WebSocket API

Home Assistant hosts a WebSocket API at `/api/websocket`. This API can be used to stream information from a Home Assistant instance to any client that implements WebSockets. We maintain a JavaScript library which we use in our frontend.

Server states

1. Client connects.
2. Authentication phase starts.
   - Server sends `auth_required` message.
   - Client sends `auth` message.
   - If `auth` message correct: go to 3.
   - Server sends `auth_invalid`. Go to 6.
3. Send `auth_ok` message
4. Authentication phase ends.
5. Command phase starts.
   i. Client can send commands.
   ii. Server can send results of previous commands.
6. Client or server disconnects session.

During the command phase, the client attaches a unique identifier to each message. The server will add this identifier to each message so that the client can link each message to its origin.

Message format

Each API message is a JSON serialized object containing a `type` key. After the authentication phase messages also must contain an `id`, an integer that the caller can use to correlate messages to responses.

Example of an auth message:

```
{
  "type": "auth",
  "access_token": "ABCDEFGHIJKLMNOPQ"
}
```

```
{
   "id": 5,
   "type":"event",
   "event":{
      "data":{},
      "event_type":"test_event",
      "time_fired":"2016-11-26T01:37:24.265429+00:00",
      "origin":"LOCAL"
   }
}
```

Authentication phase

When a client connects to the server, the server sends out `auth_required`.

```
{
  "type": "auth_required",
  "ha_version": "2021.5.3"
}
```

The first message from the client should be an auth message. You can authorize with an access token.

```
{
  "type": "auth",
  "access_token": "ABCDEFGH"
}
```

If the client supplies valid authentication, the authentication phase will complete by the server sending the `auth_ok` message:

```
{
  "type": "auth_ok",
  "ha_version": "2021.5.3"
}
```

If the data is incorrect, the server will reply with `auth_invalid` message and disconnect the session.

```
{
  "type": "auth_invalid",
  "message": "Invalid password"
}
```

Feature enablement phase

Clients that supports features that needs enabling should as their first message (with `"id": 1`) send a message in the form:

```
{
  "id": 1,
  "type": "supported_features",
  "features": { coalesce_messages: 1 }
}
```

As of now the only feature supported is 'coalesce_messages' which result in messages being sent coalesced in bulk instead of individually.

Command phase

During this phase the client can give commands to the server. The server will respond to each command with a `result` message indicating when the command is done and if it was successful along with the context of the command.

```
{
  "id": 6,
  "type": "result",
  "success": true,
  "result": {
    "context": {
      "id": "326ef27d19415c60c492fe330945f954",
      "parent_id": null,
      "user_id": "31ddb597e03147118cf8d2f8fbea5553"
    }
  }
}
```

Subscribe to events

The command `subscribe_events` will subscribe your client to the event bus. You can either listen to all events or to a specific event type. If you want to listen to multiple event types, you will have to send multiple `subscribe_events` commands.

```
{
  "id": 18,
  "type": "subscribe_events",
  // Optional
  "event_type": "state_changed"
}
```

The server will respond with a result message to indicate that the subscription is active.

```
{
  "id": 18,
  "type": "result",
  "success": true,
  "result": null
}
```

For each event that matches, the server will send a message of type `event`. The `id` in the message will point at the original `id` of the `listen_event` command.

```
{
   "id": 18,
   "type":"event",
   "event":{
      "data":{
         "entity_id":"light.bed_light",
         "new_state":{
            "entity_id":"light.bed_light",
            "last_changed":"2016-11-26T01:37:24.265390+00:00",
            "state":"on",
            "attributes":{
               "rgb_color":[
                  254,
                  208,
                  0
               ],
               "color_temp":380,
               "supported_features":147,
               "xy_color":[
                  0.5,
                  0.5
               ],
               "brightness":180,
               "white_value":200,
               "friendly_name":"Bed Light"
            },
            "last_updated":"2016-11-26T01:37:24.265390+00:00",
            "context": {
               "id": "326ef27d19415c60c492fe330945f954",
               "parent_id": null,
               "user_id": "31ddb597e03147118cf8d2f8fbea5553"
            }
         },
         "old_state":{
            "entity_id":"light.bed_light",
            "last_changed":"2016-11-26T01:37:10.466994+00:00",
            "state":"off",
            "attributes":{
               "supported_features":147,
               "friendly_name":"Bed Light"
            },
            "last_updated":"2016-11-26T01:37:10.466994+00:00",
            "context": {
```

```
                "id": "e4af5b117137425e97658041a0538441",
                "parent_id": null,
                "user_id": "31ddb597e03147118cf8d2f8fbea5553"
            }
         }
      },
      "event_type":"state_changed",
      "time_fired":"2016-11-26T01:37:24.265429+00:00",
      "origin":"LOCAL",
      "context": {
         "id": "326ef27d19415c60c492fe330945f954",
         "parent_id": null,
         "user_id": "31ddb597e03147118cf8d2f8fbea5553"
      }
   }
}
```

