Version: 1.0

Cover

The `Cover` component handles the operation of motorized garage doors, window blinds, roof skylights etc. It uses `Cover` as RPC namespace and provides the methods:

- `Cover.GetConfig` to obtain the component's configuration
- `Cover.SetConfig` to update the component's configuration
- `Cover.GetStatus` to obtain the component's status
- `Cover.Open` to open the cover
- `Cover.Close` to close the cover
- `Cover.Stop` to stop the covers movement
- `Cover.GoToPosition` to move the cover to a certain position
- `Cover.Calibrate` to calibrate the cover
- `Cover.ResetCounters` to reset component's energy counters

Methods

Cover.SetConfig

Preconditions:

`Cover` configuration can not be changed if:

- `Cover` is moving at the time of the request
- `Cover` calibration is running at the time of the request

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance
`config`	*object*	Configuration that the method takes

The parameters in the config must meet the following restrictions:

- `name`: [0..64]chars
- `power_limit`: [0..`max_rated`]W or `null` to set default (= `max_rated`)

- `voltage_limit`: (`undervoltage_limit`..`max_rated`]V or `null` to set default (= `max_rated`)
- `undervoltage_limit`: [0..`voltage_limit`)V or `null` to set default (= 0 -> disabled)
- `current_limit`: [0..`max_rated`]A or `null` to set default (= `max_rated`)
- `motor.idle_power_thr`: [0..50]W
- `motor.idle_confirm_period`: [0.25..2]seconds
- `maxtime_open`: [0.1..300]sec
- `maxtime_close`: [0.1..300]sec
- `obstruction_detection.power_thr`: [0..`max_rated`]W
- `obstruction_detection.holdoff`: [0.1..300]seconds

Find more about the config properties in config section

Cover.GetConfig

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance

Find more about the config properties in config section

Cover.GetStatus

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance

Find more about the status properties in status section

Cover.Calibrate

Preconditions:

`Cover` will not accept the command if:

- `Cover` has any of the following errors set at the time of the request: `safety_switch`, `overpower`, `overvoltage`, `undervoltage`, `overcurrent`, `obstruction`, `overtemp`
- `Cover` is moving at the time of the request
- `Cover` is already calibrating

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance

Response:

`null` on success; error if the request can not be executed or failed

Notes:

1) The calibration procedure will be aborted if:

- `Cover` fails to reach a fully open / fully closed position within the configured timeout `maxtime_open` / `maxtime_close`. This can happen if *a)* `maxtime_open` / `maxtime_close` are set too low and the `Cover` stops before reaching the end positions, or *b)* there is a physical problem with the limit switches and the end positions can not be detected;
- Any `Cover` safety feature (except obstruction detection, see point 3) is triggered during the calibration procedure (`overpower`, `overtemp`, etc.);
- `Cover` receives an external command to stop during the calibration procedure (via input, RPC call, etc.);
- `Cover` reports a mismatch between motor feedback and expected state during the calibration procedure, e.g. motor turning in the wrong direction;
- The device reboots (for any reason, incl. ota, factory reset, etc.) during the calibration procedure;

2) Once started, the calibration procedure invalidates any previous calibration data and only saves new calibration data if the complete procedure finishes successfully. If the calibration procedure is interrupted due to any of the reasons described in point 1), calibration data will not be available.

3) During calibration, obstruction detection is ignored and the `Cover` will not stop if the power consumption rises above the obstruction detection threshold.

Cover.Open

Preconditions:

`Cover` will not accept the command if:

- An `overvoltage` error is set at the time of the request
- An `undervoltage` error is set at the time of the request
- An `overtemp` error is set at the time of the request
- An engaged `safety_switch` prohibits movement in the requested direction
- `Cover` calibration is running at the time of the request

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance
`duration`	*number*	If `duration` is not provided, `Cover` will fully open, unless it times out because of `maxtime_open` first. If `duration` (seconds) is provided, `Cover` will move in the open direction for the specified time. `duration` must be in the range [0.1..`maxtime_open`]**Optional**

Response:

`null` on success; error if the request can not be executed or failed

Cover.Close

Preconditions:

`Cover` will not accept the command if:

- An `overvoltage` error is set at the time of the request
- An `undervoltage` error is set at the time of the request
- An `overtemp` error is set at the time of the request
- An engaged `safety_switch` prohibits movement in the requested direction
- `Cover` calibration is running at the time of the request

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance
`duration`	*number*	If `duration` is not provided, `Cover` will fully close, unless it times out because of `maxtime_close` first. If `duration` (seconds) is provided, `Cover` will move in the close direction for the specified time. `duration` must be in the range [0.1..`maxtime_open`]**Optional**

