

# 18.2 Optional parameters for signals

The signals sheet contains all signals linked to devices. Each signal is defined in a single row. The Signal list can be split into multiple sheets. Each sheet name may start as Signals.

## Required attributes

These attributes are mandatory for every configured signal. Every Excel configuration should have specified them in the first row of the Signals sheet:

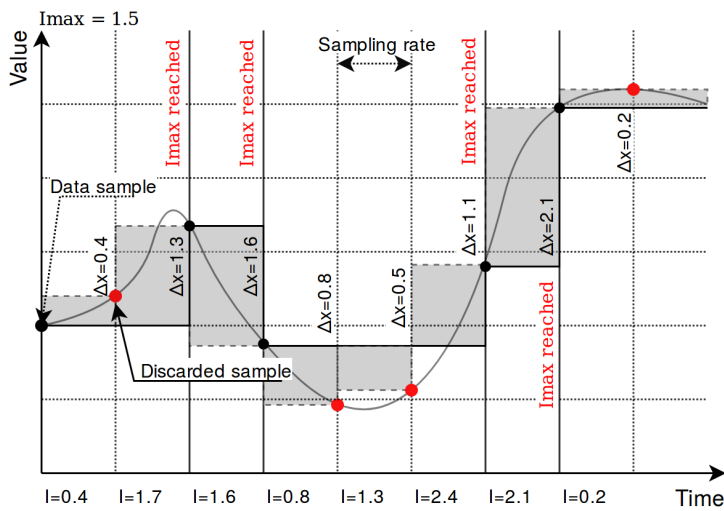
- **signal\_name** - Name of the signal. Used for representation only.
- **device\_alias** - Alias of a device defined in the Devices sheet. A signal is linked to a matching device.
- **signal\_alias** - A unique short name for the signal. It is used for linking signals to other signals. The alias can only contain alphanumeric characters and dashes ( - and \_ ). The device and signal alias combination must be unique.

## Optional attributes

Optional attributes are required depending on the protocol in use and they can be used to extend protocol functionality:

- **source\_device\_alias** - Alias of a source device defined in the Devices sheet. If a user intends to use several signals and combine them via mathematical or logical function, every alias should be separated by a newline symbol (in the same cell). An operation used must also be defined in an operation column.
- **source\_signal\_alias** - Alias of a source signal defined in the Signals sheet. If a user intends to use several signals and combine them via mathematical or logical function, every alias should be separated by a **separator** symbol (in the same cell). An operation used must also be defined in an operation column. Each `source_signal_alias` should be posted in the same line as its respective `source_device_alias`. Aliases can only contain alphanumeric characters and dashes ( - and \_ ). The device and signal alias combination must be unique.
- **enable** - Flag to enable or disable signal on the system. Can contain values 0 or 1.
- **tag\_type** - Tag type. Simple signals are polled from the device. Virtual signals are computed internally.
- **units** - Signal value measurements units.
- **multiply** - Multiply the value by this number.
- **add** - Add this number to a value.
- **min\_value** - Minimum expected value. If the result is lower than this value, the overflow flag is raised.
- **max\_value** - Maximum expected value. If the result is higher than this value, the overflow flag is raised.
- **absolute\_threshold** - Absolute threshold level.
- **integral\_threshold** - Integral threshold level.
- **integral\_threshold\_interval** - Integral threshold addition interval in milliseconds.
- **threshold\_units** - Units used in threshold level fields (percent/real).
- **log** - Maximum number of records for this tag to keep in events log.
- **suppression\_values** - Space-separated numeric values to be used in suppression.
- **suppression\_time\_ms** - Suppression time in milliseconds.
- **operation** - Mathematical or logical operation to be used for signals defined in `source_signal_alias` column which are separated using **separators**. Following mathematical operations for source signal values can be used: avg (average of all values), min (lowest value), max (highest value), median (median value), and sum (all values accumulated to a single number). Logical operations are intended for unsigned integers only.
- **bit\_select** - selecting an individual bit of an integer number; bit numeration starts from zero.
- **math\_expression** - a mathematical expression for master protocol monitor direction or slave command direction signals to be evaluated against. Explained in detail in **Mathematical expression document**.
- **source\_math\_expression** - a mathematical expression for master protocol command direction or slave monitor direction signals to be evaluated against. Explained in detail in **Mathematical expression document**.

[Picture. Result of using an absolute threshold:](#)



Picture. Result of using an integral threshold:



## Signal recalculation operation priority

A value generated by some protocol usually has to be recalculated in one way or another. This might mean changing the value of an argument as well as adding flags needed for other protocols to correctly interpret results. As recalculation is a sequential process, some actions are done before others. The sequence of operations done to a value is as follows:

- *Edition of attributes.* Attributes for further interpretation are added. This might, for example, include a flag to show that a signal resembles an answer to a command;
- *Mathematical calculations.* **multiply**, **add**, **bit\_select**, and **math\_expression** columns are evaluated here;
- *Usage of last value.* The decision if last value for a signal should be used if a new value of a signal is not a number (NaN) or contains a non-topical (NT) flag;
- *Limiting of values.* If a value exceeds a lower or higher configured limit, the value is approximated not be lower (or higher) than the limit. An additional overflow (OV) flag is added as frequently used in IEC-60870-5 protocols;
- *Suppression of values.* As electrical circuits can be noisy, protocols may generate multiple values in a short amount of time. What is more, some values are considered as intermediaries and ideally should not be sent to SCADA unless they stay in the same state for some amount of time. **suppression\_values** and **suppression\_time\_ms** are used to configure this functionality;
- *Threshold checking.* If a new signal doesn't cross a threshold target value, the value is suppressed and not used in further stages. **absolute\_threshold**, **integral\_threshold**, **integral\_threshold\_interval**, **threshold\_units** columns are used to configure this functionality.