Subscribe to trigger

You can also subscribe to one or more triggers with `subscribe_trigger`. These are the same triggers syntax as used for automation triggers. You can define one or a list of triggers.

```
{
    "id": 2,
    "type": "subscribe_trigger",
    "trigger": {
        "platform": "state",
        "entity_id": "binary_sensor.motion_occupancy",
        "from": "off",
        "to":"on"
    }
}
```

As a response you get:

```
{
 "id": 2,
 "type": "result",
 "success": true,
```

```
 "result": null
}
```

For each trigger that matches, the server will send a message of type `trigger`. The `id` in the message will point at the original `id` of the `subscribe_trigger` command. Note that your variables will be different based on the used trigger.

```
{
    "id": 2,
    "type": "event",
    "event": {
        "variables": {
            "trigger": {
                "id": "0",
                "idx": "0",
                "platform": "state",
                "entity_id": "binary_sensor.motion_occupancy",
                "from_state": {
                    "entity_id": "binary_sensor.motion_occupancy",
                    "state": "off",
                    "attributes": {
                        "device_class": "motion",
                        "friendly_name": "motion occupancy"
                    },
                    "last_changed": "2022-01-09T10:30:37.585143+00:00",
                    "last_updated": "2022-01-09T10:33:04.388104+00:00",
                    "context": {
                        "id": "90e30ad8e6d0c218840478d3c21dd754",
                        "parent_id": null,
                        "user_id": null
                    }
                },
                "to_state": {
                    "entity_id": "binary_sensor.motion_occupancy",
                    "state": "on",
                    "attributes": {
                        "device_class": "motion",
                        "friendly_name": "motion occupancy"
                    },
                    "last_changed": "2022-01-09T10:33:04.391956+00:00",
                    "last_updated": "2022-01-09T10:33:04.391956+00:00",
                    "context": {
```

```
                        "id": "9b263f9e4e899819a0515a97f6ddfb47",
                        "parent_id": null,
                        "user_id": null
                    }
                },
                "for": null,
                "attribute": null,
                "description": "state of binary_sensor.motion_occupancy"
            }
        },
        "context": {
            "id": "9b263f9e4e899819a0515a97f6ddfb47",
            "parent_id": null,
            "user_id": null
        }
    }
}
```

Unsubscribing from events

You can unsubscribe from previously created subscriptions. Pass the id of the original subscription command as value to the subscription field.

```
{
  "id": 19,
  "type": "unsubscribe_events",
  "subscription": 18
}
```

The server will respond with a result message to indicate that unsubscribing was successful.

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": null
}
```

Fire an event

This will fire an event on the Home Assistant event bus.

```
{
  "id": 24,
  "type": "fire_event",
  "event_type": "mydomain_event",
  // Optional
  "event_data": {
    "device_id": "my-device-id",
    "type": "motion_detected"
  }
}
```

The server will respond with a result message to indicate that the event was fired successful.

```
{
  "id": 24,
  "type": "result",
  "success": true,
  "result": {
    "context": {
      "id": "326ef27d19415c60c492fe330945f954",
      "parent_id": null,
      "user_id": "31ddb597e03147118cf8d2f8fbea5553"
    }
  }
}
```

Calling a service action

This will call a service action in Home Assistant. Right now there is no return value. The client can listen to `state_changed` events if it is interested in changed entities as a result of a call.