Response:

`null` on success; error if the request can not be executed or failed

Cover.Stop

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance

Response:

`null` on success; error if the request can not be executed or failed

Cover.GoToPosition

Preconditions:

`Cover` will not accept the command if:

- `Cover` is not calibrated or `Cover` calibration is running at the time of the request

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance
`pos`	*number*	Required and mutually exclusive with `rel` (`pos` or `rel` must be provided, but not both at the same time). `pos` represents target position in %, allowed range [0..100].
`rel`	*number*	Required and mutually exclusive with `pos` (`pos` or `rel` must be provided, but not both at the same time). `rel` represents a relative move in %, allowed range [-100..100]. If `rel` is provided, `Cover` will move to a `target_position` = `current_position` + `rel`. If the value of `rel` is so big that it results in overshoot (i.e. `target_position` is beyond fully open / fully closed), `target_position` will be silently capped to fully open / fully closed.
`slat_pos`, `slat_rel`	*number*	**Only available if slat control is supported.** Same semantics as `pos` and `rel` but applied to slat position. `slat_pos` or `slat_rel` can be used on their own, i.e. if either is supplied, `pos` or `rel` are not required. `slat_pos` can also be used in combination with `pos` or `rel`.

Response:

`null` on success; error if the request can not be executed or failed

Cover.ResetCounters

This method resets associated counters.

Request

Parameters:

Property	Type	Description
`id`	*number*	Id of the cover component instance. **Required**
`type`	*array of strings*	Array of strings, selects which counter to reset **Optional**

NOTE

If no 'type' is provided, the method will reset all available counters.

Response

Attributes in the result:

Property	Type	Description
`aenergy`	*object*	Information about the active energy counter prior to reset Property | Type | Description `total` | *number* | Last counter value of the total energy consumed in Watt-hours

HTTP Endpoint: /roller/`id`

Controls the `roller` instance and retrieves its current status. This can be used to trigger webhooks. More information about the events triggering webhooks available for this component can be found below.

Request

Parameters:

Property	Type	Description
`go`	*string*	Action to be executed, one of `open`, `close`, `stop`, `to_pos`. **Required**.
`roller_pos`	*number*	Target position, [0..100]%. **Required when `go=to_pos`**.
`duration`	*number*	Movement duration, [0.1..300]s. **Optional**. If missing, `config.maxtime_open`/`config.maxtime_close` will be used.
`offset`	*number*	Offset from current position, [0..100]%. **Optional**. Used in combination with `go=open/close`. Either `duration` or `offset` can be provided, not both.

Response

Received attributes:

Property	Type	Description
`state`	*string*	Roller state, one of `open`, `close`, `stop`
`source`	*string*	Source of the last command, for example: `init`, `WS_in`, `http`, ...
`power`	*number*	Active power in Watts
`is_valid`	*bool*	Whether the power meter functions properly
`safety_switch`	*bool*	Whether the safety input is currently triggered
`overtemperature`	*bool*	Whether an overtemperature condition occurred
`stop_reason`	*string*	Last stop cause, one of `normal`, `safety_switch`, `obstacle`, `overpower`
`last_direction`	*string*	Last direction of motion, one of `open`, `close`
`current_pos`	*number*	Current position in %

Property	Type	Description
`calibrating`	*bool*	Whether the device is currently performing a calibration procedure
`positioning`	*bool*	Whether the device is successfully calibrated

Configuration

The configuration of the Cover component contains information about the input mode, the timers and the protection settings of the chosen cover instance. To Get/Set the configuration of the Cover component its `id` must be specified.

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance
`name`	*string*	Name of the `Cover` component instance
`in_mode`	*string*	One of `single`, `dual` or `detached`, only present if there is at least one input associated with the `Cover` instance:

Value	Description
`single`	`Cover` operation in both open and close directions is controlled via a single input. In this mode, only *input_0* is used to open/close/stop the `Cover`. It doesn't matter if *input_0* has `in_type`=`switch` or `in_type`=`button`, the behavior is the same: each switch toggle or button press cycles between open/stop/close/stop/... In `single` mode, *input_1* is free to be used as a safety switch (e.g. end-of-motion limit switch, emergency-stop, etc.), see below

Property	Type	Description
		Value | **Description**
		`dual` | (Value: `when slat control disabled` — Description: `Cover` operation is controlled via two inputs, one for open and one for close. In this mode, *input_0* is used to open the `Cover`, *input_1* is used to close the `Cover`.The exact behavior depends on the `in_type` of the inputs: if `in_type` = `switch`: toggle the switch to `ON` to move in the associated direction; toggle the switch to `OFF` to stop, if `in_type` = `button`: press the button to move in the associated direction; press the button again to stop.) (Value: `when slat control enabled` — Description: `Cover` operation is controlled via two inputs, one for open and one for close. In this mode, *input_0* is used to open the `Cover`, *input_1* is used to close the `Cover`.The exact behavior depends on the `in_type` of the inputs: if `in_type` = `switch`: toggle the switch to `ON` to move in the associated direction; toggle the switch to `OFF` to stop, if `in_type` = `button`: **short push** moves slats in open or close direction by `cover.config.slat.step` percent when cover is idle or stops if cover is in motion; **long push** moves whole cover in open or close direction.)

