Version: 1.0

Switch

The Switch component handles a switch (relay) output terminal with optional power metering capabilities. It uses `Switch` as an RPC namespace and provides the following methods:

- `Switch.Set` to control the output state
- `Switch.Toggle` to toggle the output state
- `Switch.SetConfig` to update the component's configuration
- `Switch.GetConfig` to obtain the component's configuration
- `Switch.GetStatus` to obtain the component's status
- `Switch.ResetCounters` to reset component's energy counters (if applicable)

Switch components are identified with `switch:<id>` in objects containing multiple component payloads.

Methods

Switch.SetConfig

Properties:

Property	Type	Description
`id`	*number*	Id of the Switch component instance
`config`	*object*	Configuration that the method takes

Find more about the config properties in config section

Switch.GetConfig

Properties:

Property	Type	Description
`id`	*number*	Id of the Switch component instance

Find the Switch.GetConfig response properties in config section

Switch.GetStatus

Properties:

Property	Type	Description
`id`	*number*	Id of the Switch component instance

Find more about the status response properties in status section

Switch.Set

This method sets the output of the Switch component to on or off. It 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
`id`	*number*	Id of the Switch component instance. **Required**
`on`	*boolean*	True for switch on, false otherwise. **Required**

Property	Type	Description
`toggle_after`	*number*	Optional flip-back timer in seconds. **Optional**

Response

Attributes in the result:

Property	Type	Description
`was_on`	*boolean*	True if the switch was on before the method was executed, false otherwise.

Switch.Toggle

This method toggles the output state. It 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
`id`	*number*	Id of the Switch component instance. **Required**

Response

Attributes in the result:

Property	Type	Description
`was_on`	*boolean*	True if the switch was on before the method was executed, false otherwise.

Switch.ResetCounters

This method resets associated counters (if applicable).

Request

Parameters:

Property	Type	Description
`id`	*number*	Id of the Switch 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 (shown if applicable) **Property** / **Type** / **Description** `total` / *number* / Last counter value of the total energy consumed in Watt-hours
`ret_aenergy`	*object*	Information about the returned active energy counter prior to reset (shown if applicable)

Property	Type	Description
		<table><tr><th>Property</th><th>Type</th><th>Description</th></tr><tr><td><code>total</code></td><td>number</td><td>Last counter value of the total returned energy consumed in Watt-hours</td></tr></table>

HTTP Endpoint: /relay/`id`

Through this endpoint, a switch can be turned on/off with or without a timer. 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
`turn`	*string*	Action to be executed. Range of values: `on`, `off`, `toggle`. **Required**
`timer`	*number*	A one-shot flip-back timer in seconds.

Response

Received attributes:

Property	Type	Description
`ison`	*boolean*	True if the switch is turned on, false otherwise
`has_timer`	*boolean*	True if the switch is turned on, false otherwise

Property	Type	Description
`timer_started_at`	*number*	Unix timestamp, start time of the timer (in UTC)
`timer_duration`	*number*	Duration of the timer in seconds
`timer_remaining`	*number*	Time remaining (in seconds) until the request is executed
`overpower`	*boolean*	True if overpower condition occurred, false otherwise (shown if applicable)
`source`	*string*	Source of the last command, for example: `init`, `WS_in`, `http`, ...

Configuration

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

Property	Type	Description
`id`	*number*	Id of the Switch component instance
`name`	*string or null*	Name of the switch instance
`in_mode`	*string*	Mode of the associated input. Range of values: `momentary`, `follow`, `flip`, `detached`, `cycle` (if applicable), `activate` (if applicable)

Property	Type	Description
`in_locked`	*boolean*	If True, all changes to physical inputs are ignored, regardless of mode.
`initial_state`	*string*	Output state to set on power_on. Range of values: `off`, `on`, `restore_last`, `match_input`
`auto_on`	*boolean*	True if the "Automatic ON" function is enabled, false otherwise
`auto_on_delay`	*number*	Seconds to pass until the component is switched back on
`auto_off`	*boolean*	True if the "Automatic OFF" function is enabled, false otherwise
`auto_off_delay`	*number*	Seconds to pass until the component is switched back off
`autorecover_voltage_errors`	*boolean*	True if switch output state should be restored after over/undervoltage error is cleared, false otherwise (shown if applicable)
`input_id`	*number*	Id of the Input component which controls the Switch. Applicable only to Pro1 and Pro1PM devices. Valid values: `0`, `1`
`power_limit`	*number*	Limit (in Watts) over which overpower condition occurs (shown if applicable)

