# vNode Derived Tags

**Reference Manual**



# Index

## General features

Using DerivedTags module it is possible to obtain new tags as a result of calculations involving other tags. It provides two different ways to get a derived value:

- Aggregation Tags: the result is the aggregation of a tag over a fixed period of time.
- Expression Tags: the value is the result of JavaScript expression.

A derived tag is configured as any other regular tag but choosing the DerivedTags module as source instead a communication driver.

## Configuration

### Global options

These options affect all derived tags relying on this module. In the following screenshot we can see the different options:

Configuration

| Property | Value | Output |
|---|---|---|
| AggregatedTags | | |
| Rate | 100 | 100 |
| Delay | 5000 | 5000 |
| Maximum buffer size | 50 | 50 |
| ExpressionTags | | |
| Execution Rate | 100 | 100 |
| Tag Rate | 500 | 500 |
| Maximum buffer size | 50 | 50 |

DerivedTags options

As we can see, options are divided into two categories, those affecting **Aggregated Tags**, and those that affect **Expression Tags.**

Options for **Aggregated Tags**:

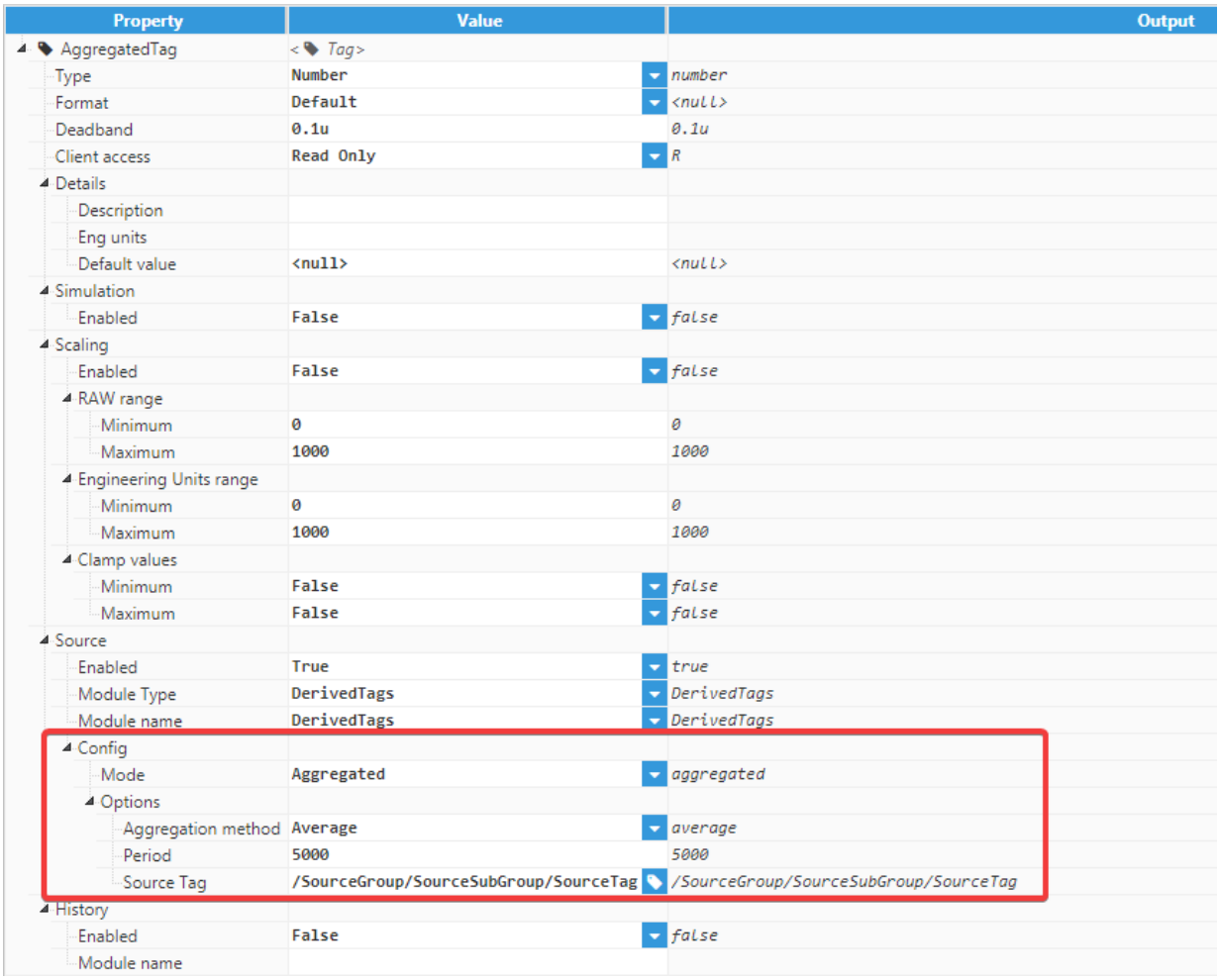
- **Rate:** how often the module checks to see if it has to do any calculation, in milliseconds. The minimum value is 100 ms, but the recommended value is 1000 milliseconds or more, depending on the lowest aggregation period.
- **Delay:** time, in milliseconds, the module waits for the last event to reach this vNode node before it starts the aggregation. With a delay of 5000 milliseconds, and a period of one minute, the calculation is executed at 00:01:05, 00:02:05, etc, although the interval will still be from 00:00:00 to 00:01:00. The delay is used to correct for network delays. The minimum value is 0, which will mean that the calculations will start as soon as the period finishes.
- **Maximum buffer size:** the amount of events that can be saved into the internal buffer before old events are discarded. If this value is too small and the aggregation period is too long, some events might be discarded from the current interval, so this value should be high enough to allow to buffer to contain all events relevant to at least the current interval. This value is per tag, so in this example all tags would have a buffer of 50 events.