```
{
  "id": 24,
  "type": "call_service",
```

```
  "domain": "light",
  "service": "turn_on",
  // Optional
  "service_data": {
    "color_name": "beige",
    "brightness": "101"
  }
  // Optional
  "target": {
    "entity_id": "light.kitchen"
  }
  // Must be included for service actions that return response data
  "return_response": true
}
```

The server will indicate with a message indicating that the action is done executing.

```
{
  "id": 24,
  "type": "result",
  "success": true,
  "result": {
    "context": {
      "id": "326ef27d19415c60c492fe330945f954",
      "parent_id": null,
      "user_id": "31ddb597e03147118cf8d2f8fbea5553"
    },
    "response": null
  }
}
```

The `result` of the call will always include a `response` to account for service actions that support responses. When an action that doesn't support responses is called, the value of `response` will be `null`.

Fetching states

This will get a dump of all the current states in Home Assistant.

```
```

```
{
  "id": 19,
  "type": "get_states"
}
```

The server will respond with a result message containing the states.

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": [ ... ]
}
```

Fetching config

This will get a dump of the current config in Home Assistant.

```
{
  "id": 19,
  "type": "get_config"
}
```

The server will respond with a result message containing the config.

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": { ... }
}
```

Fetching service actions

This will get a dump of the current service actions in Home Assistant.

```
{
  "id": 19,
  "type": "get_services"
}
```

The server will respond with a result message containing the service actions.

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": { ... }
}
```

Fetching panels

This will get a dump of the current registered panels in Home Assistant.

```
{
  "id": 19,
  "type": "get_panels"
}
```

The server will respond with a result message containing the current registered panels.

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": [ ... ]
}
```

Pings and pongs

The API supports receiving a ping from the client and returning a pong. This serves as a heartbeat to ensure the connection is still alive:

```
{
    "id": 19,
    "type": "ping"
}
```

The server must send a pong back as quickly as possible, if the connection is still active:

```
{
    "id": 19,
    "type": "pong"
}
```

Validate config

This command allows you to validate triggers, conditions and action configurations. The keys `trigger`, `condition` and `action` will be validated as if part of an automation (so a list of triggers/conditions/actions is also allowed). All fields are optional and the result will only contain keys that were passed in.

```
{
  "id": 19,
  "type": "validate_config",
  "trigger": ...,
  "condition": ...,
  "action": ...
}
```

The server will respond with the validation results. Only fields will be included in the response that were also included in the command message.

```
{
  "id": 19,
  "type": "result",
```

```
  "success": true,
  "result": {
    "trigger": {"valid": true, "error": null},
    "condition": {"valid": false, "error": "Invalid condition specified
for data[0]"},
    "action": {"valid": true, "error": null}
  }
}
```

Extract from target

This command allows you to extract entities, devices, and areas from one or multiple targets.

```
{
  "id": 19,
  "type": "extract_from_target",
  "target": {
    "entity_id": ["group.kitchen"],
    "device_id": ["device_abc123"],
    "area_id": ["kitchen"],
    "label_id": ["smart_lights"]
  },
  // Optional: expand group entities to their members (default: false)
  "expand_group": true
}
```

The target parameter follows the same structure as service call targets.

The server will respond with the information extracted from the target:

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": {
    "referenced_entities": ["light.kitchen", "switch.kitchen",
"light.living_room", "switch.bedroom"],
    "referenced_devices": ["device_abc123", "device_def456"],
    "referenced_areas": ["kitchen", "living_room"],
```

```
    "missing_devices": [],
    "missing_areas": [],
    "missing_floors": [],
    "missing_labels": []
  }
}
```

The response includes:

- `referenced_entities`: All entity IDs that would be targeted (includes entities from devices/areas/labels)
- `referenced_devices`: All device IDs that were referenced
- `referenced_areas`: All area IDs that were referenced
- `missing_devices`: Device IDs that don't exist
- `missing_areas`: Area IDs that don't exist
- `missing_floors`: Floor IDs that don't exist
- `missing_labels`: Label IDs that don't exist

When `expand_group` is set to `true`, group entities will be expanded to include their member entities instead of the group entity itself.