Value	Description
`detached`	`Cover` operation via the input/inputs is prohibited

Property	Type	Description
`in_locked`	*boolean*	If True, all changes to physical inputs are ignored, regardless of mode.
`initial_state`	*string*	Defines `Cover` target state on power-on, one of `open` (`Cover` will fully open), `closed` (`Cover` will fully close) or `stopped` (`Cover` will not change its position)
`power_limit`	*number*	Watts, a limit that must be exceeded to trigger an `overpower` error
`voltage_limit`	*number*	Volts, a limit that must be exceeded to trigger an `overvoltage` error
`undervoltage_limit`	*number*	Volts, a limit that must be subceeded to trigger an `undervoltage` error
`current_limit`	*number*	Amperes, a limit that must be exceeded to trigger an `overcurrent` error
`motor`	*object*	configuration of the `Cover` motor. The exact contents depend on the type of motor used. The descriptions below are valid when an AC motor is used
`motor.idle_power_thr`	*number*	Watts, threshold below which the motor is considered `stopped`
`motor.idle_confirm_period`	*number*	Seconds, the minimum period of time in idle state before the state is confirmed

Property	Type	Description
`maxtime_open`	*number*	Default timeout after which `Cover` will stop moving in open direction
`maxtime_close`	*number*	Default timeout after which `Cover` will stop moving in a close direction
`swap_inputs`		Only present if there are two inputs associated with the `Cover` instance, defines whether the functions of the two inputs are swapped. The effect of `swap_inputs` is observable only when `in_mode` != `detached`:

Value	Description
`false`	When `swap_inputs` is `false`: If `in_mode` = `dual`: *input_0* is used to open, *input_1* is used to close. If `in_mode` = `single`: *input_0* is used to open/close/stop, *input_1* is used as a safety switch or is not used at all
`true`	When `swap_inputs` is `true`: If `in_mode` = `dual`: *input_0* is used to close, *input_1* is used to open. If `in_mode` = `single`: *input_0* is used as a safety switch or is not used at all, *input_1* is used to open/close/stop

Property	Type	Description
`invert_directions`	*boolean*	Defines the motor rotation for open and close directions (changing this parameter requires a reboot):

Value	Description
`false`	On open motor rotates clockwise, on close motor rotates counter-clockwise
`true`	On open motor rotates counter-clockwise, on close motor rotates clockwise

Property	Type	Description
`maintenance_mode`	*boolean*	Can be used to temporarily freeze all motions for maintenance:
`obstruction_detection`	*object*	Defines the behavior of the obstruction detection safety feature

`maintenance_mode`:

Value	Description
`false`	Normal mode, the device is operational.
`true`	Maintenance mode, the device will not respond to any movement commands or inputs.

`obstruction_detection`:

Property	Type	Description
`enable`	*boolean*	`true` when obstruction detection is enabled, `false` otherwise
`direction`	*string*	The direction of motion for which the safety switch should be monitored, one of `open`, `close`, `both`
`action`	*string*	The recovery action that should be performed if the safety switch is engaged while moving in a monitored direction, one of the:

`action`:

Value	Description
`stop`	Immediately stop `Cover`
`reverse`	Immediately stop `Cover`, then move in the opposite direction until a fully open or fully closed position is reached. If `Cover` encounters

Property	Type	Description

Property	Type	Description

Value	Description
	a new obstruction while reversing from a previous one, it will unconditionally stop

Property	Type	Description
`power_thr`	*number*	Watts, power consumption above this threshold should be interpreted as objects obstructing `Cover` movement. This property is editable at any time, but note that during the cover calibration procedure (`Cover.Calibrate`), `power_thr` will be automatically set to the peak power consumption + 15%, overwriting the current value. The automatic setup of `power_thr` during calibration will only start tracking power values when the `holdoff` time (see below) has elapsed
`holdoff`	*number*	Seconds, time to wait after `Cover` starts moving before obstruction detection is activated (to avoid false detections because of the initial power consumption spike)

Property	Type	Description
`safety_switch`	*object*	Defines the behavior of the safety switch feature, only present if there are two inputs associated with the `Cover` instance. The `safety_switch` feature will only work when `in_mode`=`single`

