple-Logical-And module example

The ObVerse strategy (Fig. 59) determines whether all requirements have been met before running a boiler.

Some external tasks write values into the HeatDemand, PumpOk, PumpRunning, DoorShut, and WindowShow properties. The MAnd module writes the value '1' to RunBoiler only if all five inputs are '1'.

Related Modules

Multiple-Logical-Or, Logical-Or, *Logical-Exclusive-Or*

Multiple-Logical-Or

Object Type: *[Obv\MOr]*

The MOr module (Fig. 60) performs the logic operation to determine whether any of its specified inputs are set non-zero. It performs the logical or operation (OR).

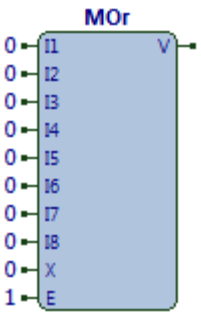


Fig. 60 Multiple-Logical-Or module

When enabled, the formula is:

```
if (I1 or I2 or I3 .. or Ix) then
   V = 1
else
   V = 0
```

I1	I2	I3	I4	I5	I6	I7	I8	V
0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1
1	0	1	0	1	0	0	1	1
1	1	1	1	1	1	1	1	1

The module contains the following objects:

Description	Reference	Type
Input *x* Input to OR together, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Num Inputs Number of Inputs (starting from Input 1) to include in calculation	X	Obj\Num; Range 0..8; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to '1' (Yes) if any of the required inputs is non-zero, otherwise set to '0' (No)	V	Obj\NoYes

Example

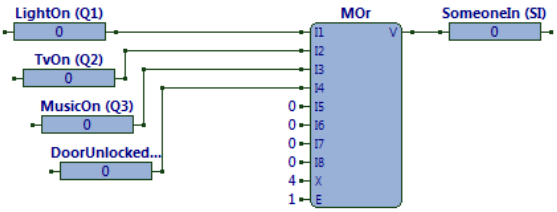


Fig. 61 Multiple-Logical-Or module example

The ObVerse strategy (Fig. 61) uses binary states from a range of systems to determine whether something is occupied.

Some external tasks write the LightOn, TvOn, MusicOn, and DoorUnlocked properties. The MOr module writes the value '1' to the SomeoneIn property if any of the four inputs are '1'; otherwise, it writes '0'.

Related Modules

Multiple-Logical-And, Logical-Or, *Logical-Exclusive-Or*

Multiply

Object Type: *[Obv\Mult]*

The Mult module (Fig. 62) performs the maths operation to multiply two numbers.

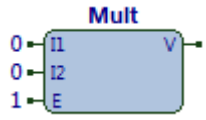


Fig. 62 Multiply module

When enabled, the formula is:

V = I1 x I2

The module contains the following objects:

Description	Reference	Type
Input *x* Inputs to include in calculation, where *x* is in the range 1..2 Input 1 is the multiplier, Input 2 is the multiplicand	I*x*	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value, the product	V	Obj\Float

Example

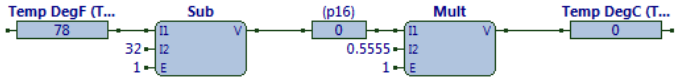


Fig. 63 Multiply module example

The ObVerse strategy (Fig. 63) converts the temperature in degrees Fahrenheit to degrees Celsius, by using the formula:

°C = (°F - 32) x (5 / 9)

Some external tasks writes a temperature value (in degrees Fahrenheit) into the Temp DegF property. The Sub module subtracts 32 from this, and writes the result to the private property. The Mult module reads the private property, multiplies it by 0.5555 (5/9), and writes the final result to the Temp DegC property

Related Modules

Divide, *Modulus-Remainder*

Num-To-Bit

Object Type: *[Obv\NumToBit]*

The NumToBit module (Fig. 64) performs the maths operation to set one output state to '1' (Yes), corresponding to the input value. All other outputs are set to '0'.

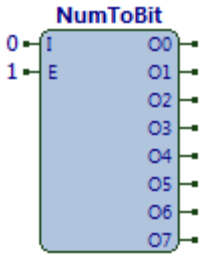


Fig. 64 Num-To-Bit module

When enabled, the formula is:

```
if I == 0 then
   O0 = 1
else if I == 1 then
   O1 = 1
else if I == 2 then
   O2 = 2
else if I == 3 then
   O3 = 1
else if I == 4 then
   O4 = 1
else if I == 5 then
   O5 = 1
else if I == 6 then
   O6 = 1
else if I == 7 then
   O7 = 1
```

The module contains the following objects:

Description	Reference	Type
Num Input to include in calculation. Sets the corresponding output Num to '1' (Yes)	I	Obj\Num: 0…7; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs are left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Num Is *x* The output number, *x*, is in the range 0…7. One output is set '1' (Yes) depending on the input Num. Other outputs are set '0' (No)	O*x*	Obj\NoYes

Example

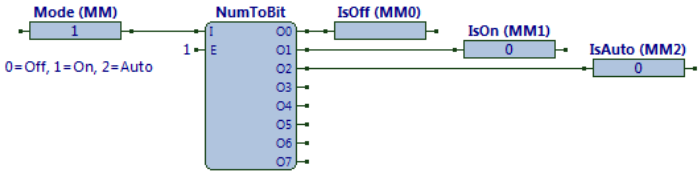


Fig. 65 Num-To-Bit module example

The ObVerse strategy (Fig. 65) shows a user-adjustable Mode parameter ('Off', 'On', or 'Auto') converted to individual states for use by the strategy.

The NumToBit module input value I is read from the Mode property. The outputs, O0 to O2, pass the result to the linked properties. Enable is set to '1' (Yes).

Some external task, perhaps a user via a display, writes a value to the Mode property ('0'=Off, '1'=On, and '2'=Auto). The NumToBit writes a '1' (Yes) to the corresponding property IsOff, IsOn, or IsAuto, and sets the others to '0' (No).

Related Modules

Bit-To-Num, *Byte-To-Bits*, *Bits-To-Byte*

Availability

Available in standard and advanced processor versions dated September 2012 and later.

Object-Read

Object Type: *[Obv\ObjRead]*

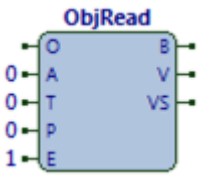


Fig. 66 Object-Read module

The ObjRead module (Fig. 66) performs the remote object operation to read the value of an object from a task external to the processor.

The module performs its object-read operation only when triggered. Link the trigger input to a pulser or system-information module.

When enabled, the operation is:

```
V = ObjRead(O, A, T, P)
```

When the module is triggered, it is added to the processors list of remote object tasks. This allows one Pulser or System-Information module to trigger multiple Object-Read modules. Remember to allow a pulse time long enough for all modules to complete their action.

The module can use the remote object as either an absolute reference or relative reference. In absolute mode, the object requested is specified by the remote object reference. In relative mode, if the property O is present within the ObVerse strategy, then the remote object reference will be prefixed with this remote object prefix (see *Reserved References*). If the property O does not exist, or is blank, then the absolute input has no effect.