Property	Type	Description
`voltage_limit`	*number*	Limit (in Volts) over which overvoltage condition occurs (shown if applicable)
`undervoltage_limit`	*number*	Limit (in Volts) under which undervoltage condition occurs (shown if applicable)
`current_limit`	*number*	Number, limit (in Amperes) over which overcurrent condition occurs (shown if applicable)
`reverse`	*boolean*	Reverse measurement direction of active power and energy. *setting the reverse option requires restart* (shown if applicable)

INFO

- `momentary` and `cycle`: available only when the corresponding input is stateless (e.g. `type`: `button`)
- `follow`: the state of the switch is the same as the state of the input (e.g. when the input is off => the switch is off)
- `flip`: change of the state of the input causes a change of the state of the switch (e.g. when input is toggled the switch is also)
- `detached`: the state of the input doesn't affect the state of the switch
- `activate`: available only on devices with physical input

  `follow` and `flip` are available only when the input has a state (e.g. `type`:`switch`).

  `cycle` mode is available only for `ShellyPlus2PM`, `ShellyPro2PM` and `ShellyPro2`

`reverse` is available only on devices capable of measuring returned energy, for example `ShellyPlus2PM`, `ShellyPro2PM`, `ShellyPro4PM` etc.

INFO

When setting a configuration through Switch.SetConfig() takes into account that:

- `power_limit` can be set from 0W to the max rated output power or to `null` to reset to default value
- `voltage_limit` can be set from `undervoltage_limit` to the max-rated output voltage or to `null` to reset to the default value
- `undervoltage_limit` can be set from 0V (disabled) to the `voltage_limit` or to `null` to reset to the default value
- `current_limit` can be set from 0A to the max-rated output current or to `null` to reset to the default value

Status

The status of the Switch component contains information about the temperature (shown if applicable), voltage, energy level, and other physical characteristics of the switch instance. To obtain the status of the Switch component its `id` must be specified.

For switches with power metering capabilities, the status payload contains an additional set of properties with information about instantaneous power, supply voltage parameters, and energy counters.

Property	Type	Description
`id`	*number*	Id of the Switch component instance
`source`	*string*	Source of the last command, for example: `init`, `WS_in`, `http`, ...

Property	Type	Description
`output`	*boolean*	`true` if the output channel is currently on, `false` otherwise
`timer_started_at`	*number*	Unix timestamp, start time of the timer (in UTC) (shown if the timer is triggered)
`timer_duration`	*number*	Duration of the timer in seconds (shown if the timer is triggered)
`apower`	*number*	Last measured instantaneous active power (in Watts) delivered to the attached load (shown if applicable)
`voltage`	*number*	Last measured voltage in Volts (shown if applicable)
`current`	*number*	Last measured current in Amperes (shown if applicable)
`pf`	*number*	Last measured power factor (shown if applicable)
`freq`	*number*	Last measured network frequency in Hz (shown if applicable)
`aenergy`	*object*	Information about the active energy counter (shown if applicable)

Property	Type	Description
`total`	*number*	Total energy consumed in Watt-hours

<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td><code>by_minute</code></td><td>array of type number</td><td>Total energy flow in Milliwatt-hours for the last three complete minutes. The 0-th element indicates the counts accumulated during the minute preceding <code>minute_ts</code>. Present only if the device clock is synced.</td></tr>
<tr><td><code>minute_ts</code></td><td>number</td><td>Unix timestamp marking the start of the current minute (in UTC).</td></tr>
</table>
</td></tr>
<tr><td><code>ret_aenergy</code></td><td>object</td><td>Information about the returned active energy counter * (shown if applicable)
<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td><code>total</code></td><td>number</td><td>Total returned energy consumed in Watt-hours</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td><code>by_minute</code></td><td>array of type number</td><td>Returned energy in Milliwatt-hours for the last three complete minutes. The 0-th element indicates the counts accumulated during the minute preceding <code>minute_ts</code>. Present only if the device clock is synced.</td></tr>
<tr><td><code>minute_ts</code></td><td>number</td><td>Unix timestamp marking the start of the current minute (in UTC).</td></tr>
</table>
</td></tr>
<tr><td><code>temperature</code></td><td>object</td><td>Information about the temperature (shown if applicable)
<table>
<tr><th>Property</th><th>Type</th><th>Description</th></tr>
<tr><td><code>tC</code></td><td>number or null</td><td>Temperature in Celsius (<code>null</code> if the temperature is out of the measurement range)</td></tr>
<tr><td><code>tF</code></td><td>number or null</td><td>Temperature in Fahrenheit (<code>null</code> if the temperature is out of the measurement range)</td></tr>
</table>
</td></tr>
</table>