Property	Type	Description

Property	Type	Description
`enable`	*boolean*	`true` when the safety switch is enabled, `false` otherwise
`direction`	*string*	The direction of motion for which the safety switch should be monitored, one of `open`, `close`, `both`
`action`	*string*	The recovery action which should be performed if the safety switch is engaged while moving in a monitored direction, is one of the:

Value	Description
`stop`	Immediately stop `Cover`, then wait for a command to move in an allowed direction (see below)
`reverse`	Immediately stop `Cover`, then move in the opposite direction until a fully open or fully closed position is reached. `action` = `reverse` requires that `allowed_move` = `reverse`

Property	Type	Description

Property	Type	Description

Value	Description
`pause`	Immediately stop `Cover`, then either: wait for a command to move in an allowed direction (see below) or automatically continue movement in the same direction (i.e. the one that was interrupted) when the safety switch is disengaged

Property	Type	Description
`allowed_move`	*string or null*	Allowed movement direction when the safety switch is engaged while moving in a monitored direction:

Value	Description
`null`	`null` means `Cover` can't be moved in either open nor closed directions while the safety switch is engaged
`reverse`	The only other option is `reverse`, which

Property	Type	Description

Property	Type	Description

Value	Description
	means `Cover` can only be moved in the direction opposite to the one that was interrupted (for example, if the safety switch was hit while opening, `Cover` can only be commanded to close if the switch is not disengaged)

Property	Type	Description
`slat`	*object*	Defines the behavior of slat control (venetian blinds), *only present if slat control is supported **

Property	Type	Description
`enable`	*boolean*	`true` when slat control is enabled, `false` otherwise
`open_time`	*number*	Time it takes for slats to move from fully closed (0%) to fully open (100%) position, seconds. Must be manually configured by the user. Accepted range: [0.5..30]s
`close_time`	*number*	Time it takes for slats to move from fully open (100%) to fully closed (100%) position, seconds. Must be manually configured by

Property	Type	Description
		<table><tr><th>Property</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>the user. Accepted range: [0.5..30]s</td></tr><tr><td>`step`</td><td>*number*</td><td>Single step movement spread, represented as % from the full range (used only when slats are controlled via inputs). Accepted range: [1..100]%</td></tr><tr><td>`retain_pos`</td><td>*boolean*</td><td>Whether to retain slat position when cover (vertical) position is changed</td></tr><tr><td>`precise_ctl`</td><td>*boolean*</td><td>This property only applies when cover (vertical) position is less than one full slat rotation away from the fully closed position. If this condition is met, any slat movement will always go through fully closed position first. This improves slat positioning accuracy, yet is slower to execute.</td></tr></table>

NOTE

* Slat control (venetian blinds) is supported on:

- [Plus2PM] (/Devices/Gen2/ShellyPlus2PM.md)
- 2PM Gen3
- Shelly Shutter
- Pro2PM
- ProDualCoverPM

Config defaults:

- `name` = `null`
- `in_mode` = `dual`
- `initial_state` = `stopped`

- `power_limit` = max rated
- `voltage_limit` = max rated
- `undervoltage_limit` = 0
- `current_limit` = max rated
- `motor.idle_power_thr` = 2
- `motor.idle_confirm_period` = 0.25
- `maxtime_open` = 60
- `maxtime_close` = 60
- `swap_inputs` = `false`
- `invert_directions` = `false`
- `maintenance_mode` = `false`
- `obstruction_detection.enable` = `false`
- `obstruction_detection.direction` = `both`
- `obstruction_detection.action` = `stop`
- `obstruction_detection.power_thr` = 1000
- `obstruction_detection.holdoff` = 1
- `safety_switch.enable` = `false`
- `safety_switch.direction` = `both`
- `safety_switch.action` = `stop`
- `safety_switch.allowed_move` = `null`
- `slat.enable` = `false`
- `slat.open_time` = 1.5
- `slat.close_time` = 1.5
- `slat.step` = 20
- `slat.retain_pos` = `false`
- `slat.precise_ctl` = `false`

Status

The status of the Cover component contains information about the temperature, voltage, accumulated energy level, and other physical characteristics of the cover instance. To obtain the status of the Cover component its `id` must be specified.