Options for **Expression Tags:**

- **Execution Rate:** how often the module checks to see if any expressions are ready to execute, in milliseconds. This is used to cluster together multiple expressions, and then run them all at the same time. A higher value increases performance by clustering many expressions at the cost of a higher latency to generate the result.
- **Tag Rate:** same as the **Rate** in **Aggregated Tags**, it's how often each tag checks to see if it has to execute. This is only used if Expression Tags is working in synchronous mode.
- **Maximum buffer size:** same as in **Aggregated Tags**. The only difference is that the buffer in this case is per alias, so a tag with 2 alias and a buffer size of 50 can hold up to 100 events, 50 for each alias.

## Aggregated Tags

Aggregated Tags are used to generate aggregated values based on the values of other tags acting as data source. In the following screenshot we can see the configuration of any tag, and in the red square is the configuration specific to Derived Tags.




AggregatedTags options

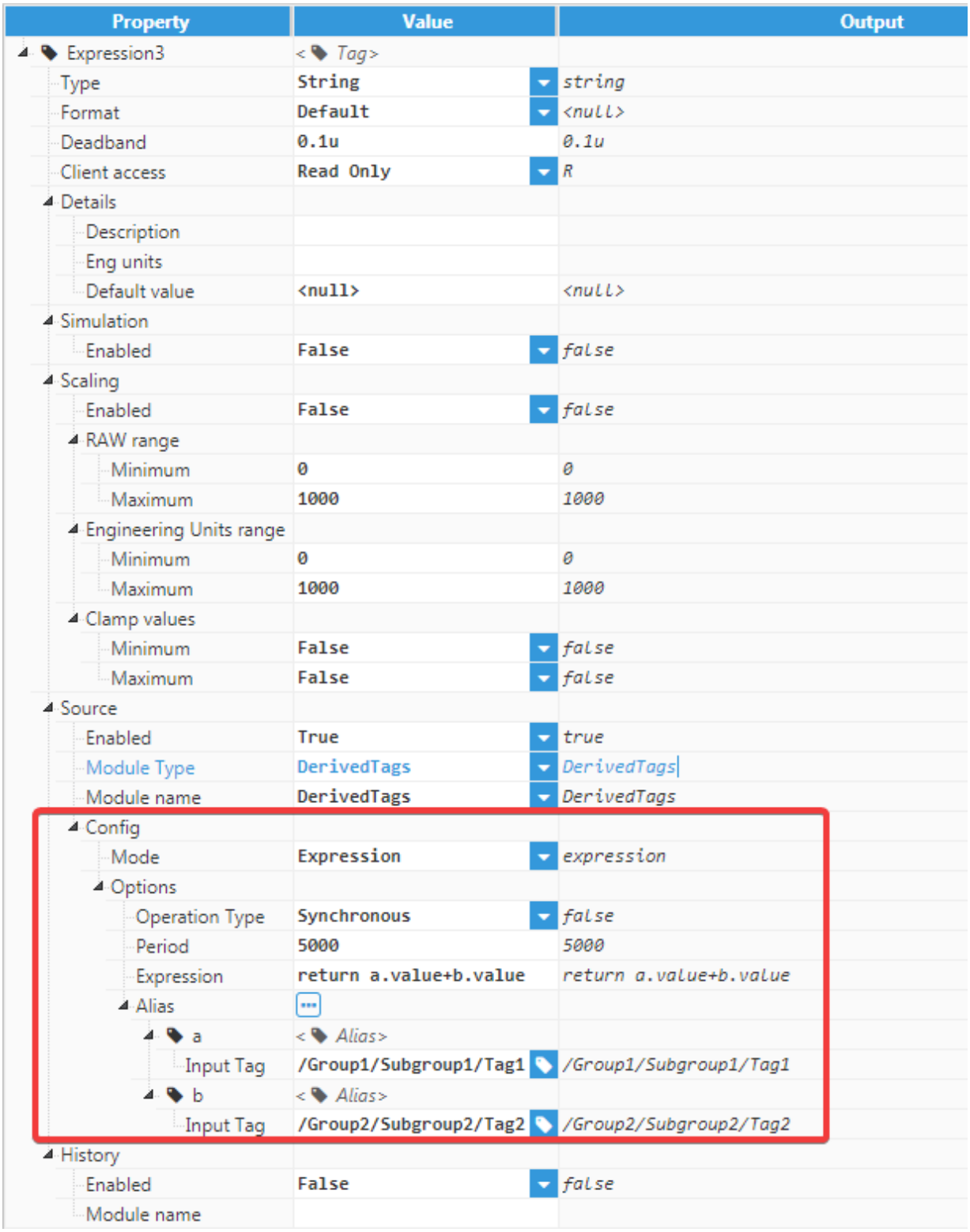
Aggregated Tags options:

- **Aggregation method**:
  - **Average**: Time weighted average of the values received in the selected interval.
  - **Minimum**: Lowest value received in the interval.
  - **Maximum**: Highest value received in the interval.
  - **First**: The first value that occurs in the interval. This is not the first value received, but the value at the start of the interval (which typically would be the last value of the last interval).
  - **Last**: Last value received in the interval.
  - **Count**: Total number of events received in the interval. This includes both good and bad quality events, as well as good quality events with null value.

    - **Good**: Ratio of good quality events to total received events, expressed as a decimal. 1 means all events had good quality, 0 means all events had bad quality.
- **Period**: how often the aggregation will be run, it has a minimum value of 1000 milliseconds and can be as high as needed.
- **Source Tag**: source that will be used for the aggregation. This can be any tag, including tags obtained via a link with another vNode instance. However, if the tag is not available on startup, it will log a warning and all DerivedTags with that tag as a source will stop.