The module contains the following objects:

Description	Reference	Type
Remote Object The reference of the object to read, either relative to the object of the process, or an absolute reference	O	Obj\Obj; Adjustable Default value: ‘’
Absolute When set ‘0’ (No) the object requested will be relative to the ObVerse O property, when present. When set ‘1’ (Yes) the Remote Object will be requested as is.	A	Obj\NoYes; Adjustable Default value: 0 (No)
Trigger When this value changes from ‘0’ (No) to ‘1’ (Yes), the object-read operation is triggered. Typically, this is linked to a pulser or system-information module.	T	Obj\NoYes; Adjustable Default value: 0 (No)
Priority Action Be default, all remote object access operations are performed in order within the ObVerse. If set ‘1’ (Yes), this module’s object-read is performed before others.	P	Obj\NoYes; Adjustable Default value: 0 (No)
Enable Enables the module’s operation. If set to ‘0’ (No), then no operation occurs	E	Obj\NoYes; Adjustable Default value: 1
Busy Set ‘1’ (Yes) when the module is triggered to read the Remote Object, and set ‘0’ (No) when the module completes the operation.	B	Obj\NoYes
Value Contains the last value read by the module	V	Obj\Text; Max chars: 31

Description	Reference	Type
Value Set Set '1' (Yes) when the value has been read successfully. Set '0' (No) when the value fails to read (after two attempts), and on initialisation	VS	Obj\NoYes

Example

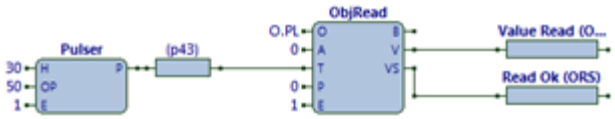


Fig. 67 Object-Read module example

The ObVerse strategy (Fig. 67) periodically reads an object reference from an external task.

The ObjRead module has input O set to 'O.PL', specifying the object reference to read. Input A is set '0', indicating a relative reference – so the object would be prefixed by property O in the ObVerse, if present. Input T is linked to a Pulser module, which will write '0' and '1' alternately to the private property (cycling every 30 seconds), which is used as a trigger by the ObjRead module.

When the reading is triggered, the output B writes a value of '1' to its connected property (none in the example), and the read operation is added to the processor's list of remote object tasks. When the processor completes this read operation, the module's output B is set back to '0', and outputs V and VS are set. If the read was successful, the output value, V, writes the received value to the linked property Value Read and sets output VS to '1'. If the read fails after two attempts, perhaps because the remote object is not available, then output V is left unchanged, and output VS is writes a value of '0' to the Read Ok property.

Related Modules

Object-Write

Object-Write

Object Type: *[Obv\ObjWrite]*

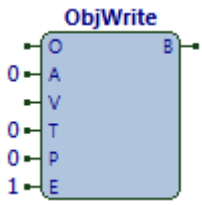


Fig. 68 Object-Write module

The ObjWrite module (Fig. 68) performs the remote object operation to write the value of an object to a task external to the processor.

The module performs its object-write operation either when triggered, or when the value changes when the trigger input is '1'. Link the trigger input to a system-information or pulser module, or set it to a constant '1'.

When enabled, the operation is:

```
B = ObjWrite(O, A, V, T, P)
```

The module can use the remote object as either an absolute reference or relative reference.

In absolute mode, the object requested is specified by the remote object reference. In relative mode, if the property O is present within the ObVerse strategy, then the remote object reference will be prefixed with this remote object prefix (see *Reserved References*). If the property O does not exist, or is set to blank, the absolute input has no effect.

The module contains the following objects:

Description	Reference	Type
Remote Object The reference of the object to write, either relative to the object of the process, or an absolute reference	O	Obj\Obj; Adjustable; Default value: ''
Absolute When set '0' (No) the object requested will be relative to the ObVerse O property, when present. When set '1' (Yes) the Remote Object will be requested as is.	A	Obj\NoYes; Adjustable Default value: 0 (No)
Value The value to write to the Remote Object. If Trigger is set when this value changes, the write operation is triggered	V	Obj\Text; Adjustable; Max chars :31 Default value: ''
Trigger When this value changes from '0' (No) to '1' (Yes), the object-write operation is triggered. When set '1' (Yes), the object-write also occurs when the Value changes. When set '0' (No), no object-write is performed. Typically, this is linked to a system-information or pulser module.	T	Obj\NoYes; Adjustable Default value: 0 (No)
Priority Action Be default, all remote object access operations are performed in order within the ObVerse. If set '1' (Yes), this module's object-write is performed before others.	P	Obj\NoYes; Adjustable Default value: 0 (No)
Enable Enables the module's operation. If set to '0' (No), then no operation occurs	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)

Description	Reference	Type
Busy Set '1' (Yes) when the module is triggered to write to the Remote Object, and set '0' (No) when the module completes the operation.	B	Obj\NoYes

Example

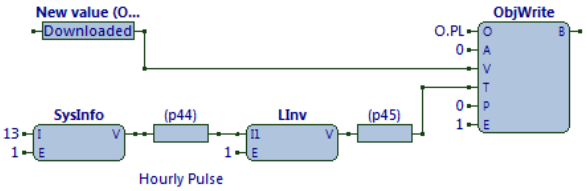


Fig. 69 Object-Write module example

The ObVerse strategy (Fig. 69) writes a value, when changed and periodically, to an object reference within an external task.

The ObjWrite module has input O set to 'O.PL', specifying the object reference to write. Input A is set '0', indicating a relative reference – so the object would be prefixed by property O in the ObVerse strategy, if present. Input T is linked to an inverted hourly pulse, which is usually set to '1' (meaning the value will write whenever it changes). Every hour, on the hour, the input T will change to '0' then return to '1', again triggering the write operation –as a background re-write.

When the module is triggered (either by the trigger changing to '1', or the value changing when the trigger is '1'), output B is set to '1' and the operation is added to the processor's list of remote object tasks. When the processor completes it write operation, the module writes '0' via output B (not connected in the example).

Related Modules

Object-Read

On-Off-Delay

Object Type: *[Obv\OODelay]*

The OODelay module (Fig. 70) performs the timer operation to delay a digital input state, before setting the output value to the same state. An on and off delay time are specified in seconds.

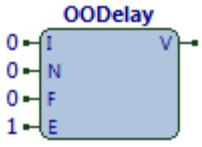


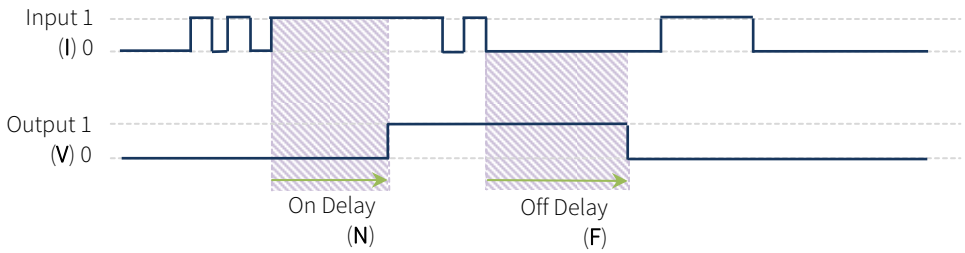
Fig. 70 On-Off-Delay module

The input must remain at the '1' (On) state for at least the On Delay time before the output value is set to '1'. Similarly, the input must remain at the '0' (Off) state for at least the Off Delay time before the output value is set to '0'. Otherwise the output is left unchanged.