Properties:

Property	Type	Description
`id`	*number*	The numeric ID of the `Cover` component instance

Property	Type	Description
`source`	*string*	Source of the last command
`state`	*string*	One of `open` (`Cover` is fully open), `closed` (`Cover` is fully closed), `opening` (`Cover` is actively opening), `closing` (`Cover` is actively closing), `stopped` (`Cover` is not moving, and is neither fully open nor fully closed, or the open/close state is unknown), `calibrating` (Cover is performing a calibration procedure)
`apower`	*number*	Active power in Watts
`voltage`	*number*	Volts
`current`	*number*	Amperes
`pf`	*number*	power factor
`freq`	*number*	network frequency, Hz
`aenergy`	*object*	Energy counter information, same as in the `Switch` component status
`current_pos`	*number ot null*	Only present if `Cover` is calibrated. Represents current position in percent from 0 (fully closed) to 100 (fully open); *null* if the position is unknown
`target_pos`	*number or null*	Only present if `Cover` is calibrated and is actively moving to a requested position in either open or closed directions. Represents the target position in percent from 0 (fully closed) to 100 (fully open); *null* if target position has been reached or the movement was canceled
`move_timeout`	*number*	Seconds, only present if `Cover` is actively moving in either open or closed directions. `Cover` will automatically stop after the timeout expires
`move_started_at`	*number*	Only present if `Cover` is actively moving in either open or closed directions. Represents the time at which the movement has begun

Property	Type	Description
`pos_control`	*bool*	`False` if `Cover` is not calibrated and only discrete open/close is possible; `true` if `Cover` is calibrated and can be commanded to go to arbitrary positions between fully open and fully closed
`last_direction`	*string or null*	Direction of the last movement: `open`/`close` or `null` when unknown
`temperature`	*object*	Temperature sensor information, only present if a temperature monitor is associated with the `Cover` instance
`slat_pos`	*number*	Only present if slat control is supported and enabled. Represents current slat position in percent from 0 (fully closed) to 100 (fully open); *null* if the position is unknown
`errors`	*array*	Only present if an error condition has occurred

The errors are set when one or more of the following conditions occurs:

Error	Condition
`overtemp`	When the temperature exceeds a predefined limit, `Cover` will stop immediately
`overpower`	When power exceeds the configured limit, `Cover` will stop immediately
`overvoltage`	When the voltage exceeds configured limit, `Cover` will stop immediately
`undervoltage`	When voltage subceeds configured limit, `Cover` will stop immediately
`overcurrent`	When current exceeds the configured limit, `Cover` will stop immediately

Error	Condition
`obstruction`	When an obstruction is hit, `Cover` will execute a configured recovery action (stop or reverse)
`safety_switch`	When the safety switch is engaged while `Cover` is moving in one of the monitored directions (see `safety_switch.direction` below) the error will be set immediately, `Cover` will execute a configured recovery action (stop, reverse of pause). When the safety switch is engaged while `Cover` is idle or moving in a non-monitored direction, the error will be set on the first open/close/go to position command that requests a movement in one of the monitored directions (see `safety_switch.direction` below)
`bad_feedback:rotating_in_wrong_direction`	When the power meter's feedback indicates the motor is rotating in a different direction than commanded; `Cover` will stop immediately
`bad_feedback:both_directions_active`	When power meters feedback indicates both outputs connected to the motor are active simultaneously (this should never be possible); `Cover` will stop immediately
`bad_feedback:failed_to_halt`	When power meters feedback indicates the motor is still rotating though it has been commanded to stop
`cal_abort:timeout_open`	When calibration is aborted since `Cover` failed to reach a fully open position within the configured timeout (`maxtime_open`)
`cal_abort:timeout_close`	When calibration is aborted since `Cover` failed to reach a fully closed position within the configured timeout (`maxtime_close`)

Error	Condition
`cal_abort:safety`	When calibration is aborted since a `Cover` safety feature got triggered during the calibration process (this means that at least one of `overtemp`, `overpower`, `overvoltage`, `undervoltage`, `overcurrent`, `safety_switch` will also be present in `errors`)
`cal_abort:ext_command`	When calibration is aborted since `Cover` received an external command to stop during the calibration process (via input, RPC call, etc.)
`cal_abort:bad_feedback`	When calibration is aborted since `Cover` reported a mismatch between expected motor state and power meters feedback during the calibration process (this means that at least one of `bad_feedback:rotating_in_wrong_direction`, `bad_feedback:both_directions_active`, `bad_feedback:failed_to_halt` will also be present in `errors`)
`cal_abort:implausible_time_to_fully_close`	When calibration is aborted since the measured time to fully close is negative or 0
`cal_abort:implausible_time_to_fully_open`	When calibration is aborted since the measured time to fully open is negative or 0
`cal_abort:implausible_power_consumption_in_close_dir`	When calibration is aborted since measured power consumption in close direction is negative or 0
`cal_abort:implausible_power_consumption_in_open_dir`	When calibration is aborted since measured power consumption in open direction is negative or 0

