ezlo innovation
mios vera fortrezz centralite

Table of contents

Offline mode guide

This guide is designed to explain how to set up the Linux based controller to local network control and how to control it through API calls.

To properly communicate with the device, once the connection has been configured the user must be able to:

- Detect in the network the IP address of the controller
- Configure a WebSocket client, the port used for local communication is 17000

First steps

To start controlling the Hub locally is required to register it to the network, to do, after connecting the Hub to power, the user must:

- Connect to Hub
- Query the list of available networks
- Connect the Hub to an available network
- Query connection status
- Close connection with the Hub
- Open WebSocket connection
- Change access point configuration through WeSocket connection

Find controller in the local network

Linux controllers use mDNS protocol for broadcasting his main information in the local network. You can use avahi-browse for searching controllers in your network:

```
avahi-browse _ezlo._tcp --resolve
```

Result will be like that:
= enp0s25 IPv4 eZLO g150 controller (46154962) _ezlo._tcp local
hostname = [HUB46154962.local]
address = [192.168.11.133]
port = [17000]
txt = ["Hub Type=g150" "Vendor=eZLO" "Firmware Version=1.0.13" "Serial=46154962"]
address - it's ip of controller in your network
port - port for connecting to the controller
txt.Hub Type - type of controller
txt.Serial - it's serial number of your controller

Open WebSocket connection

Once the IP is known is possible to control the Hub with the calls described in this guide, using a WebSocket client configured in port 17000

Messages structure

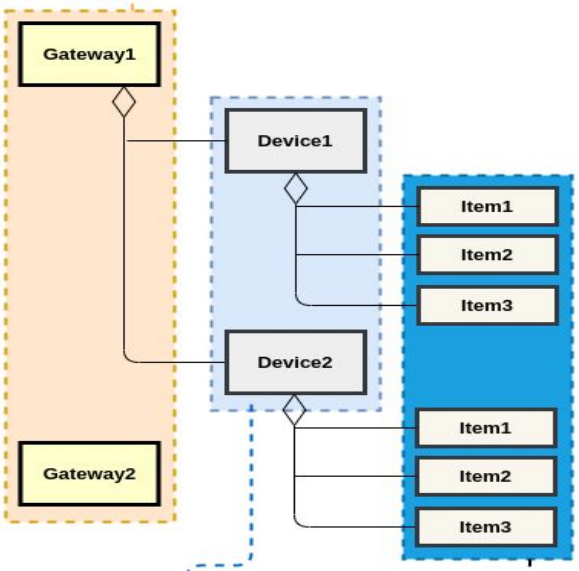
All interaction between the client and Hub must be done exchanging messages in JSON format, and the content will depend on the sender of the message:

- All messages sent by the client to the Hub will contain a “method” parameter to identify the intention of the client. The common methods used to interact with the Hub will be defined in this guide.
- All messages sent by the client to the Hub will contain an “id” parameter to identify the reply of the Hub. This parameter is any string defined by the client just to match the reply with the request performed.
- All messages generated by the Hub without any request, or as part of interaction for some processes will have in its body the key-value "id": "ui_broadcast". These messages can be seen when the Hub is periodically reporting the state of devices linked or actions triggered by the user e.g. temperature, motion detected, system errors, pairing/remove flow, etc…

General considerations

To properly understand the content of this guide there are some considerations to keep in mind:

- The Hub exposes three kinds of components:
    - Gateways - list of supported protocols:
    - Devices: Represent physical components of hardware.
    - Items: represent the minimum unit of interaction with the Hub that are mapped as devices or services.



Implementation of smart protocol(ZWave, BLE, etc)

Requests:

New Name	Description
hub.gateways.list	get all gateways on hub

Events:

New Name	Description
hub.gateway.added	New gateway was installed
hub.gateway.removed	Gateway was removed
hub.gateway.updated	Gateway property was changed

Fields:

Name	Description
_id	Unique id of gateway within the hub
name	Unique name of gateway within the hub
ready	Could be new operation executed or not
pluginId	Id of parent plugin
operations	Supported operation(adding and removing device)

Smart Device

Example:

Device Type	Description
light.bulb	Light Bulb
dimmer.inwall	In-wall Dimmer

Requests:

New Name	Description
hub.devices.list	get all devices on the hub
hub.device.name.set	Rename device

Events:

New Name	Description
hub.device.added	Device was added
hub.device.removed	Device was removed
hub.device.updated	Device property was changed

Example of fields:

Name	Description
_id	Unique id of device within the hub
gatewayId	Id of parent gateway
name	Name of device
batteryPowered	Is it battery device or not

Minimum unit of device functionality.