Either or both of the delay times can be set to '0', to remove the validation timer.

When enabled, the operation is:

```
onChange(I) and I == 1 then
   if I == 1 for N seconds
      V = I
onChange(I) and I == 0 then
   if I == 0 for F seconds
      V = I
```



The module contains the following objects:

Description	Reference	Type
Input Input state to delay	I	Obj\OffOn; Adjustable; Default value: 0 (Off)
On Delay (s) Number of seconds the input state must remain '1' (On) before setting the Output Value to '1'	N	Obj\Float; Adjustable; Default value: 0
Off Delay (s) Number of seconds the input state must remain '0' (Off) before setting the Output Value to '0'	F	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (No)
Output Value The delayed input state	V	Obj\OffOn

Example

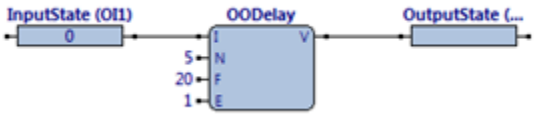


Fig. 71 On-Off-Delay module example

The ObVerse strategy (Fig. 71) ensures that the input state remains '1' (On) for a minimum of 5 seconds before the output value is set to '1' (On). However, it checks that the input state is '0' (Off) for at least 20 seconds before setting the output value '0' (Off).

Some external task writes a state into the InputState property. If the value changes to '1', the OODelay waits for 5 seconds before doing anything: if the InputState remains at '1' for 5 seconds (the on delay), the OODelay writes a value '1' to the OutputState property; if the InputState property changes back to '0' within 5 seconds, the OODelay does not change the OutputState property. Once stabilised in the '1' state, a similar thing happens when the InputState value changes to '0': the OODelay waits for the 'off' delay before writing '0' to the OutputState property.

Sometimes this function is used to 'debounce' an input: if the input flips on and off temporarily when generally changing between states; or if a temperature fluctuates around an alarm level crossing back and forth as the value slowly rises.

Optimum-Start-Stop

Object Type: *[Obv\OSS]*

The OSS module (Fig. 72) performs the control operation to determine the optimum time to start and stop plant in order to achieve a zone setpoint within the times specified.

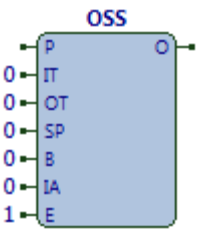


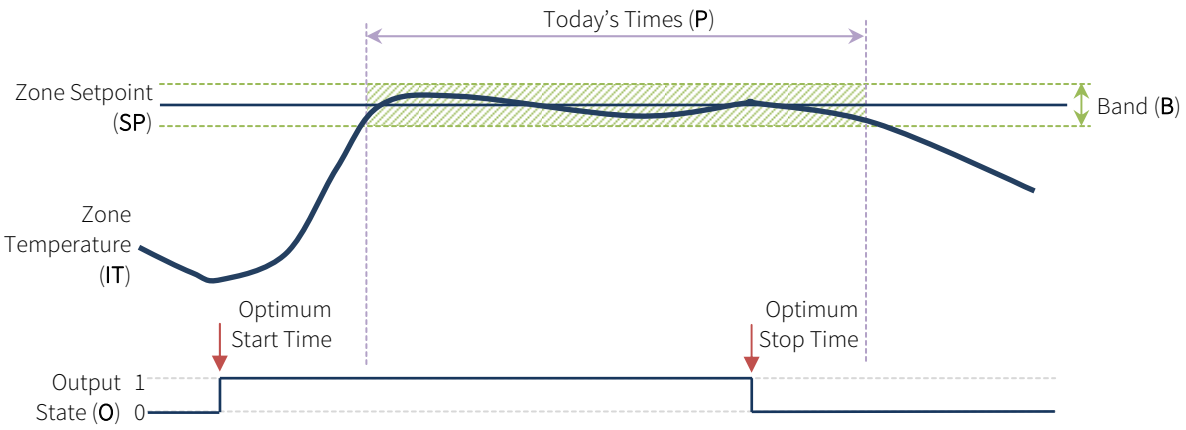
Fig. 72 Optimum-Stop-Start module

By monitoring the zone temperature after heating starts, the module learns how quickly the temperature rises (allowing for heat loss to the outside). Similarly, by monitoring the zone temperature after plant stops, the module learns how quickly the zone loses heat to the outside. The module then uses these learned responses to offset start and stop times in future. The module will continue to learn after the first start-stop cycle.

Optimum-start-stop cannot achieve perfect results, due to factors outside of its control: difference in solar gain, windows and doors being left in different positions, wind direction or speed, or differing rate of rise and fall of outside temperature.

When enabled, the operation is:

```
O = Oss(P, IT, OT, SP, B, IA)
```



The module has a simple model of the heat loss and gain. It assumes the heat loss is proportional to the difference between zone temperature and outside temperature. It calculates a heat loss factor during the 'heating off' stage. During 'heating on' stage, the module uses its current heat loss factor along with temperature gains to calculate a heat gain factor. These factors are used on subsequent starts and stops.

During optimum start, if the zone temperature is not within a band of variance by today's required on-time, the module learns to start earlier. If the zone temperature reaches this band early, it needs to start later. During optimum stop, if the zone temperature drops below the band of variance before today's required off-time, the module learns to stop later. If the zone temperature is still within the band, it needs to stop earlier.

The module contains the following objects:

Description	Reference	Type
Today's Times On-Off times to try to control to zone setpoint	P	Obj\Times or Obj\Text; Adjustable; On-off periods: 2 (6 in advanced processors) Default value: ''
Zone Temperature Temperature inside zone, to control to setpoint	IT	Obj\Float; Adjustable; Default value: 0

Description	Reference	Type
Outside Temperature Temperature outside of zone/building. The module uses this to calculate heat loss.	OT	Obj\Float; Adjustable; Default value: 0
Zone Setpoint Required temperature of zone during Today's Times	SP	Obj\Float; Adjustable; Default value: 0
Band The amount of variability in the setpoint that is acceptable. This is distributed evenly around the Zone Setpoint, 50% above and 50% below	B	Obj\Float; Adjustable; Default value: 0
Inverse Action Inverts module action, used to control cooling rather than heating.	IA	Obj\NoYes; Adjustable; Default value: 0 (No)
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and O is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Output State Set '1' (On) to indicate that heating should be enabled	O	Obj\OffOn

Example

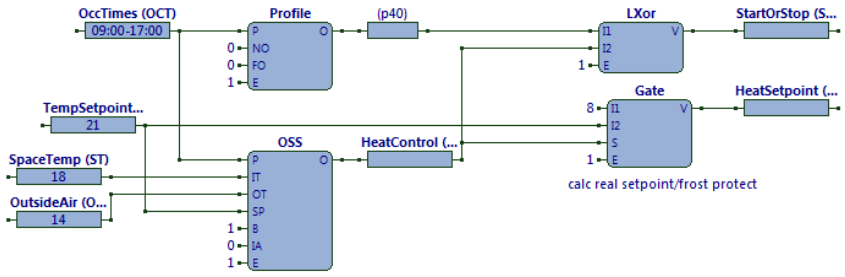


Fig. 73 Optimum-Stop-Start module example

The ObVerse strategy (Fig. 73) shows optimum-stop-start used to calculate when to enable heating based on today's occupancy times, room temperature setpoint, current room temperature, and outside air temperature. The occupancy times and setpoint are also used to calculate a heating setpoint, and when the system is in an optimum start or stop phase of operation.