Error	Condition
`cal_abort:too_many_steps_to_close`	When calibration is aborted since it took more steps than allowed to reach a fully closed position during the 10-step-calibration-movement
`cal_abort:too_few_steps_to_close`	When calibration is aborted since it took fewer steps than allowed to reach a fully closed position during the 10-step-calibration-movement
`cal_abort:implausible_time_to_fully_close_w_steps`	When calibration is aborted since the time to reach a fully closed position during the 10-step-calibration-movement is negative or 0
`cal_abort:implausible_step_duration_in_open_dir`	When calibration is aborted since the actual step duration during the 10-step-calibration-movement in open direction is negative or 0
`cal_abort:too_many_steps_to_open`	When calibration is aborted since it took more steps than allowed to reach a fully open position during the 10-step-calibration-movement
`cal_abort:too_few_steps_to_open`	When calibration is aborted since it took fewer steps than allowed to reach a fully open position during the 10-step-calibration-movement
`cal_abort:implausible_time_to_fully_open_w_steps`	When calibration is aborted since the time to reach fully open position during the 10-step-calibration-movement is negative or 0

The errors are cleared when the device recovered from the wrong condition:

Error	Condition
`overtemp`	when temperature falls below a predefined limit
`overpower`	On the first open/close/go to position command after the emergency stop

Error	Condition
`overvoltage`	When voltage falls below configured limit
`undervoltage`	When voltage goes above configured limit
`overcurrent`	On the first open/close/go to position command after the emergency stop
`obstruction`	On the first open/close/go to position command after the recovery action
`safety_switch`	When the safety switch is disengaged
`bad_feedback`	On the first open/close/go to position/calibrate command after the emergency stop
`cal_abort`	On the first open/close/calibrate command after the error has been set

Webhook Events

There are five events related to the `Cover` component that can trigger webhooks:

- `cover.open`: `Cover` has reached a fully open position
- `cover.closed`: `Cover` has reached fully closed position
- `cover.opening`: `Cover` has begun moving in an open direction
- `cover.closing`: `Cover` has begun moving in close direction
- `cover.stopped`: `Cover` has stopped moving, but is neither fully open nor fully closed

MQTT Control

Since version 0.14.0

- Shelly will subscribe to the following topics, accepting commands:
  - `<topic_prefix>/command/cover:<id>`
- Shelly publishes on:
  - `<topic_prefix>/status/cover:<id>` - topic where the result of the `status_update` command is published. The command can be sent either on the common device topic or the component-specific topic. This topic is already used for existing status notifications if enabled.

  - `<topic_prefix>/error/cover:<id>` - error message is published if the cover command receives incorrect parameters or results in an error.

- Accepted commands are:

  - `status_update` - causes the status of the corresponding component to be published on its `<topic_prefix>/status/cover:<id>` topic.
  - `calibrate` - starts calibration procedure.
  - `open[,number]` - opens the cover, optional `duration` parameter.
  - `close[,number]` - closes the cover, optional `duration` parameter.
  - `stop` - stops previously initiated movement.
  - `pos,number` - cover goes to `position`, required parameter.
  - `rel,number` - cover goes to `relative position`, required parameter.
  - `slat_pos,number[,boolean]` - cover slats go to `position`, required parameter, optional `close_cover_first`, defaults to `false`.
  - `slat_rel,number[,boolean]` - cover slats go to `relative position`, required parameter, optional `close_cover_first`, defaults to `false`.

> **NOTE**
>
> `<topic_prefix>` is a custom prefix if set or defaults to `<device_id>`

Calibration KB

Overview

The purpose of the `Cover` component calibration procedure is to endow position control. Once successfully calibrated, positioning between 0% and 100% becomes available. An uncalibrated `Cover` component supports only `Open` and `Close` commands.

Whether a `Cover` component is calibrated or not can be derived from the boolean `pos_control` status attribute, which reflects the presence of valid calibration data. If valid calibration data is available this attribute is set to `true`.

The calibration procedure relies on feedback from the device's power meters. Most motors used in roller shutters, blinds, garage doors, etc. have integrated limit switches or load control electronics which cut off power to the motor when fully open / fully closed positions are reached. During the calibration procedure, the `Cover` component constantly monitors the power meter's feedback, and treats power consumption drops below `motor.idle_power_thr` as an end position.

Correctly detecting fully open / fully closed positions is essential for calculating calibration data, and therefore, the success of the calibration procedure. Calibration will fail if the motor's power consumption can not be monitored (e.g. when the hardware setup is such that the `Cover` component controls the motor via intermediate switches, contactors, etc.). The absence of valid calibration data is considered as an indication that adequate power meter's feedback can not be acquired and power meter readings will be ignored during `Cover` component operation in regards to movement control and monitoring. In this case, the `Cover` component will rely solely on `maxtime_open` / `maxtime_close` to report that end positions are reached.