Property	Type	Description
`errors`	*array of type string*	Error conditions occurred. May contain `overtemp`, `overpower`, `overvoltage`, `undervoltage`, (shown if at least one error is present)

NOTE

- `ret_aenergy` - the active energy added to this container is also added to `aenergy` container. All the consumed energy is collected in `aenergy` regardless of the direction(consumed or returned) of the active energy.

Webhook Events

Currently, there are four events related to the Switch component that can trigger webhooks:

- `switch.on` - produced when the switch changes its state from `off` to `on`
- `switch.off` - produced when the switch changes its state from `on` to `off`
- `switch.active_power_change` - produced when the active power has changed (if applicable)
- `switch.active_power_measurement` - produced on a monotonic measurement period (if applicable)

`switch.active_power_change` and `switch.active_power_measurement` support *attributes*, that can be used to compose conditional webhooks:

Property	Type	Description
`apower`	*number*	New active power in Watts

MQTT Control

Since version 0.14.0

- Shelly will subscribe to the following topics, accepting commands:
  - `<topic_prefix>/command/switch:<id>`
- Shelly publishes on:
  - `<topic_prefix>/status/switch:<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/switch:<id>` - error message is published if the switch 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/switch:<id>` topic.
  - `on[,number]` - turns the switch on, optional `toggle_after` parameter.
  - `off[,number]` - turns the switch off, optional `toggle_after` parameter.
  - `toggle` - toggles the switch.

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

Modbus registers

Coils

Coils add the ability to drive an output of the device. **

Address	Type	Description
100	1 bit	State and control of the device output *

* Writing `1` to the coil sets the output to `ON`, and writing `0` to the coil sets the output to `OFF`. Reading the coil returns the current state of the output.

** Currently available only on `ShellyPro3EM` with `ShellyProOutputAddon` and `ShellyProEM`

> **NOTE**
>
> To calculate the address of a coil corresponding to a Switch component on the device, use the following formula: `coil_address = coils_offset + switch_id * coils_size`, where:
>
> coils_offset: fixed to 100 for coils.
>
> switch_id: id of the Switch component of the device.
>
> coils_size: fixed to 10 for coils.
>
> Example:
>
> An addon is added to the `Pro3EM` device with `switch_id: 100` to be controlled over ModBus, how to find the coil address:
>
> `coil_address = offset + switch_id * coils_size = 100 + 100 * 10 = 1100`
>
> the coil can be read or written on the address `1100`

Input registers

Addr	Type	Description
33000	boolean	Output Status
33001	float	Voltage*
33003	float	Current*
33005	float	Active Power*, W
33007	float	Frequency*, Hz
33009	float	Power Factor*
33011	float	Active Energy*, Wh

Addr	Type	Description
33013	float	Active Energy* Returned, Wh
33015	boolean	Undervoltage error*
33016	boolean	Overvoltage error*
33017	boolean	Overcurrent error*
33018	boolean	Overpower error*
		Reserved

If there are more than one Switch components on the device, every next component's register block start address is calculated by adding 20 to the start address of the previous component. For example, Switch:1 will start at 33020.