The Oss module's P, IT, OT, and SP inputs are read from the OccTimes, TempSetpoint, SpaceTemp, and OutsideAir properties. The band, input B, is set to '1'. Enable is set to '1' (Yes).

Some external task writes the OccTimes and TempSetpoint properties. Some extern task will periodically write the SpaceTemp and OutsideAir properties. If the TempSetpoint is '21', then because the band to '1' the optimum-stop-start module allows a SpaceTemp in the range 20.5…21.5. If OccTimes is set to '09:00-17:00', the module will attempt to set HeatControl to '1' (On) in time for the room to reach this temperature range by 09:00. When the time is 09:00, the module will examine the SpaceTemp value to determine how it should modify its heat-gain factor: if SpaceTemp is low, it will lower the factor; otherwise, it will raise the heat-gain factor.

After starting is complete, the module will use the heat-loss factor to determine when to write a '0' to the HeatControl property. When the time is 17:00, the module will examine the current SpaceTemp value to determine how it should modify its heat-loss factor: if SpaceTemp is low, it will increase the factor; otherwise, it will lower the heat-loss factor.

Based on the value of HeatControl, the Gate module outputs a HeatSetpoint of a constant ‘8’ or the current TempSetpoint: this is the setpoint for the heating system control elsewhere. The Profile module calculates whether the current time is within the OccTimes value. The LXor module uses the output of the Profile mode to determine whether the strategy is within optimum start or optimum stop periods, and if so writes ‘1’ to the StartOrStop property.

Related Modules

Profile

Profile

Object Type: *[Obv\Profile]*

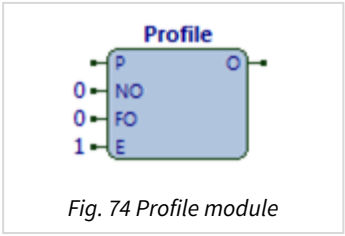


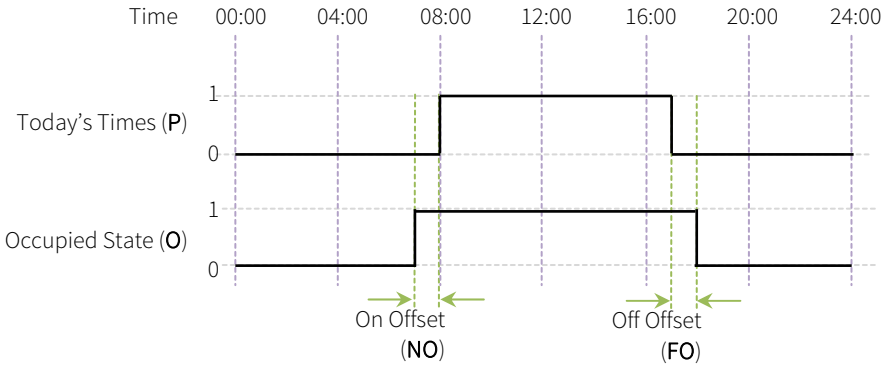
Fig. 74 Profile module

The Profile module (Fig. 74) performs the timer operation to determine the current occupied state from a set of on-off times, based on the device's current time.

If necessary, the module can offset the on and off times, forward or backwards.

When enabled, the operation is:

```
if (timeNow > (startTime(P) + NO)) and (timeNow < (endTime(P) + FO)) then
  O = 1
else
  O = 0
```



The module contains the following objects:

Description	Reference	Type
Today's Times On-Off times for today	P	Obj\Times or Obj\Text; Adjustable; On-off periods: 2 (6 in advanced processors) Default value: ''
On Offset (mins) Number of minutes to adjust the on time from Today's Times. Set a positive value to offset the on state to a later time, set negative for an earlier time	NO	Obj\Num: -300…300; Adjustable; Default value: 0
Off Offset (mins) Number of minutes to adjust the off time from Today's Times. Set a positive value to offset the off state to a later time, set negative for an earlier time	FO	Obj\Num: -300…300; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and O is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Occupied State The calculated state	O	Obj\OffOn

Example

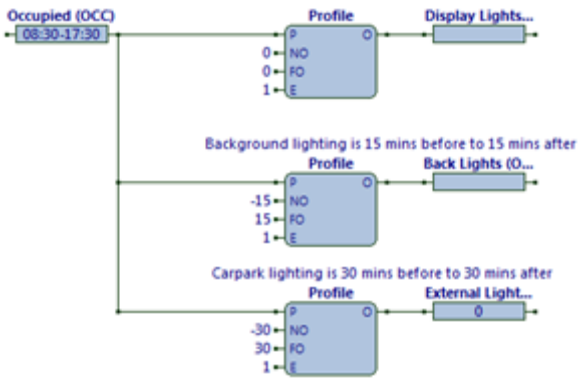


Fig. 75 Profile module example

The ObVerse strategy (Fig. 75) calculates when different types of lighting are turned on and off, from a single set of on-off times by using the offset capabilities of the Profile module.

Each Profile module read its P input from the Occupied property, which provides on-off times for today. Each profile writes its output a property. The enable is set to '1' (Yes)

Some external task writes a value to the Occupied property.

If it is set to '08:30-17:30', then because the first Profile module has its NO and FO inputs set to '0', the Profile module uses the current time to determine whether to write '0' or '1' to the Display Lights property – if the current time is between the 08:30 and 17:00, '1' (On) will be written. As the current time changes, the Profile module recalculate whether to write '0' or '1' to the Display Lights property.

The second Profile module has NO set to '-15' and FO to '15', which will write a value of '1' to Back Lights property if the current time is between 08:15 (08:30 - 15 minutes) and 17:15 (17:00 + 15 minutes).

The third Profile module will write a '1' to the External Lights property between 08:00 and 17:30.

Related Modules

Optimum-Start-Stop

Proportional-Integral-Derivative

Object Type: *[Obv\Pid]*

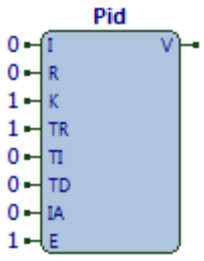


Fig. 76 Proportional-Integral-Derivative module

The PID module (Fig. 76) performs the control operation to vary an output value (percent) to try to make a measured input value (such as a temperature) match a required value (sometimes called a setpoint), in a closed-loop control system. It can perform proportional, proportional-integral, or proportional-integral-derivative control.

Proportional Control

The module determines the 'error' – how far the measured value is away from the required value – and multiplies this by a constant, called the gain K to produce a simple output percent. This means that as the error increases, the output increases, which causes more effort (for example heating) to be done; this subsequently should increase the measured value, therefore reduce the error, which it turn reduces the output percentage.