Before starting the calibration procedure

This checklist can be useful to rule out configuration, electrical, or other hardware problems before starting:

- It should be confirmed that the cover can be operated in both directions with simple `Open` / `Close` / `Stop` commands. It should be possible to perform a complete movement from a fully open position to a fully closed position and vice-versa without a timeout. If not, `maxtime_open` and/or `maxtime_close` should be increased.
- The power consumption of the motor `apower` should rise above `motor.idle_power_thr` while moving, and should fall below `motor.idle_power_thr` when stopped. If not, `motor.idle_power_thr` should be adjusted accordingly.
- For vertical covers, the `invert_directions` parameter should be set up correctly (`Open` should cause upward movement, `Close` should cause downward movement). The calibration procedure can run anyway but will result in reduced position control accuracy.
- `maintenance_mode` config parameter should be `false`.
- Before starting calibration, the cover must be idle (stopped) and all of the following errors must be cleared: `safety_switch`, `overpower`, `overvoltage`, `undervoltage`, `overcurrent`, `obstruction`, `overtemp`. If these conditions are not met, the calibration request will be rejected.

DANGER

Aside from calculating movement parameters, during the calibration procedure, the `Cover` component will monitor the peak power consumption and automatically set the value of `obstruction_detection.power_thr`. Hence, **the obstruction detection protection mechanism will be disabled** for the duration of the calibration procedure, regardless of the `obstruction_detection.enable` property. Before starting calibration, it must be ensured there are no objects obstructing the movement and the device should not be left without supervision while calibration is in progress.

Calibration workflow

The execution flow of the calibration procedure is as follows:

- **1)** Go to the initial position (fully open)
- **2)** Go to a fully closed position in one uninterrupted movement
- **3)** Go to a fully open position in one uninterrupted movement
- **4)** Go to the fully closed position in consecutive movement steps
- **5)** Go to the fully open position in consecutive movement steps

The gathered measurements are used to calculate calibration data consisting of:

- the time it takes to travel 1% in a `close` direction

- the time it takes to travel 1% in an `open` direction
- the motor ramp-up time in a `close` direction
- the motor ramp-up time in the `open` direction

When calibration data is available, the `Cover` component will keep track of the duration of each movement and use the calibration data to determine the current position relative to the end positions. Similarly, the `Cover` component will use the calibration data to translate `GoToPosition` requests to movement durations in the corresponding direction.

Troubleshooting

1) Calibration fails with error `cal_abort:timeout_open` / `cal_abort:timeout_close`

Reason

The `Cover` component failed to detect a fully open / fully closed position within the configured timeout `maxtime_open` / `maxtime_close`

Mitigation

- If the cover stops moving before (visibly) reaching the end positions, the timeouts are probably too short. Try to increase `maxtime_open` / `maxtime_close`;
- If the cover reaches the end positions but the calibration still fails with the same error, it is possible that the motor's power consumption remains above `motor.idle_power_thr` even when the motor has stopped. Try to increase the value of `motor.idle_power_thr`.

2) Calibration fails with error `cal_abort:implausible_power_consumption_in_close_dir` / `cal_abort:implausible_power_consumption_in_open_dir`

Reason

The `Cover` component didn't detect positive power consumption during the calibration procedure.

Mitigation

- Check the reported power consumption when a simple `Open` / `Close` command is issued. If the power consumption is 0 while the cover is moving, check the hardware setup;
- If the power consumption is OK during simple `Open` / `Close` but calibration still fails, it is possible that the value of `obstruction_detection.holdoff` is configured to be greater than the time to fully open / fully close, which effectively impedes power monitoring during this period. Try to decrease the value of `obstruction_detection.holdoff`.

3) Device is calibrated succesfully (`pos_control`=`true`) but current position is unknown (`current_pos`=`null`)

Reason

In certain conditions (e.g. after a power outage) the `Cover` component can lose track of its current position. This doesn't mean that calibration data is lost or broken, but only that the cover needs to be moved to a known reference point to resume tracking the current position.

Mitigation

`Open` / `Close` the cover until a fully open / fully closed position is reached.

4) `Cover.GoToPosition` fails with `"-109 Precondition failed: Current position unknown!"`

Reason

Position control is only possible if the cover movement starts at a known reference point. If the `Cover` component has lost its current position (see previous point), it's not possible to execute position requests.

Mitigation

`Open` / `Close` the cover until a fully open / fully closed position is reached.

