Binary Sensor Component

With ESPHome you can use different types of binary sensors. They will automatically appear in the Home Assistant front-end and have several configuration options.

Base Binary Sensor Configuration

All binary sensors have a platform and an optional device class. By default, the binary will chose the appropriate device class itself, but you can always override it.

```yaml
binary_sensor:
  - platform: ...
    device_class: motion
```

Configuration variables:

- **device_class** (*Optional*, string): The device class for the sensor. See https://developers.home-assistant.io/docs/core/entity/binary-sensor/#available-device-classes for a list of available options.
- **icon** (*Optional*, icon): Manually set the icon to use for the binary sensor in the frontend.
- **filters** (*Optional*, list): A list of filters to apply on the binary sensor values such as inverting signals. See Binary Sensor Filters.

Automations:

- **on_press** (*Optional*, Automation): An automation to perform when the button is pressed. See on_press.
- **on_release** (*Optional*, Automation): An automation to perform when the button is released. See on_release.
- **on_state** (*Optional*, Automation): An automation to perform when a state change is published. See on_state.
- **on_click** (*Optional*, Automation): An automation to perform when the button is held down for a specified period of time. See on_click.
- **on_double_click** (*Optional*, Automation): An automation to perform when the button is pressed twice for specified periods of time. See on_double_click.
- **on_multi_click** (*Optional*, Automation): An automation to perform when the button is pressed in a specific sequence. See on_multi_click.

Advanced options:

- **internal** (*Optional*, boolean): Mark this component as internal. Internal components will not be exposed to the frontend (like Home Assistant). Only specifying an `id` without a `name` will implicitly set this to true.
- **disabled_by_default** (*Optional*, boolean): If true, then this entity should not be added to any client's frontend, (usually Home Assistant) without the user manually enabling it (via the Home

Assistant UI). Requires Home Assistant 2021.9 or newer. Defaults to `false`.

- **publish_initial_state** (*Optional*, boolean): If true, then the sensor will publish its initial state at boot or when HA first connects, depending on the platform. This means that any applicable triggers will be run. Defaults to `false`.
- **entity_category** (*Optional*, string): The category of the entity. See https://developers.home-assistant.io/docs/core/entity/#generic-properties for a list of available options. Requires Home Assistant 2021.11 or newer. Set to `""` to remove the default entity category.
- If MQTT enabled, all other options from MQTT Component.

Binary Sensor Filters

With binary sensor filters you can customize how ESPHome handles your binary sensor values even more. They are similar to Sensor Filters. All filters are processed in a pipeline. This means all binary sensor filters are processed in the order given in the configuration (so order of these entries matters!)

```yaml
binary_sensor:
  - platform: ...
    # ...
    filters:
      - invert:
      - delayed_on: 100ms
      - delayed_off: 100ms
      - delayed_on_off: 100ms
      - autorepeat:
        - delay: 1s
          time_off: 100ms
          time_on: 900ms
        - delay: 5s
          time_off: 100ms
          time_on: 400ms
      - lambda: |-
          if (id(other_binary_sensor).state) {
            return x;
          } else {
            return {};
          }
```

`invert`

Simple filter that just inverts every value from the binary sensor.

`delayed_on`

(**Required**, Time): When a signal ON is received, wait for the specified time period until publishing an ON state. If an OFF value is received while waiting, the ON action is discarded. Or in other words:

Only send an ON value if the binary sensor has stayed ON for at least the specified time period. **Useful for debouncing push buttons**.

`delayed_off`

(**Required**, Time): When a signal OFF is received, wait for the specified time period until publishing an OFF state. If an ON value is received while waiting, the OFF action is discarded. Or in other words: Only send an OFF value if the binary sensor has stayed OFF for at least the specified time period. **Useful for debouncing push buttons**.

`delayed_on_off`

(**Required**, Time): Only send an ON or OFF value if the binary sensor has stayed in the same state for at least the specified time period. **Useful for debouncing binary switches**.

`autorepeat`

A filter implementing the autorepeat behavior. The filter is parametrized by a list of timing descriptions. When a signal ON is received it is passed to the output and the first `delay` is started. When this interval expires the output is turned OFF and toggles using the `time_off` and `time_on` durations for the OFF and ON state respectively. At the same time the `delay` of the second timing description is started and the process is repeated until the list is exhausted, in which case the timing of the last description remains in use. Receiving an OFF signal stops the whole process and immediately outputs OFF.

The example thus waits one second with the output being ON, toggles it once per second for five seconds, then toggles twice per second until OFF is received.

An `autorepeat` filter with no timing description is equivalent to one timing with all the parameters set to default values.

Configuration variables:

- **delay** (*Optional*, Time): Delay to proceed to the next timing. Defaults to `1s`.
- **time_off** (*Optional*, Time): Interval to hold the output at OFF. Defaults to `100ms`.
- **time_on** (*Optional*, Time): Interval to hold the output at ON. Defaults to `900ms`.

`lambda`

Specify any lambda for more complex filters. The input value from the binary sensor is `x` and you can return `true` for ON, `false` for OFF, and `{}` to stop the filter chain.

Binary Sensor Automation

The triggers for binary sensors in ESPHome use the lingo from computer mouses. For example, a `press` is triggered in the first moment when the button on your mouse is pushed down.