Get triggers/conditions/services for target

The `get_triggers_for_target`, `get_conditions_for_target`, and `get_services_for_target` commands allow you to get all applicable triggers, conditions, and services for entities of a given target. The three commands share the same input and output format.

```
{
  "id": 20,
  "type": "get_triggers_for_target",
  "target": {
    "entity_id": ["light.kitchen", "light.living_room"],
    "device_id": ["device_abc123"],
    "area_id": ["bedroom"],
    "label_id": ["smart_lights"]
  },
```

```
  // Optional: expand group entities to their members (default: true)
  "expand_group": true
}
```

The target parameter follows the same structure as service call targets.

The server will respond with a set of trigger/condition/service identifiers that are applicable to any of the entities of the target, in the format `domain.trigger_name`:

```
{
  "id": 20,
  "type": "result",
  "success": true,
  "result": [
    "homeassistant.event",
    "homeassistant.state",
    "light.turned_on",
    "light.turned_off",
    "light.toggle"
  ]
}
```

When `expand_group` is set to `true` (default), group entities will be expanded to include their member entities, and triggers applicable to any member will be included in the results. Otherwise, only triggers applicable to the group entities themselves will be included.

Get Entity Registry entries for display

`config/entity_registry/list_for_display` returns a lightweight, optimized list of entity registry entries suitable for display in the UI. Only enabled (non-disabled) entities are included.

The response contains entity data in a compact format using short property keys to minimize bandwidth and improve performance.

Use Cases

This endpoint is designed for:

- Displaying lists of entities in the UI
- Real-time entity updates for dashboard and UI components
- Bandwidth-efficient data transfer for mobile clients
- Rendering entity information in device management interfaces

Request

```
{
  "id": 1,
  "type": "config/entity_registry/list_for_display"
}
```

Response

```
{
  "id": 1,
  "type": "result",
  "success": true,
  "result": {
    "entity_categories": {
      0: "config",
      1: "diagnostic"
    },
    "entities": [
      {
        "ei": "light.living_room",
        "pl": "hue",
        "ai": "living_room",
        "di": "abc123def456",
        "en": "Living Room",
        "hn": true
      },
      {
        "ei": "switch.setting",
        "pl": "esphome",
        "di": "cde83923",
        "en": "Setting for something",
        "hn": true,
        "ec": 0
```

```
      }
    ]
  }
}
```

Response Properties

Root Object

Name	Type	Description
`id`	int	Echo of the request ID
`type`	string	Always `"result"`
`success`	boolean	Always `true` for successful responses
`result`	object	The actual response data containing `entity_categories` and `entities`

Entity Categories Mapping

Name	Type	Description
`entity_categories`	object[number, string]	Maps numeric indices to entity category strings for decoding the `ec` property in entities. Allows the UI to interpret category indices back to human-readable names.

Entities

Filtering and Behavior

- **Disabled Entities Excluded**: Only entities with `disabled_by = null` are included. Disabled entities (by user, integration, device, config entry, or system) are filtered out.

- **Property Abbreviation**: Property keys are abbreviated to minimize JSON payload size for better performance.
- **Type Conversion**: Sets (like `labels`) are converted to lists for JSON serialization.
- **Category Encoding**: Entity categories are sent as numeric indices rather than strings to reduce data size. Use the `entity_categories` mapping to decode them on the UI.
- **Conditional Properties**: Optional properties are only included in the response if they have meaningful values (non-null, non-empty, or true).