> **NOTE**
>
> * Available on devices with power meter

Examples

Switch.Set example

Switch.Set HTTP GET Request | **Switch.Set Curl Request** | **Switch.Set Mos Request**

```
http://192.168.33.1/rpc/Switch.Set?id=0&on=true
```

Response

```
{
  "was_on": false
}
```

Switch.Toggle example

Switch.Toggle HTTP GET Request | Switch.Toggle Curl Request | Switch.Toggle Mos Request

```
http://192.168.33.1/rpc/Switch.Toggle?id=0
```

Response

```
{
  "was_on": false
}
```

Switch.SetConfig example

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

```
http://192.168.33.1/rpc/Switch.SetConfig?id=0&config={"name":"Switch0"}
```

Response

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

Switch.GetConfig example

Switch.GetConfig HTTP GET Request | Switch.GetConfig Curl Request | Switch.GetConfig Mos Request

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

Response

```
{
  "id": 0,
  "name": "Switchdsds",
  "in_mode": "follow",
  "in_locked": false,
  "initial_state": "match_input",
  "auto_on": false,
  "auto_on_delay": 60,
  "auto_off": false,
  "auto_off_delay": 60,
  "autorecover_voltage_errors": false,
  "power_limit": 4480,
  "voltage_limit": 280,
  "undervoltage_limit": 0,
  "current_limit": 16
}
```


Switch.GetStatus example

Switch.GetStatus HTTP GET Request | Switch.GetStatus Curl Request | Switch.GetStatus Mos Request

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

Response


```
{
  "id": 0,
  "source": "WS_in",
  "output": false,
  "apower": 0,
  "voltage": 225.9,
  "current": 0,
  "freq": 50,
  "aenergy": {
    "total": 11.679,
    "by_minute": [
      0,
      0,
      0
    ],
    "minute_ts": 1654511972
```

```
  },
  "ret_aenergy": {
    "total": 5.817,
    "by_minute": [
      0,
      0,
      0
    ],
    "minute_ts": 1654511615
  },
  "temperature": {
    "tC": 53.3,
    "tF": 127.9
  }
}
```


Switch.ResetCounters example

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

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

Response


```
{
  "aenergy": {
    "total": 11.679
  },
  "ret_aenergy": {
    "total": 5.817
  }
}
```


HTTP Endpoint example

> *Example:*

```
curl http://${SHELLY}/relay/0?turn=on
```

Example:


```
{
   "ison": true,
   "has_timer":false,
   "timer_started_at": 0,
   "timer_duration": 0.00,
   "timer_remaining": 0.00,
   "overpower": false,
   "source": "http"
}
```


Request status over MQTT example

Subscription

```
export MQTT_SERVER="broker.hivemq.com"
export MQTT_PORT=1883
export SHELLY_ID="shellyplus1-a8032abe54dc" # The <shelly-id> of your device
mosquitto_sub -h ${MQTT_SERVER} -p ${MQTT_PORT} -t
${SHELLY_ID}/status/switch:0
```

Send command

```
export MQTT_SERVER="broker.hivemq.com"
export MQTT_PORT=1883
export SHELLY_ID="shellyplus1-a8032abe54dc" # The <shelly-id> of your device
mosquitto_pub -h ${MQTT_SERVER} -p ${MQTT_PORT} -t
${SHELLY_ID}/command/switch:0 -m status_update
```

Result


```
{
  "id": 0,
  "source": "init",
  "output": false,
  "temperature": {
```

```
    "tC": 46.4,
    "tF": 115.6
  }
}
```

Turn switch on with toggle_after over MQTT example

Turn switch on with toggle_after 10 seconds

```
export MQTT_SERVER="broker.hivemq.com"
export MQTT_PORT=1883
export SHELLY_ID="shellyplus1-a8032abe54dc" # The <shelly-id> of your device
mosquitto_pub -h ${MQTT_SERVER} -p ${MQTT_PORT} -t ${SHELLY_ID}/command -m
on,10
```

Additional information

`Cycle` in_mode explanation

`in_mode: cycle` (if applicable) is an input mode, on devices with two inputs and two outputs like the `ShellyPlus2PM`, `ShellyPro2PM` and `ShellyPro2`.

Operation principle: at a push of a button with active `cycle` in_mode, the sequence is: **SW:0-ON && SW:1-OFF**, **SW:0-OFF && SW:1-ON**, **SW:0-ON && SW:1-ON**, **SW:0-OFF && SW:1-OFF**.

`Activate` in_mode explanation

`in_mode: activate` (if applicable) is an input mode, on devices with a physical input. *This mode is designed to be used with PIR sensors.*

Operation principle: at a push of a button with active `activate` in_mode, the output designated to the input is triggered **ON**, and if an **auto-off** timer is set, it is started/restarted at every trigger of the input.