Example:

Item Name	Description
switch	Switch on / Switch Off device
thermostat_mode	Thermostat mode
electric_meter_amper	Amper Meter

Requests:

New Name	Description
hub.items.list	get all items on hub
hub.item.value.set	set value for item

Events:

New Name	Description
hub.item.added	Item was added
hub.item.removed	Item was removed
hub.item.updated	Item property was changed

Fields:

Name	Description
deviceId	Id of parent device
name	Name of item
value	Value of item

Add a ZWave device

The Hub supports the devices listed on the compatibility list. Other devices out of the list may be added following the same procedure, however its behaviour is not guaranteed.

To add a device it is necessary to put the Hub in inclusion mode, this is done with the following call:

```
{
   "method": "hub.extensions.plugin.run",
   "id": "_ID_",
   "params": {
       "script": "HUB:zwave/scripts/start_include"
   }
}
```

The following message should appear to indicate the inclusion mode state:

```
Answer received: { "method": "hub.extensions.plugin.run", "result": {}, "error": null,
"id": "_ID_", "sender": { "conn_id": "aa136e6f-ac1d-40da-824f-fd3bc6ec946c", "type":
"ui" } }

Answer received: { "id": "ui_broadcast", "msg_subclass":
"hub.extensions.plugin.ui_broadcast", "result": { "event": "include_invoked",
"plugin": "zwave" } }
```

Once “include_invoked” event appears is time to put the device in inclusion mode as well. This configuration is unique for each device and must be provided by the manufacturer.

The Hub will start an exchange of commands with the device until the process is done.

The following code snippet is an example of the messages that will appear during the pairing process, they may be different for each device:

```
Answer received: { "id": "ui_broadcast", "msg_subclass":
"hub.extensions.plugin.ui_broadcast", "result": { "event": "include_started",
"plugin": "zwave" } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.device.added", "result":
{ "_id": "Z6622669B", "deviceTypeId": "134_259_75", "parentDeviceId": "", "category":
```

```
"switch", "subcategory": "interior_plugin", "gatewayId": "zwave", "name": "Smart
Switch Gen5", "type": "dimmer.outlet", "batteryPowered": false, "reachable": true,
"persistent": false, "serviceNotification":false, "roomId": "" } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "switch475F9370", "deviceId": "Z6622669B", "hasGetter": true, "hasSetter":
true, "name": "switch", "show": true, "valueType": "bool", "value": false } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "electric_meter_kwh643D9171", "deviceId": "Z6622669B", "hasGetter": true,
"hasSetter": true, "name": "electric_meter_kwh", "show": true, "valueType": "float",
"value": 0 } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "electric_meter_watt72DC7934", "deviceId": "Z6622669B", "hasGetter": true,
"hasSetter": true, "name": "electric_meter_watt", "show": true, "valueType": "float",
"value": 0 } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "electric_meter_volt1FF3A209", "deviceId": "Z6622669B", "hasGetter": true,
"hasSetter": true, "name": "electric_meter_volt", "show": true, "valueType": "float",
"value": 0 } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "electric_meter_amper722354A4", "deviceId": "Z6622669B", "hasGetter": true,
"hasSetter": true, "name": "electric_meter_amper", "show": true, "valueType": "float",
"value": 0 } }

Answer received: { "id": "ui_broadcast", "msg_subclass": "hub.item.added", "result": {
"_id": "meter_reset5A0037E9", "deviceId": "Z6622669B", "hasGetter": false,
"hasSetter": false, "name": "meter_reset", "show": true, "valueType": "float",
"value": 0 } }

Answer received: { "id": "ui_broadcast", "msg_subclass":
"hub.extensions.plugin.ui_broadcast", "result": { "event": "include_finished",
"plugin": "zwave" } }
```


At the end of the process, an event message of "***include_finished***" should indicate that everything went well, in the case of "***include_finished_timeout***" the process must be restarted.

In case of failure during the process, the Hub will send an error message (like "***include_finished_error***"). In this case, the remove sequence must be applied to the device and try again the add device process.