You can access the current state of the binary sensor in lambdas using `id(binary_sensor_id).state`.

on_press

This automation will be triggered when the button is first pressed down, or in other words on the leading edge of the signal.

```
binary_sensor:
  - platform: gpio
    # ...
    on_press:
      then:
        - switch.turn_on: relay_1
```

Configuration variables: See Automation.

on_release

This automation will be triggered when a button press ends, or in other words on the falling edge of the signal.

```
binary_sensor:
  - platform: gpio
    # ...
    on_release:
      then:
        - switch.turn_off: relay_1
```

Configuration variables: See Automation.

on_state

This automation will be triggered when a new state is received (and thus combines `on_press` and `on_release` into one trigger). The new state will be given as the variable `x` as a boolean and can be used in lambdas.

```
binary_sensor:
  - platform: gpio
    # ...
    on_state:
```

```
      then:
        - switch.turn_off: relay_1
```

Configuration variables: See Automation.

on_click

This automation will be triggered when a button is pressed down for a time period of length `min_length` to `max_length`. Any click longer or shorter than this will not trigger the automation. The automation is therefore also triggered on the falling edge of the signal.

```
binary_sensor:
  - platform: gpio
    # ...
    on_click:
      min_length: 50ms
      max_length: 350ms
      then:
        - switch.turn_off: relay_1
```

Configuration variables:

- **min_length** (*Optional*, Time): The minimum duration the click should last. Defaults to `50ms`.
- **max_length** (*Optional*, Time): The maximum duration the click should last. Defaults to `350ms`.
- See Automation.

Note

Multiple `on_click` entries can be defined like this (see also on_multi_click for more complex matching):

```
binary_sensor:
  - platform: gpio
    # ...
    on_click:
    - min_length: 50ms
      max_length: 350ms
      then:
        - switch.turn_off: relay_1
    - min_length: 500ms
      max_length: 1000ms
      then:
        - switch.turn_on: relay_1
```

on_double_click

This automation will be triggered when a button is pressed down twice, with the first click lasting between `min_length` and `max_length`. When a second leading edge then happens within `min_length` and `max_length`, the automation is triggered.

```yaml
binary_sensor:
  - platform: gpio
    # ...
    on_double_click:
      min_length: 50ms
      max_length: 350ms
      then:
        - switch.turn_off: relay_1
```

Configuration variables:

- **min_length** (*Optional*, Time): The minimum duration the click should last. Defaults to `50ms`.
- **max_length** (*Optional*, Time): The maximum duration the click should last. Defaults to `350ms`.
- See Automation.

on_multi_click

This automation will be triggered when a button is pressed in a user-specified sequence.

```yaml
binary_sensor:
  - platform: gpio
    # ...
    on_multi_click:
    - timing:
        - ON for at most 1s
        - OFF for at most 1s
        - ON for 0.5s to 1s
        - OFF for at least 0.2s
      then:
        - logger.log: "Double-Clicked"
```

Configuration variables:

- **timing** (**Required**): The timing of the multi click. This uses a language-based grammar using these styles:
  - `<ON/OFF> for <TIME> to <TIME>`
  - `<ON/OFF> for at least <TIME>`
  - `<ON/OFF> for at most <TIME>`
- **invalid_cooldown** (*Optional*, Time): If a multi click is started, but the timing set in `timing` does not match, a “cool down” period will be activated during which no timing will be matched. Defaults to `1s`.
- See Automation.

> **Note**
>
> Getting the timing right for your use-case can sometimes be a bit difficult. If you set the global log level to `VERBOSE`, the multi click trigger shows logs about what stopped the trigger from happening.

You can use an `OFF` timing at the end of the timing sequence to differentiate between different kinds of presses. For example the configuration below will differentiate between double, long and short presses.

```
on_multi_click:
- timing:
    - ON for at most 1s
    - OFF for at most 1s
    - ON for at most 1s
    - OFF for at least 0.2s
  then:
    - logger.log: "Double Clicked"
- timing:
    - ON for 1s to 2s
    - OFF for at least 0.5s
  then:
    - logger.log: "Single Long Clicked"
- timing:
    - ON for at most 1s
    - OFF for at least 0.5s
  then:
    - logger.log: "Single Short Clicked"
```

`binary_sensor.is_on` / `binary_sensor.is_off` Condition

This Condition checks if the given binary sensor is ON (or OFF).

```
# In some trigger:
on_...:
  if:
    condition:
      # Same syntax for is_off
      binary_sensor.is_on: my_binary_sensor
```

lambda calls

From lambdas, you can call several methods on all binary sensors to do some advanced stuff.

- `publish_state()`: Manually cause the binary sensor to publish and store a state from anywhere in the program.

```
// Within lambda, publish an OFF state.
id(my_binary_sensor).publish_state(false);

// Within lambda, publish an ON state.
id(my_binary_sensor).publish_state(true);
```

- `.state`: Retrieve the current state of the binary sensor.

```
// Within lambda, get the binary sensor state and conditionally do som
if (id(my_binary_sensor).state) {
  // Binary sensor is ON, do something here
} else {
  // Binary sensor is OFF, do something else here
}
```

See Also

- API Reference
- Edit this page on GitHub

- Analog Threshold Binary Sensor
- ESP32 Bluetooth Low Energy Device
- CAP1188 Capacitive Touch Sensor
- Custom Binary Sensor
- ESP32 Touch Pad
- GPIO Binary Sensor
- Home Assistant Binary Sensor
- Hydreon Rain Sensor Binary Sensor
- Modbus Controller Binary Sensor
- MPR121 Capacitive Touch Sensor
- Nextion Binary Sensor Component
- PN532 NFC/RFID
- NDEF
- RC522 RFID
- RDM6300 NFC/RFID
- Status Binary Sensor
- Template Binary Sensor
- TTP229 Capacitive Touch Sensor
- Tuya Binary Sensor