Rather than calculate this constantly when systems are very slow to change, a recalculation period is specified. The recalculation period depends on the response rate of the system being controlled, as well as how fast the output needs to be calculated – a building temperature heating system might use a recalculation time of 15 seconds or more. This is because the temperature changes are typically slow, or the output movements need to be kept to a minimal to reduce wear on, say, a valve.

If the gain is too high for the response speed of the system, the measured value keeps oscillating and gets worse over time – reduce the gain. The period of the oscillation is due to the time it takes between enabling the output and the system delivering the effort to the measured value, and the time between disabling the output and the system stopping the delivery of the effort. The 'tuning' requires time to monitor the loop output over several oscillation cycles, followed by adjustment, followed by more tuning.

Pure proportional control suffers if the measured value reaches the required value: the error is zero, therefore the output is zero, and therefore the effort (for example heating) becomes zero. This appears as the measured value stabilising (or gently oscillating) below the required value. One solution is to add a constant 'offset' to the required setpoint to make the system 'stabilise' around actual required value.

One solution to this 'adding an offset' problem is to have the loop itself try to calculate this – the integral

Proportional + Integral Control

This type of control works as proportional control, but also automatically adds small amounts to the output whilst an error exists. This effectively calculates and adds the offset (discussed above) slowly over time. The engineer defines the rate to add to the offset, as the time over which a whole gain is added to the output.

Ideally, the system should cycle a few times, and stabilise at the required value. However, the system may cycle continuously (or even wildly) due to the gain being too large, or the integral time being too short.

The gain of a proportional-only loop will need to be decreased when the integral part is enabled, otherwise too much oscillating occurs when gain and integral act together. Decreasing the gain slows down the overall response of the system, but it does achieve the requirement eventually.

One solution to this 'slowing down' problem: increase the output even more when the required value changes. This is the effect of derivative.

Proportional + Integral + Derivative Control

This type of control works as proportional + integral control, but also monitors the rate of change of the error, and increases the output as the rate of change of the error increases. This has the effect of a quick burst of extra output when the required value changes. The effect is specified as a period over which the burst should be applied – the longer the time, the more effect. If the effect becomes too much, it makes the output, and therefore the system, oscillate.

Derivative control is used less frequently, as it requires more tuning.

When enabled, the operation is:

```
V = Pid(I, R, K, TR, TI, TD, IA)
```

The module contains the following objects:

Description	Reference	Type
Input The measured value to control to the Required value	I	Obj\Float; Adjustable; Default value: 0
Required The value to attempt to control the Input to	R	Obj\Float; Adjustable; Default value: 0
Gain The gain – used to multiply the error by	K	Obj\Float; Adjustable; Default value: 1
Recalculation Time (s) The period, in seconds, between recalculating the output	TR	Obj\Num; Adjustable; Default value: 1
Integral Time (s) The period over which to add the equivalent of one gain to the output – set to '0' to disable integral control	TI	Obj\Num; Adjustable; Default value: 0
Derivative Time (s) The period over which to have a derivative effect equivalent to a whole gain – set to '0' to disable derivative control	TD	Obj\Num; Adjustable; Default value: 0
Inverse Action Inverts action, used to control cooling rather than heating.	IA	Obj\NoYes; Adjustable; Default value: 0 (No)
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and output V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Output (%) This indicates percent of heating (or cooling) in the range 0..100	V	Obj\Float

Example

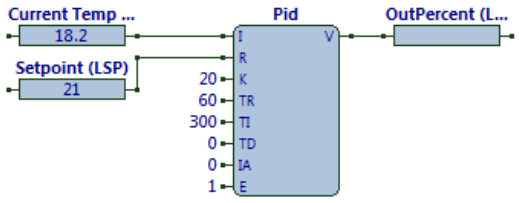


Fig. 77 Add module example

The ObVerse strategy (Fig. 77) calculates an output percentage, in the range '0'..'100', to control the heat supply. The Gain (K) is '20', which implies the proportional part of the output will give 100% output if the temperature is 5 degrees below the Setpoint, because Error x Gain = Output. The integral part of the output is quite fast, allowing 300 seconds to add another gain to the output – after 5 minutes another 20%, after 10 minutes another 20%, etc.

Some external task writes the Setpoint property. Some task periodically writes into the Current Temp. It is assumed this occurs every 10-20 seconds, because the module recalculates its output every 20 seconds. Every 20 seconds the Pid module calculates the error, multiplies this by the gain to produce a proportional element. It also calculates an internal integral element (which is incremented whilst an error exists). The proportional element and the integral element are added, and the value written to the OutPercent property.

If the initial Setpoint is '20' and the Current Temp is '18', the error is 2, the proportional element will be 40% and the integral element will be 0%, so the module will write '40' to the OutPercent property.

If after 50 seconds the Current Temp is '19.5' (a very fast heating system), the error is now 0.5, the proportional element is now 10, the integral element is now 50/300 of the gain, or 3.3, so the module writes '13.3' to the OutPercent property.

If after 100 seconds the Current Temp is '19.9', the error is now 0.1, the proportional element is therefore 2, the integral element is now 100/300 of the gain, or 6.6, so the module writes '8.6' to the OutPercent property.

If after 110 seconds the Current Temp is '20', the error is now 0, the proportional element is 2, the integral element is now 110/300 of the gain, or 7.3, so the module writes '7.3' to the OutPercent property.

If after 120 seconds the Current Temp is '20.1', the error is now -0.1, the proportional element is 0, the integral element is now 100/300 of the gain, or 6.3, so the module writes '6.3' to the OutPercent property.

As you can see, the Current Temp overshot the Setpoint, but the integral action is reducing. Ultimately, the OutPercent will stabilise at a positive value, the Current Temp will match the Setpoint, and the system will be stable – until the Setpoint changes again

Related Modules

Rescale

Pulser

Object Type: *[Obv\Pulser]*

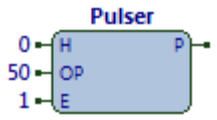


Fig. 78 Pulser module

The Pulser module (Fig. 78) performs the timer operation to produce a continuously cycling digital state.

For the cycle period specified (in seconds), the output state will be set '1' (On) for a percentage of this period (specified by an input), and '0' (Off) for the remainder of this time.

Use the module, for example, to provide a time-based trigger input to modules such as Object-Read, Usage-Over-Period, etc.

When enabled, the operation is:

```
P = 1
wait for OP% of H
P = 0
wait for (100-OP)% of H
```

The module contains the following objects:

Description	Reference	Type
Period (s) The length of the cycle, in seconds	H	Obj\Float; Adjustable; Default value: 0
On Percent (%) Percent of the Period that the Pulse output will be set to '1' (On)	OP	Obj\Float: 0…100; Adjustable; Default value: 50
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and P is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Pulse The calculated output state	P	Obj\OffOn

Example

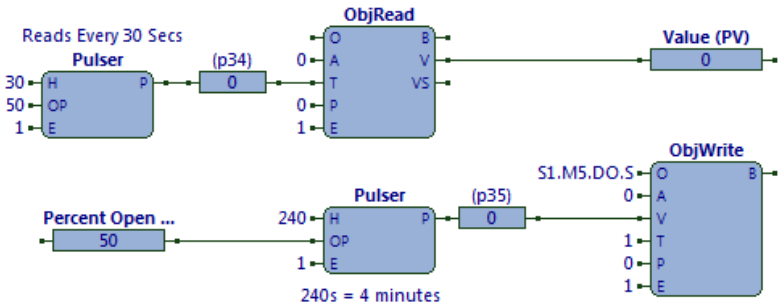


Fig. 79 Add module example

The ObVerse strategy (Fig. 79) shows two examples of using the Pulser module.