Remove a device

In order to remove a device, both, Hub and device must be in exclusion mode. For the Hub the following call must be performed:

```
{
   "method": "hub.extensions.plugin.run",
   "id": "_ID_",
   "params": {
       "script": "HUB:zwave/scripts/start_exclude"
   }
}
```

The following messages must appear to indicate that exclusion process begin:

```
Answer received: { "method": "hub.extensions.plugin.run", "result": {}, "error": null,
"id": "_ID_", "sender": { "conn_id": "aa136e6f-ac1d-40da-824f-fd3bc6ec946c", "type":
"ui" } }
Answer received: { "id": "ui_broadcast", "msg_subclass":
"hub.extensions.plugin.ui_broadcast", "result": { "event": "exclude_invoked",
"plugin": "zwave" } }
```

Then the device must be set on exclusion mode. This must be specified in the user/installation guide provided by the manufacturer.

At the end of the process the Hub must confirm that exclusion ended as expected:

```
Answer received: {
   "id": "ui_broadcast",
   "msg_subclass": "hub.extensions.plugin.ui_broadcast",
   "result": {
       "event": "exclude_finished",
       "plugin": "zwave"
   }
}
```

In case of error event messages, the process must be restarted.

Get devices list

The following call allows querying about the devices added to the Hub. Some devices are multi-sensors and may appear as several devices for a single hardware piece:

```
{
   "method": "hub.devices.list",
   "id": "_ID_",
   "params": {}
}
```

The Hub will reply with the following structure (as big as devices added):

```
Answer received: {
   "method": "hub.devices.list",
   "result": {
       "devices": [
           {
               "_id": "ZFD0894A6",
               "deviceTypeId": "134_259_75",
               "parentDeviceId": "",
               "category": "switch",
               "subcategory": "interior_plugin",
               "gatewayId": "zwave",
               "name": "Switch 1",
               "type": "dimmer.outlet",
               "batteryPowered": false,
               "reachable": true,
               "persistent": false,
               "serviceNotification": false,
               "roomId": "0"
           }
       ]
   },
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "aa136e6f-ac1d-40da-824f-fd3bc6ec946c",
       "type": "ui"
   }
```

```
}
```

Get items list

Provides a list of registered items on the Hub:

```
{
   "method": "hub.items.list",
   "id": "_ID_",
   "params": {}
}
```

The Hub will reply with the list of items:

```
Answer received: {
   "method": "hub.items.list",
   "result": {
       "items": [
           {
               "_id": "switchDB1FCA84",
               "deviceId": "ZFD0894A6",
               "hasGetter": true,
               "hasSetter": true,
               "name": "switch",
               "show": true,
               "valueType": "bool",
               "value": true
           },
           {
               "_id": "electric_meter_kwhD03F7BB4",
               "deviceId": "ZFD0894A6",
               "hasGetter": true,
               "hasSetter": true,
               "name": "electric_meter_kwh",
               "show": true,
               "valueType": "float",
               "value": 0
           }
       ]
   },
   "error": null,
```

```
    "id": "_ID_",
    "sender": {
        "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
        "type": "ui"
    }
}
```

This call is required to understand the structure of the items in the Hub, the following information can be exposed:

Field	Type	Required	Description
_id	string	yes	id of the item
deviceId	string	yes	id of a device this item belongs to
enum	array	no	Finite array of possible token values
hasGetter	bool	yes	Whether the item provides an ability to get a value
hasSetter	bool	yes	whether the item provides an ability to set a value
name	string	yes	A name(type) of the item
show	bool	yes	Whether to show the item (on the UI) or not
scale	string	no	A name of measurement units
valueType	string	yes	A type of an item's value
valueFormatted	string	yes	An item formatted value
value	object	yes	An item value
valueMin	object	no	Lower limit of item's value field
valueMax	object	no	Upper limit of item's value field

Send commands to items

The following call allows to change the state of the items:

```
{
    "method": "hub.item.value.set",
    "id": "_ID_",
```

```
   "params": {
       "_id": "switchDB1FCA84",
       "value": true
   }
}
```

The Hub will reply on success:

```
Answer received: {
   "method": "hub.item.value.set",
   "result": {},
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
       "type": "ui"
   }
}
Answer received: {
   "id": "ui_broadcast",
   "msg_subclass": "hub.item.updated",
   "result": {
       "_id": "switchDB1FCA84",
       "deviceId": "ZFD0894A6",
       "deviceName": "Switch 1",
       "deviceCategory": "switch",
       "deviceSubcategory": "interior_plugin",
       "serviceNotification": false,
       "roomName": "",
       "userNotification": false,
       "notifications": null,
       "name": "switch",
       "value": true
   }
}
```

Otherwise will reply an error in case of bad item requested or network error.