5) Position control loses accuracy over time

Reason

The `Cover` component position control relies on time counting to keep track of the current position and execute `GoToPosition` requests. As an open-loop control system, the only reference points that can be feedback-confirmed during cover movement are the fully open / fully closed positions. If the cover is moved many times between fully open / fully closed without ever reaching them, the error is accumulated and position control accuracy deteriorates.

Mitigation

`Open` / `Close` the cover until a fully open / fully closed position is reached.

Examples

Cover.SetConfig example

Note: Setting a specific `section.configurationOption` (for example `motor.idle_power_thr`) can be done via:

Cover.SetConfig HTTP GET Request | Cover.SetConfig Curl Request | Cover.SetConfig Mos Request

```
http://192.168.33.1/rpc/Cover.SetConfig?id=0&config={"motor":{"idle_power_thr":3}}
```

Response

```
{
  "restart_required": false
}
```

Cover.GetConfig example

Cover.GetConfig HTTP GET Request Cover.GetConfig Curl Request Cover.GetConfig Mos Request

```
http://192.168.33.1/rpc/Cover.GetConfig?id=0
```

Response

```
{
  "id": 0,
  "name": "myCover",
  "motor": {
    "idle_power_thr": 2,
    "idle_confirm_period": 0.25
  },
  "maxtime_open": 60,
  "maxtime_close": 60,
  "initial_state": "stopped",
  "invert_directions": false,
  "maintenance_mode": false,
  "in_mode": "dual",
  "in_locked": false,
  "swap_inputs": false,
  "safety_switch": {
    "enable": false,
    "direction": "both",
    "action": "stop",
    "allowed_move": null
  },
  "power_limit": 2800,
  "voltage_limit": 280,
  "undervoltage_limit": 0,
  "current_limit": 10,
  "obstruction_detection": {
    "enable": false,
    "direction": "open",
    "action": "stop",
    "power_thr": 120,
    "holdoff": 1
  }
}
```

Cover.GetStatus example

Cover.GetStatus HTTP GET Request Cover.GetStatus Curl Request Cover.GetStatus Mos Request

```
http://192.168.33.1/rpc/Cover.GetStatus?id=0
```

Response

```
{
  "id": 0,
  "source": "limit_switch",
  "state": "open",
  "apower": 0,
  "voltage": 233,
  "current": 0,
  "pf": 0,
  "freq": 50,
  "aenergy": {
    "total": 48.996,
    "by_minute": [
      0,
      0,
      0
    ],
    "minute_ts": 1654604045
  },
  "temperature": {
    "tC": 55.4,
    "tF": 131.7
  },
  "pos_control": true,
  "last_direction": "open",
  "current_pos": 100
}
```

Cover.Calibrate example

Cover.Calibrate HTTP GET Request | Cover.Calibrate Curl Request | Cover.Calibrate Mos Request

```
http://192.168.33.1/rpc/Cover.Calibrate?id=0
```

Response

```
null
```

Cover.Open example

Cover.Open HTTP GET Request | Cover.Open Curl Request | Cover.Open Mos Request

```
http://192.168.33.1/rpc/Cover.Open?id=0
```

Response

```
null
```

Cover.Close example

Cover.Close HTTP GET Request | Cover.Close Curl Request | Cover.Close Mos Request

```
http://192.168.33.1/rpc/Cover.Close?id=0
```

Response

```
null
```

Cover.GoToPosition example

Cover.GoToPosition HTTP GET Request | Cover.GoToPosition Curl Request | Cover.GoToPosition Mos Request

```
http://192.168.33.1/rpc/Cover.GoToPosition?id=0&pos=20
```

Response

```
null
```

Cover.ResetCounters example

Cover.ResetCounters HTTP GET Request | Cover.ResetCounters Curl Request | Cover.ResetCounters Mos Request

```
http://192.168.33.1/rpc/Cover.ResetCounters?id=0&type=["aenergy"]
```

Response

```
{
  "aenergy": {
    "total": 11.679
  }
}
```

HTTP Endpoint example

> *Example:*
>
> ```
> curl http://${SHELLY}/roller/0
> ```

> *Example:*
>
> ```
> {
>    "state":"stop",
>    "source":"init",
>    "power":0.0,
>    "is_valid":true,
>    "safety_switch":false,
>    "overtemperature":false,
>    "stop_reason":"normal",
>    "last_direction":"close",
>    "calibrating":false,
>    "positioning":false
> }
> ```

Open cover for 10 seconds over MQTT example

Open cover for 10 seconds

```
export MQTT_SERVER="broker.hivemq.com"
export MQTT_PORT=1883
export SHELLY_ID="shellyplus2pm-a8032ab67a84" # The <shelly-id> of your device
mosquitto_pub -h ${MQTT_SERVER} -p ${MQTT_PORT} -t ${SHELLY_ID}/command -m open,10
```