The upper example uses a Pulser module output to trigger the reading of an object.

The module has input H set to '30' seconds and input OP set to '50' %. So the output state will be set to '1' (On) for 50% of 30 seconds (15 seconds), then '0' (Off) for 50% of 30 seconds (another 15 seconds), continuously. The object-read action is performed whenever the trigger input changes from '0' to '1'.

The lower example uses a pulser module to generate a pulse-width modulation (PWM) style output. The PWM duty cycle is specified by the Percent Open property. A 240 second period, input H, is chosen to stop the output chattering too much. The output, P, is passed to the value of an object-write. This value will write on change to a Zip relay output.

Some external task writes a value to the Percent Open property. If the Percent Open has a value of 10, then at the start of its next cycle, the Pulser module will write ‘1’ to the private property. After 10% of 240 seconds (24 seconds) it will write ‘0’ to the private property. After another 216 seconds the cyucle will end and the next cycle starts with the module reading the inputs again.

Related Modules

On-Off-Delay, System-Information

Raise-Lower

Object Type: *[Obv\RaiseLower]*

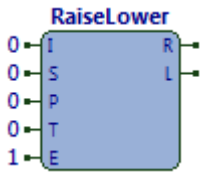


Fig. 80 Raise-Lower module

The RaiseLower module (Fig. 80) performs the control operation to regulate flow through a valve fitted with a motor to drive the valve both open (raise the flow) and closed (lower the flow). When the motor is not driven, the valve remains in position. This allows control over the amount of flow through the valve.

Due to gearing, the motor takes time to drive the valve from fully closed to fully open, and vice versa. This period is called the 'stroke time', and determines the amount of time needed to reposition the valve to the required position using one of the motor drives.

The module has a 'precision' input. This reduces motor usage by only repositioning the valve when a significant change occurs – the required position must change more than the precision input before repositioning occurs.

As the required valve position changes and the module continuously repositions the valve, the valve's actual position can drift away from where the module calculates. To overcome this drifting, the module automatically overdrives the motors when the fully open or fully closed positions are required, to ensure the valve is in the desired position. If the module drives on continuously operating plant, the module supports a resynchronisation trigger, which causes the module to overdrive the valve to the nearest end position before driving it back to the required position.

When enabled, the operation is:

```
if I > (I_previous + P) then
   R = 1
   wait for (I - I_previous)% of S
   R = 0
else if I < (I_previous – P) then
   L = 1
   wait for (I_previous - I)% of S
   L = 0
```

The module contains the following objects:

Description	Reference	Type
Input Position (%) Required position of valve, as a percentage open	I	Obj\Float: 0…100; Adjustable Default value: 0
Stroke time (s) The time required to move the value from fully closed to fully open. Also used as overdrive time at either end of stroke	S	Obj\Num: 0…3600; Adjustable Default value: 0
Precision (%) Minimum change in Input position required to cause a raise/lower movement	P	Obj\Float: 0…50; Adjustable Default value: 0
Resync Trigger When this changes from '0' (No) to '1' (Yes), the module starts a resynchronisation of the valve's position, by overdriving to the nearest end, before driving back to the Input position	T	Obj\NoYes; Default value: 0 (No)

Description	Reference	Type
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs R and L are left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Raise Output set to '1' (Yes) when valve opening is required	R	Obj\NoYes
Lower Output set to '1' (Yes) when valve closing is required	L	Obj\NoYes

Example

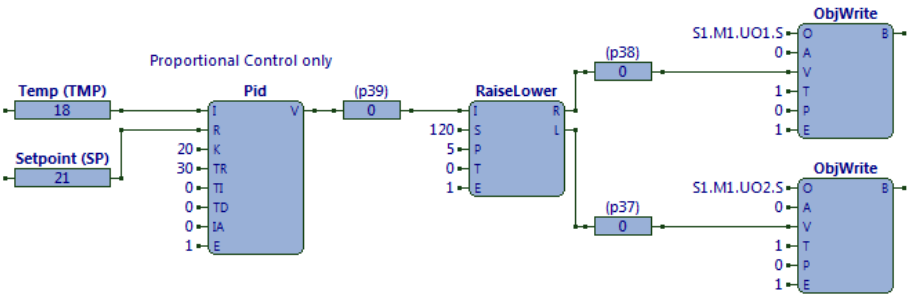


Fig. 81 Raise-Lower module example

The ObVerse strategy (Fig. 81) uses a Pid module to determine the percentage of heat required, based on the difference between the Temp and the Setpoint parameters. The RaiseLower module uses this heat demand to control a valve position, passing the raise and lower outputs to Zip digital outputs using the ObjWrite modules.

The RaiseLower module's input position, I, is provided by a Pid module. Stroke time, S, is set to '120' seconds, and precision, P, to '5'. Enable is set to '1' (Yes).

Some external task writes the Setpoint and periodically writes the Temp property. These are fed into a proportional-only Pid module, which in turn feeds the RaiseLower module. When the Temp value is written which causes the Pid output to change by more than 5% (the precision input to the RaiseLower),the RaiseLower module will set its output R to '1' (On) for a percentage of the stroke time, depending on where it had driven it to before, and the amount of changes of its input I. When the Pid output decreases value by more than 5, then the RaiseLower module will set its output L to '1' (On) for a percentage of the stroke time, depending on where it had driven it before, and the amount of change of its input I.

If the Temp value stays higher than the Setpoint, the Pid will write '0' to the private property. The RaiseLower will write a '1' to its output L for an 'overdrive' period, thereby ensuring that the valve really is at its lowest level.

Related Modules

Pulser

Availability

Available in standard and advanced processor versions dated November 2015 and later.

Random

Object Type: *[Obv\Random]*

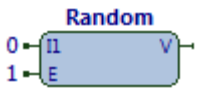


Fig. 82 Random module

The Random module (Fig. 82) performs the maths operation to create a random number.

The module outputs a pseudo-random floating-point number in the range 0 .. 0.9999 every time its input value changes. The output is not guaranteed to be truly random, but can be used for examples and demos.

When enabled, the operation is:

```
if I1 != I1previous then
   V = Random()
```

It contains the following sub-objects:

Description	Reference	Type
Input Trigger Input to trigger the calculation of a new random output	I1	Obj\Float; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Value The last calculated random value	V	Obj\Float: 0…0.9999

Example

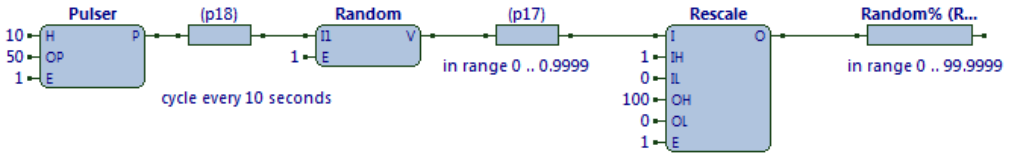


Fig. 83 Random module example

The ObVerse strategy (Fig. 83) produces a random number in the range 0 to 99.9999, changing every 5 seconds.

