

Signals sheet

Signals sheet contains all signals linked to devices. Each signal is defined in single row. Signal list can be split in 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 first row of Signals sheet:

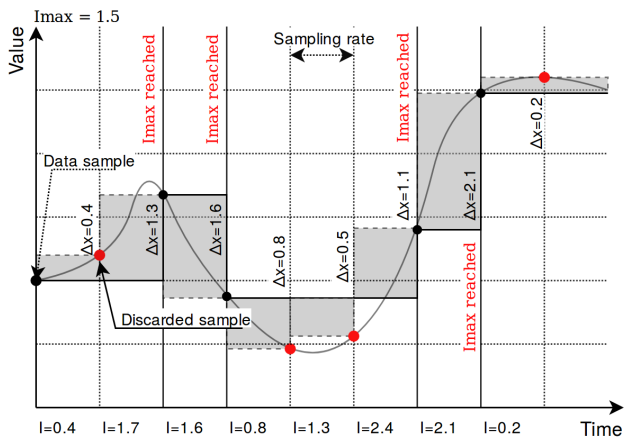
- **signal_name** - Name of the signal. Used for representation only.
- **device_alias** - Alias of a device defined in Devices sheet. Signal is linked to a matching device.
- **signal_alias** - A unique short name for the signal. It is used for linking signal to other signals. Alias can only contain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

Optional attributes

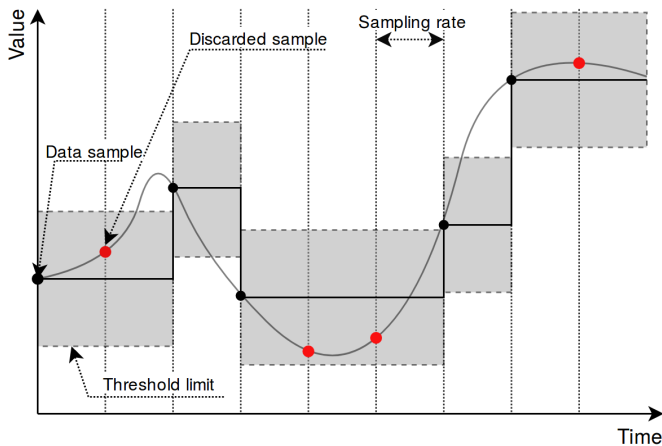
Optional attributes are required depending on protocol in use and they can be used to extend protocol functionality:

- **source_device_alias** - Alias of a source device defined in 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 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 newline symbol (in the same cell). An operation used must also be defined in an operation column. Every `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 _). Device and signal alias combination must be unique.
- **enable** - Flag to enable or disable signal on system. Can contain values 0 or 1.
- **tag_type** - Tag type. Simple signals are polled from device. Virtual signals are computed internally.
- **off_message** - Message to display when single point or double point signals are in OFF state.
- **on_message** - Message to display when single point or double point signals are in ON state.
- **units** - Signal value measurements units.
- **multiply** - Multiply value by this number.
- **add** - Add this number to a value.
- **sum_signals** - Define other signal values to add to current signal. This field uses following **format**: `dev_alias/tag_alias`. Multiple signals can be defines usign commas.
- **min_value** - Minimum expected value. If result is lower than this value, invalid flag is raised.
- **max_value** - Maximum expected value. If result is higher than this value, 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_size** - Maximum number of records for this tag to keep in storage for CloudIndustries logging.
- **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. 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, intended for unsigned integers only, are or and and operations.
- **bit_select** - selecting an individual bit of an integer number; bit numeration starts from zero.
- **math_expression** - a mathematical expression for signal value to be evaluated against. Explained in detail in **Mathematical expressions 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 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*. 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, value is approximated not be lower (or higher) than the limit. An additional invalid (IV) or 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 intermediary 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, 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 byte order on connected device side is different. In such case byte swap operations can be used. Adding swap prefixes to number type will set different a byte order while converting values. Following swap operations are available:

- **SW8** - Swap every pair of bytes (8 bits) (e.g., **0xAABBCCDD** is translated to **0xBBAA DDCC**);
- **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 byte.

Add a dot separated prefix to number format to use byte swapping. Multiple swap operations can be used simultaneously. For example, use `SW8.SW16.SIGNED32` 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. 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 unavailability of some swapping operations (`SW32` for 32-bit and smaller numbers). Using such unavailable operation might lead to an undefined behaviour.

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 target signal. This way events polled from a modbus device (e.g., Modbus, IEC 60870-5, etc.) can be delivered to 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 to IEC 60870-5-104 station, following steps may be taken:

1. Create a Modbus master configuration in Devices sheet.
2. Create a IEC 60870-5-104 slave configuration in Devices sheet.
3. Create a signal on master device to read coil status (function 1).
4. Create a signal on slave device with 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, following steps may be taken:

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

🔄Revision #4

★Created 2 October 2020 11:21:32 by Raimundas Slavinskas

✎Updated 29 December 2020 14:59:16 by Raimundas Slavinskas