Entity Properties

Each entity object in the `entities` array uses abbreviated property names for performance:

Name	Type	Required	Description	So
`ei`	string	Yes	Entity ID - unique identifier for the entity (for example, `"light.living_room"`)	`RegistryEntry.`
`pl`	string	Yes	Platform - the integration that created this entity (for example, `"hue"`, `"mqtt"`)	`RegistryEntry.`
`ai`	string	No	Area ID - the area this entity is assigned to	`RegistryEntry.` `null`)
`lb`	array[string]	No	Labels - list of label id's assigned to this entity for organization	`RegistryEntry.` to list, only if not
`di`	string	No	Device ID - the device this entity belongs to	`RegistryEntry.` not `null`)
`ic`	string	No	Icon - custom icon set by the user (overrides state icon, so if this is set, don't use the attribute value in the state) icons are in the format `"prefix:icon-`	`RegistryEntry.` `null`)

Name	Type	Required	Description	So
			`name"`, for example: `"mdi:lightbulb-on"`	
`tk`	string	No	Translation Key - key used for translating entity name from the integration	`RegistryEntry.` (only if not `null`)
`ec`	integer	No	Entity Category (index) - numeric index into the `entity_categories` mapping	`RegistryEntry.` (only if not `null`)
`hb`	boolean	No	Hidden By - present (true) if entity is hidden by user or integration	`RegistryEntry.` present as true if
`hn`	boolean	No	Has Entity Name - present (true) if entity uses the integration-provided name	`RegistryEntry.` (only present as t
`en`	string	No	Entity Name - display name for the entity (prioritizes user customization)	User-set `Registr` falls back to `RegistryEntry.` (only if either is se
`dp`	integer	No	Display Precision - sensor-specific precision for displaying values. The user-configured `display_precision` takes priority; falls back to the integration-provided `suggested_display_precision`	`RegistryEntry.` `["display_preci` or `RegistryEntry.` `["suggested_dis` (sensor domain o

Manage exposed entities

These commands manage which entities are exposed to voice assistants (`conversation` for Assist, `cloud.alexa` for Alexa, `cloud.google_assistant` for Google Assistant).

List exposed entities

Returns the exposure status of all entities across all assistants.

```
{
  "id": 18,
  "type": "homeassistant/expose_entity/list"
}
```

The server will respond with a mapping of entity IDs to their exposure status per assistant:

```
{
  "id": 18,
  "type": "result",
  "success": true,
  "result": {
    "exposed_entities": {
      "light.living_room": {
        "conversation": true,
        "cloud.alexa": false,
        "cloud.google_assistant": false
      },
      "sensor.temperature": {
        "conversation": true
      }
    }
  }
}
```

Only entities that have been explicitly exposed or unexposed will appear in the result. Entities not present in the response have not been configured and use the default exposure setting.

Expose or unexpose entities

Expose or unexpose one or more entities to one or more voice assistants. Changes take effect immediately without requiring a Home Assistant restart.

```
{
  "id": 19,
  "type": "homeassistant/expose_entity",
  "assistants": ["conversation"],
  "entity_ids": ["light.living_room", "sensor.temperature"],
  "should_expose": true
}
```

Field	Type	Description
`assistants`	array[string]	List of assistant identifiers: `"conversation"`, `"cloud.alexa"`, `"cloud.google_assistant"`
`entity_ids`	array[string]	List of entity IDs to expose or unexpose
`should_expose`	boolean	`true` to expose, `false` to unexpose

The server will respond with a result message:

```
{
  "id": 19,
  "type": "result",
  "success": true,
  "result": null
}
```

Error handling

If an error occurs, the `success` key in the `result` message will be set to `false`. It will contain an `error` key containing an object with two keys: `code` and `message`.

```
{
   "id": 12,
   "type":"result",
   "success": false,
   "error": {
```

```
      "code": "invalid_format",
      "message": "Message incorrectly formatted: expected str for
dictionary value @ data['event_type']. Got 100"
   }
}
```

Error handling during service action calls and translations

The JSON below shows an example of an error response. If `HomeAssistantError` error (or a subclass of `HomeAssistantError`) is handled, translation information, if set, will be added to the response.

When handling `ServiceValidationError` (`service_validation_error`) a stack trace is printed to the logs at debug level only.

```
{
   "id": 24,
   "type":"result",
   "success": false,
   "error": {
      "code": "service_validation_error",
      "message": "Option 'custom' is not a supported mode.",
      "translation_key": "unsupported_mode",
      "translation_domain": "kitchen_sink",
      "translation_placeholders": {
        "mode": "custom"
      }
   }
}
```

Read more about raising exceptions or and the localization of exceptions.

*Last updated on **Jun 15, 2026***