❗ Not all of the elements in this sequence have to be configured, missing operations are skipped and values are fed to a further stage of signal recalculation.

## number\_type field

This field is required for some protocols to determine a method to retrieve a signal value from hexadecimal form. Available values:

- **FLOAT** - 32-bit single precision floating point value according to IEEE 754 standard
- **DOUBLE** - 64-bit double precision floating point value according to IEEE 754 standard
- **DIGITAL** - 1-bit boolean value
- **UNSIGNED8** - 8-bit unsigned integer (0 - 255)
- **SIGNED8** - 8-bit signed integer (-128 - 127)
- **UNSIGNED16** - 16-bit unsigned integer (0 - 65535)
- **SIGNED16** - 16-bit signed integer (-32768 - 32767)
- **UNSIGNED32** - 32-bit unsigned integer (0 - 4294967295)
- **SIGNED32** - 32-bit signed integer (-2147483648 - 2147483647)
- **UNSIGNED64** - 64-bit unsigned integer (0 - 18446744073709551615)
- **SIGNED64** - 64-bit signed integer (-9223372036854775808 - 9223372036854775807)

Number conversion uses **big-endian** byte order by default. Converted data will be invalid if the byte order on the connected device side is different. In such a case, byte swap operations can be used. Adding swap prefixes to number types will set different byte orders while converting values. Following swap operations are available:

- **SW8** - Swap every pair of bytes (8 bits) (e.g.,**0xAABBCCDD** is translated to **0xBBAADDCC**);
- **SW16** - Swap every pair of words (16 bits) (e.g.,**0xAABBCCDD** is translated to **0xCCDDAABB**);
- **SW32** - Swap every pair of two words (32 bits) (e.g.,**0x1122334455667788** is translated to **0x5566778811223344**);

Table. Example of using different swapping functions:

| Address         | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      |
|-----------------|--------|--------|--------|--------|--------|--------|--------|--------|
| Original number | Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
| SW8             | Byte 1 | Byte 0 | Byte 3 | Byte 2 | Byte 5 | Byte 4 | Byte 7 | Byte 6 |
| SW16            | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 1 | Byte 6 | Byte 4 | Byte 5 |
| SW32            | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| SW8.SW16        | Byte 3 | Byte 2 | Byte 1 | Byte 0 | Byte 7 | Byte 6 | Byte 5 | Byte 4 |
| SW8.SW32        | Byte 4 | Byte 4 | Byte 7 | Byte 6 | Byte 1 | Byte 0 | Byte 3 | Byte 2 |
| SW8.SW16.SW32   | Byte 7 | Byte 6 | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |


Where Byte x, means bit x position in the byte.

Add a dot-separated prefix to the number format to use byte swapping. Multiple swap operations can be used simultaneously. For example, use `SW8.SW16.SIGNED32` it to correctly parse a 32-bit signed integer in a little-endian format. Table 35 shows in detail how bytes, words, or double-words can be swapped and how swapping functions can be combined to make different swapping patterns. The table shows how byte swap is done for 64-bit (8-byte) numbers. It doesn't matter if it is an unsigned/signed integer or double, byte swapping is considered a bit-level operation. If a number is shorter than 64 bits, the same logic applies, the only difference is the unavailability of some swapping operations (`SW32` for 32-bit and smaller numbers). Using such an unavailable operation might lead to undefined behavior.

# Linking signals

Signals can be linked together to achieve data transfer between several protocols. If a signal source is defined, all output from that source will be routed to the input of the target signal. This way events polled from a Modbus device (e.g., Modbus, IEC 60870-5, etc.) can be delivered to an external station over a different protocol. A signal source is required if a signal is created on a slave protocol configuration to link events between protocols.

## Example 1:

To read a coil state from a Modbus device and transfer it toIEC 60870-5-104 station, the following steps may be taken:


1. Create a Modbus master configuration in the Devices sheet.
2. Create an IEC 60870-5-104 slave configuration in the Devices sheet.

3. Create a signal on the master device to read coil status (function 1).
4. Create a signal on the slave device with a single point type (`data_type = 1`).
5. Set **source\_device\_alias** and **source\_signal\_alias** fields on slave device signal to match **device\_alias** and **signal\_alias** on master device's coil signal.

## Example 2

To write a coil state to a Modbus device on a command from IEC 60870-5-104 station, the following steps may be taken:

1. Follow steps 1-3 from example 1.
2. Create a signal on the slave device with a single command type (`data_type = 45`).
3. Set `source_device_alias` and `source_signal_alias` fields on the master configuration coil signal to match `device_alias` and `signal_alias` on the slave device's command signal. Coil will be written to a value received by a command.
4. Set `source_device_alias` and `source_signal_alias` fields on the command signal to match `device_alias` and `signal_alias` on the master device's coil signal. A command termination signal will be reported to the station on the coil write the result.

 For additional information regarding the configuration of IEC 60870-5-101/103/104 protocols, please refer to "IEC 60780-5-101/103/104 PID interoperability for WCC Lite devices", accordingly.

## Separators

These operators can be used when defining two or more values in a single cell. For example, `source_signal_alias` and `source_device_alias` from different signals have to be written in the same cell but separated by the separators listed below. This is useful when using the operation parameter when trying to do mathematical operations on more than one signal.

- " "
- (newline)
- ","
- ";"

---

🔄Revision #13

★Created 21 March 2022 08:15:07 by Tautvilis

✎Updated 28 October 2022 12:34:03 by Lukas Taroza