## Expression Tags

**Expression Tags** are obtained as a result of JavaScript expressions.



| Property | Value | Output |
|---|---|---|
| Expression3 | <Tag> | |
| Type | String | string |
| Format | Default | <null> |
| Deadband | 0.1u | 0.1u |
| Client access | Read Only | R |
| Details | | |
| Description | | |
| Eng units | | |
| Default value | <null> | <null> |
| Simulation | | |
| Enabled | False | false |
| Scaling | | |
| Enabled | False | false |
| RAW range | | |
| Minimum | 0 | 0 |
| Maximum | 1000 | 1000 |
| Engineering Units range | | |
| Minimum | 0 | 0 |
| Maximum | 1000 | 1000 |
| Clamp values | | |
| Minimum | False | false |
| Maximum | False | false |
| Source | | |
| Enabled | True | true |
| Module Type | DerivedTags | DerivedTags |
| Module name | DerivedTags | DerivedTags |
| Config | | |
| Mode | Expression | expression |
| Options | | |
| Operation Type | Synchronous | false |
| Period | 5000 | 5000 |
| Expression | return a.value+b.value | return a.value+b.value |
| Alias | ... | |
| a | <Alias> | |
| Input Tag | /Group1/Subgroup1/Tag1 | /Group1/Subgroup1/Tag1 |
| b | <Alias> | |
| Input Tag | /Group2/Subgroup2/Tag2 | /Group2/Subgroup2/Tag2 |
| History | | |
| Enabled | False | false |
| Module name | | |

Expression Tags options

After selecting **Expression** mode, the Expression Tags options will show. The first option is **operation type**, it can either be

**Expresssion Tags options:**

- **Operation type:**

  - **Asynchronous:** the expression will execute whenever one of the inputs change (and sets the period to 0 by default, since it won't be used)
  - **Synchronous:** the expression will execute once the period passes. The period can be set to any positive value, including zero, however, it is not recommended to go beneath 1000 ms. Setting the period to zero and the mode as synchronous is invalid and will invalidate the tag on startup.

- **Period**: when configured as synchronous, period of time to execute the expression, in milliseconds. Only applies to Synchronous expressions.
- **Expression**: JavaScript code to be executed. Any valid JavaScript code can be used, including functions, however, some newer JavaScript syntax might not be supported (such as arrow functions), and the script doesn't have access to any external libraries. See examples at the end of the document.
- **Alias**: in this section it is possible to create aliases so the expression can use these short names instead the entire tag path. Aliases contain the latest good quality events with the value, timestamp and quality. This information can be accessed using the following syntax:
  - Value: **a.value**
  - Timestamp: **a.ts**
  - Quality: **a.quality**

### Variables

The JavaScript expression can have user defined variables, and also supports global variables that are shared between all calculations within the DerivedTags module. In order to make a variable global, it is necessary to define it as a property of "**this**" (for instance this.myvariable), and it should be defined only once across the module. The value of global variables is retained from one execution cycle to the next although the value will be restarted when the DerivedTags module is restarted.

User defined variables should not have the same name as the alias, as this will overwrite it, and the module might not function properly.

## Examples

The following examples are presented in multiple lines and include comments for clarity although in the actual expression must be written in a single line with each statement ending with semicolon.

### Sum of two tags:

```
return a.value + b.value;
```

### Set a value based on a condition:

```
//If the condition is true, it will execute the statement '1', which in
JS will return the value 1, otherwise, it will return 0
if(a.value > threshold) return 1; else return 0;
```

### Increment a global counter based on a condition:

```
//This is used to initialize the counter if it's not initialized.
if(!this.counter) this.counter=0;
//This is the condition to check for, in this case it checks if the tag
value is even, and if it is, it increases the counter
if(a.value/%2 == 0) this.counter++;
//This returns the value of the counter.
return this.counter;
```

### Sum of two tags using a function

```
//This defines the function (it gets defined every time the expression is
run, which can affect performance)
function sum(a,b){return a.value+b.value};
//Then the function is invoked
sum(a,b)
```

### Sum of two tags using a global function

```
//This defines the function only once, when it's undefined
if(this.sum == undefined)
    this.sum = function(a,b){return a.value+b.value};
//Then the function is invoked
sum(a,b)
//This method also allows the same function to only be defined in one tag
and used everywhere else (although this might throw a warning if the rest
of the tags trigger before the function is created).
```

### Sum of a variable in a loop:

```
//This defines the accumulator
var acc = 0;
//This creates the loop that adds to the accumulator
for (var i = 0; i<1000; ++i) acc += i;
//This returns the accumulator
return acc;
```