The Pulser module cycles its digital output every 10 seconds – 5 seconds on, and 5 seconds off. The Random module generates a new random number (in the range 0…0.9999) every time its input changes. Since the Random module never outputs 1.0, the Rescale module effectively rescales from 0…0.9999 to 0…99.99, producing a random percentage value suitable for demonstrating.

Rescale

Object Type: *[Obv\Rescale]*

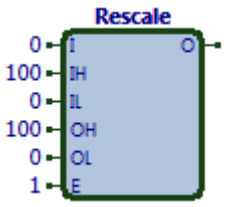


Fig. 84 Rescale module

The Rescale module (Fig. 84) performs the maths operation to rescale its input to its output, using an input range and an output range. It assumes the input and output are proportional. It also limits the output value to the output range.

When enabled, the operation is:

```
if (I > IH) then
   O = OH
else if (I < IL) then
   O = OL
else
   O = OH – ((IH - I) x (OH - OL) / (IH - IL))
```

The module contains the following objects:

Description	Reference	Type
Input The input value to rescale	I	Obj\Float; Adjustable Default value: 0
Input High The high limit of the input range	IH	Obj\Float; Adjustable Default value: 100
Input Low The low limit of the input range	IL	Obj\Float; Adjustable Default value: 0
Output High The high limit of the output range	OH	Obj\Float; Adjustable Default value: 100
Output Low The low limit of the output range	OL	Obj\Float; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and O is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Output The last calculated value	O	Obj\Float

Example

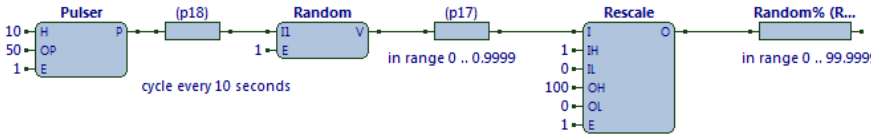


Fig. 85 Rescale module example

The ObVerse strategy (Fig. 83) rescales a number from the input range 0 to 0.9999 to the output range 0 to 99.99.

The Pulser module cycles its digital output every 10 seconds – 5 seconds on, and 5 seconds off. The Random module generates a new random number (in the range 0…0.9999) every time its input changes. Since the Random module never outputs 1.0, the Rescale module effectively rescales from 0…0.9999 to 0…99.99, producing a random percentage value suitable for demonstrating.

Related Modules

Proportional-Integral-Derivative

Select

Object Type: *[Obv\Select]*

The Select module (Fig. 86) performs the logic operation to select one of its inputs to copy to its output.

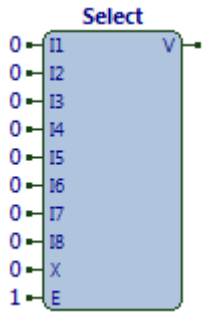


Fig. 86 Select module

When enabled, the operation is:

```
if X == 0 then
   V = I1
else if X == 1 then
   V = I2
else if X == 2 then
   V = I3
else if X == 3 then
   V = I4
else if X == 4 then
   V = I5
else if X == 5 then
   V = I6
else if X == 6 then
   V = I7
else if X == 7 then
   V = I8
else
   V = 0
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input x, where *x* is in the range 1..8	I*x*	Obj\Float; Adjustable; Default value: 0
Selector Used to select which input to pass to the output value	X	Obj\Num; Range 0..7; Adjustable; Default value: 0 (Input1) Values: 0=Input1, 1=Input2, 2=Input 3, 3=Input4, 4=Input5, 5=Input6, 6=Input7, 7=Input8
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value Set to selected input value	V	Obj\Float

Example

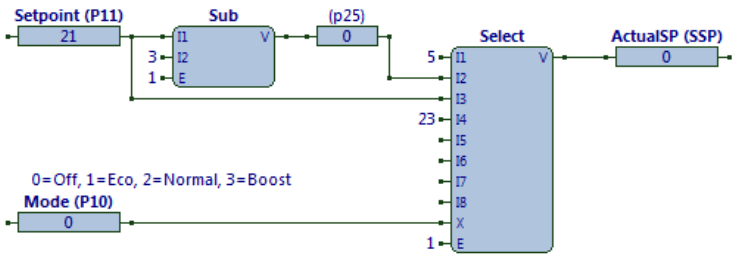


Fig. 87 Select module example

The ObVerse strategy (Fig. 87) uses a Mode property to select one of four heating setpoints.

The Select module has input 1 set to ‘5’, input 2 is linked to the result of a user setpoint minus 3, input 3 is linked to a user setpoint property, and input 4 is set to ‘23’. The selector, X, is linked to the Mode property. Enable is set to ‘1’ (Yes).

Some external task writes a value to the Setpoint and Mode properties. If Setpoint holds ‘21’, the Sub module subtracts 3 from it to produce ‘18’, which it writes to a private property. If the value within Mode is ‘2’, the Select module writes the value from its input I3, ‘21’, via its output V to the ActualSP property. If the Mode property changed to ‘0’, the select module writes the value from its input I1, ‘5’, via its output V to the ActualSP property.

Related Modules

Num-To-Bit

Square-Root

Object Type: *[Obv\Sqrt]*

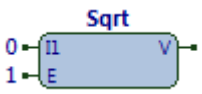


Fig. 88 Square-Root module

The Sqrt module (Fig. 88) performs the maths operation to calculate the square root of its input.

When enabled, the formula is:

$$V = \sqrt{I1}$$

The module contains the following objects:

Description	Reference	Type
Input Input to calculate the square root of	I1	Obj\Float; Adjustable; Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable; Default value: 1 (Yes)
Value The last calculated value	V	Obj\Float

Example

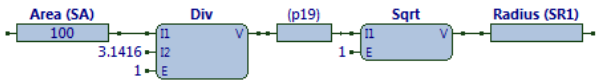


Fig. 89 Square-Root module example

The ObVerse strategy (Fig. 89) calculates the radius of a circle, from the area.

Some external task writes a value to the Area property. If the Area property has a value of '60', the Div module outputs a value of '19.0985' to the private property, which is used by the Sqrt module to write a value of '4.3702' to the Radius property.

Related Modules

Divide, *Modulus-Remainder*

Subtract

Object Type: *[Obv\Sub]*

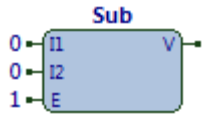


Fig. 90 Subtract module

The Sub module (Fig. 90) performs the maths operation to subtract one input from the other.

When enabled, the formula is:

```
V = I1 - I2
```

The module contains the following objects:

Description	Reference	Type
Input *x* Input to include in calculation, where *x* is in the range 1..2 Input 1 is the minuend, Input 2 is the subtrahend	I*x*	Obj\Float; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Value The last calculated value, the difference	V	Obj\Float

Example

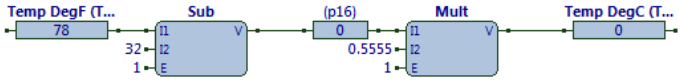


Fig. 91 Subtract module example

The ObVerse strategy (Fig. 91) converts the temperature in degrees Fahrenheit to degrees Celsius, by using the formula:

°C = (°F - 32) x (5 / 9)

Some external tasks writes a temperature value (in degrees Fahrenheit) into the Temp DegF property. The Sub module subtracts 32 from this, and writes the result to the private property. The Mult module reads the private property, multiplies it by 0.5555 (5/9), and writes the final result to the Temp DegC property.

Related Modules

Add, *Multiple-Add*

System-Information

Object Type: *[Obv\SysInfo]*

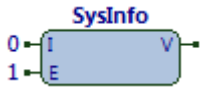


Fig. 92 System-Information module

The SysInfo module (Fig. 92) performs the system operation to make fundamental system information from the device available for use in ObVerse.

When enabled, the input value I selects which system information is output:

Input	Information	Value
1	Current time – seconds	0…59
2	Current time – minutes	0…59
3	Current time – hours	0…23
4	Current date – day	1…31
5	Current date – month	1…12
6	Current date – year	Four-digit year, e.g. 2016
7	Current date – day-of-week	0=Monday, 1=Tuesday, 2=Wednesday, 3=Thursday, 4=Friday, 5=Saturday, 6=Sunday
10	Pulse when Processor starts to run (either on ObVerse download or device restart)	1, momentarily for a single processor cycle, then 0
11	Pulse on each second change in time	1, momentarily for a single processor cycle, then 0
12	Pulse on each minute change in time	1, momentarily for a single processor cycle, then 0
13	Pulse on each hour change in time	1, momentarily for a single processor cycle, then 0
14	Pulse on each day change	1, momentarily for a single processor cycle, then 0
15	Pulse on each month change	1, momentarily for a single processor cycle, then 0

The module contains the following objects:

Description	Reference	Type
Input Set to the system information required. See table above	I	Obj\Num; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and V is left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
Value The value of the specified input	V	Obj\Text

Example

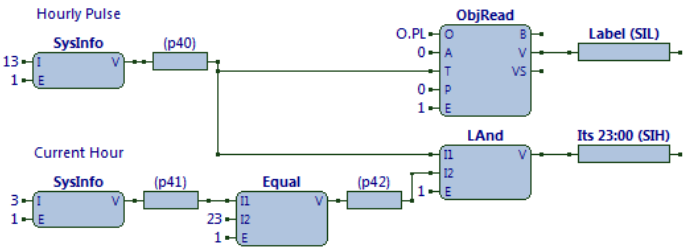


Fig. 93 System-Information module example

The ObVerse strategy (Fig. 93) uses one system-information module to provide an hourly pulse that triggers reading an object, and another to determine when the time is exactly 23:00.

The upper SysInfo module has its input set to ‘13’, so provides a pulse output each time the hour changes. This output is set to ‘1’ for a single cycle of the ObVerse processor, before being reset to ‘0’. This trigger is used by an ObjRead module, to trigger it’s operation each hour.

The lower SysInfo module has its input set to ‘3’, so provides the hour value of the current time. This is linked to an Equal module to determine when it is ‘23’. A LAnd module combines the Equal output and the hourly pulse from the top SysInfo module to pulse an output when it is exactly 23:00.

Related Modules

Object-Read

Usage-Over-Period

Object Type: *[Obv\Usage]*

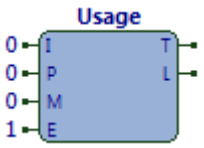


Fig. 94 Usage-Over-Period module

The Usage module (Fig. 94) provides the maths operation to determine, from an incrementing input, how much the input has increased during a period.

The module allows for the rollover of the incrementing value, rather than producing a large negative usage.

When enabled, the operation is:

```
T = Ip - I
if (P) then
   L = T
   Ip = I
```

The module contains the following objects:

Description	Reference	Type
Input Input of the incrementing, or accumulated, value	I	Obj\Num; Adjustable Default value: 0
End Period Pulse When this changes from '0' (Off) to '1' (On), the current period ends, and a new period starts. The usage for This Period is copied to Last Period, and This Period usage is set to '0'.	P	Obj\NoYes; Adjustable Default value: 0 (No)
Maximum If set to a value, then when the Input rolls around to zero, it is assumed the usage to have rolled through this maximum value. If set to '0', then when the Input rolls around to zero, the module will estimate the maximum value.	M	Obj\Num; Adjustable Default value: 0
Enable Enables the module's operation. If set to '0' (No), then no calculation occurs and outputs are left unchanged	E	Obj\NoYes; Adjustable Default value: 1 (Yes)
This Period The increments that have occurred since the End Period Pulse occurred.	T	Obj\Num
Last Period The increments that occurred during the last period	L	Obj\Num

Example

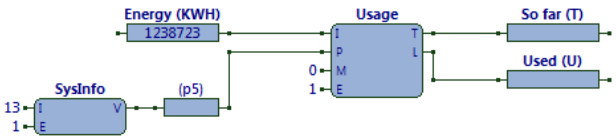


Fig. 95 Usage-Over-Period module example

The ObVerse strategy (Fig. 95) calculates how much energy was used in the last hour.

The Usage module receives an input value from an energy meter. An hourly pulse from the SysInfo module triggers the Usage module to calculate the energy used in the last complete hour, which is passed to the Used parameter. Energy used in the current hour is passed to the So far parameter.

Some external task writes a metered value to the Energy property. If the value is '123' when the SysInfo pulse occurs, the Usage module copies the output value T to output value L, and remembers the value at its input I.

Subsequently, when the Energy property holds '127', the module subtracts its remembered value from that read by input I, and writes the value, '4', via output T to the So far property.

Some time later, when the Energy property holds '135', the module subtracts its remembered value from that read by input I, and writes the value, '12', via output T to the So far property.

Some time later, when the Energy property holds '140' and So far holds '17', the hourly pulse occurs: '17' is written to the Used property, the Energy value '140' is remembered, and subtracted internally from the current Energy '140' to produce a value of '0', which is then written to the So far property.

Related Modules

Counter, Latch

Availability

Available in standard and advanced processor versions dated February 2016 and later.

ObVerse Standard Processor Versions

Version	Build Date	Details
1.0	26/10/2006	ObvProcess released
1.0	01/09/2012	Added modules: BitsToByte, BitToNum, ByteToBits, Linearize, and NumToBit
1.0	01/04/2015	Added modules: Latch, and LeadLag
1.0	01/11/2015	Added module: RaiseLower
1.0	01/02/2016	Added module: Usage
1.1	16/03/2016	Added driver object T to provide information on Object Modules Re-ordered ObVerse modules for use with ObvEditor

Next Steps…

If you require help, contact support on 01273 694422 or visit *www.northbt.com/support*

North Building Technologies Ltd
+44 (0) 1273 694422
support@northbt.com
www.northbt.com

Author: TM
Checked by: JF

Document issued 07/12/2016.