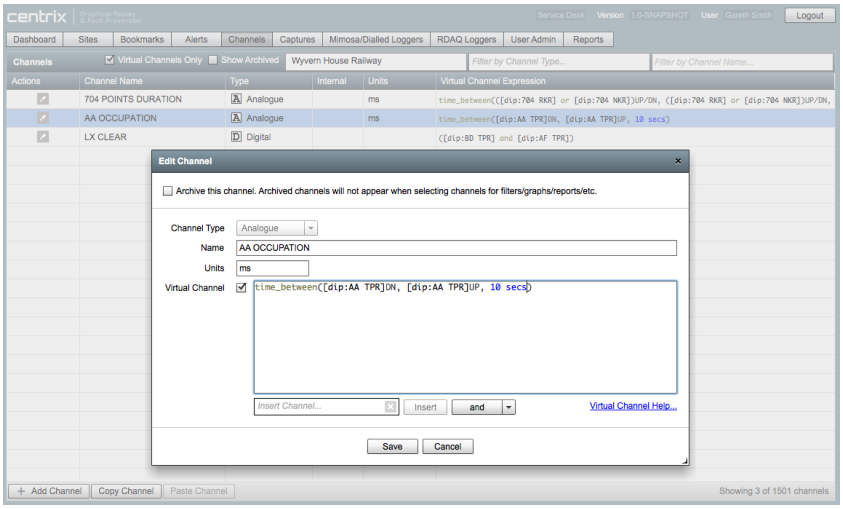
Creating Virtual Channels

> When saving changes to an existing channel, the whole virtual channel expression is recreated which will invalidate the "armed" state regardless of if there are any changes to the virtual channel expression.

Centrix allows the definition of "virtual" channels that are formed by combining other channels using a variety of operators. They provide a convenient way of measuring things on the railway that are more sophisticated than a single analogue or digital input to a logger.

In Centrix, the channels tab displays all channels for a particular site. Most of these channels will come from a logger directly, but you also add your own channels here and provide a formula that describes how the values in the channel should be created: such channels are virtual channels.



Examples

Boolean Logic

```
(([dip:A TR] and [dip:B TR]) and [dip:C TR])
```

The above virtual channel will output a value of `true` (i.e. UP) only when all input dips (**di**gital in**p**ut) `A TR`, `B TR` and `C TR` are true (i.e. when all tracks are clear). If any of the tracks are occupied, the `and` condition will output false (i.e. DN).

You can use the boolean operators `and`, `or`, `xor` and `not`. They can be combined in arbitrary ways, for example:

```
[dip:A] or [dip:B] xor [dip:C] and not [dip:D]
```

Arithmetic

```
([aip:X] * [aip:X])
```

The above virtual channel will output the value of the analogue channel `X`, squared. You can also use division, addition and subtraction operators.

> **Limitations when using Analogue channels**
>
> There is currently a serious limitation when using analogue channels: you can only use channels that have been created from time_between operators. You cannot use analogue channels from the logger. This is due to limitations of the Mimosa protocol used for sending data between loggers and Centrix. We hope to remove this limitation in future developments.

Comparisons

```
([aip:X] > 12.3)
```

You can use the comparison operators <, <=, > and >=. These operators take two analogue inputs and produce a single digital output (`true` or `false`).

Time Between

General Syntax: `time_between(start-trigger-channel (DN or UP or UP/DN), end-trigger-channel (DN or UP or UP/DN), timeout)`

E.g.: `time_between([dip:A TR]DN, [dip:A TR]UP, 10 mins)`

The `time_between` operator allows you to measure the time elapsed between two digital state changes. This can be used as above, for example, to measure how long a track circuit is occupied. The time_between operator produces analogue values that represent the time between the start trigger and the end trigger in milliseconds. The start/end triggers are fired when the input channel experiences the specified state transition (dn-to-up: UP, up-to-dn: DN or either: UP/DN). When the start trigger fires, a timer is started, and this timer finishes either when the end trigger fires, or when the timeout occurs: when the timer finishes, a new analogue event is emitted on the output channel, and the value of this event is the time between the start trigger and the end trigger or timeout.

The timeout value consists of an integer and a unit, which must be `hour`, `hours`, `min`, `mins`, `sec`, `secs` or `ms`. Note that the output of a time_between channel is always in milliseconds, regardless of the unit used in the timeout.

This operator can be combined with other operators in arbitrary ways: its inputs can be the result of other operations, and its output can be used in additional operations.

Debounce

General Syntax: `debounce(input-channel, pick: pick-timeout, drop: drop-timeout)`

E.g.: `debounce([dip:A], pick: 1 sec, drop: 0 ms)`

The `debounce` operator takes a single digital as its input and outputs a digital that matches the input after the input has been in a stable state for a given time period. This is similar to a slow-pick or slow-drop relay. In the example above, the output will go true (UP) after the input has been true continuously for 1 second, but as soon as the input goes false (DN) the output will also go down.

Operator Precedence

The operator precedence in virtual channel expressions is similar to programming languages like C and Java. This is the complete table, with the highest precedence (applied first) operator at the top, and the lowest precedence operator at the bottom. Brackets can be used to override the default precedences.

Operators	Precedence
prefix	`not time_between debounce`
multiplicative	`/ *`
additive	`+ -`
comparison	`> >= < <=`
logical AND	`and`
logical XOR	`xor`
logical OR	`or`