Get house modes

The Hub implement several house modes to apply a group of configuration to all devices with a single call, to know what house modes are implemented and get details about them the following call must be performed:

```
{
   "method": "hub.modes.get",
   "id": "_ID_",
   "params": {}
}
```

The Hub will reply with the modes:

```
Answer received: {
   "method": "hub.modes.get",
   "result": {
       "current": "1",
       "switchTo": "",
       "switchToDelay": 0,
       "modes": [
           {
               "_id": "1",
               "name": "Home",
               "description": "",
               "switchToDelay": 0,
               "alarmDelay": 0,
               "notifications": null,
               "disarmedDefault": true,
               "disarmedDevices": [],
               "alarmsOffDevices": []
           },
           {
               "_id": "2",
               "name": "Away",
               "description": "",
               "switchToDelay": 30,
               "alarmDelay": 30,
               "notifications": null,
               "disarmedDefault": true,
               "disarmedDevices": [],
               "alarmsOffDevices": []
```

```
            },
            {
                "_id": "3",
                "name": "Night",
                "description": "",
                "switchToDelay": 30,
                "alarmDelay": 30,
                "notifications": null,
                "disarmedDefault": true,
                "disarmedDevices": [],
                "alarmsOffDevices": []
            },
            {
                "_id": "4",
                "name": "Vacation",
                "description": "",
                "switchToDelay": 30,
                "alarmDelay": 30,
                "notifications": null,
                "disarmedDefault": true,
                "disarmedDevices": [],
                "alarmsOffDevices": []
            }
        ],
        "devices": [],
        "alarms": []
    },
    "error": null,
    "id": "_ID_",
    "sender": {
        "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
        "type": "ui"
    }
}
```


The possible fields on the reply are the following:

Field	Type	Description
current	string	Id of the current mode
switchTo	string	Id of the next mode (after switch to) or empty

switchToDelay	integer	Delay (sec) before switch to the mod
modes	JsonArray	Array of the houseModes entries
modes._id	string	Id of the mode
modes.name	string	Name of the mode
modes.description	string	Brief description of the mode
modes.notifications	JsonArray	Array of user IDs or null if need notify all user IDs
modes.disarmedDefault	bool	Use default (not editable) disarmed list or custom
modes.disarmedDevices	JsonArray	Array of disarmed device id (current)
modes.alarmsOffDevices	JsonArray	Array of alarmsOff device id (current)
devices	JsonArray	Array of device id with security sensors
alarms	JsonArray	Array of device id which make alarms after trips

Get current house mode

The actual house mode can be queried with the call:

```
{
   "method": "hub.modes.current.get",
   "id": "_ID_",
   "params": {}
}
```

The reply will be like:

```
Answer received: {
   "method": "hub.modes.current.get",
   "result": {
       "modeId": "1"
   },
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
       "type": "ui"
   }
}
```

Change house mode

To change the house mode the following call must be done with the id of the mode to set:

```
{
   "method": "hub.modes.switch",
   "id": "_ID_",
   "params": {
       "modeId": "4"
   }
}
```

The Hub will reply with the delay defined to change to the requested mode and will confirm the mode after the given time:

```
Answer received: {
   "method": "hub.modes.switch",
   "result": {
       "switchToDelay": 30
   },
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
       "type": "ui"
   }
}
```

Create room

Room creation allows to group devices according user needs, they can be created with this call:

```
{
   "method": "hub.room.create",
   "id": "_ID_",
   "params": {
       "name": "Test room"
   }
}
```

Get list of rooms

The existing rooms can be retrieved with the call:

```
{
   "method": "hub.room.list",
   "id": "_ID_",
   "params": {}
}
```

The reply will be the list of rooms with their IDs for reference:

```
Answer received: {
   "method": "hub.room.list",
   "result": [
       {
           "_id": "537BD0A2",
           "name": "Bedroom"
       },
       {
           "_id": "D50737BC",
           "name": "Living room"
       },
       {
           "_id": "3BC25F49",
           "name": "Test room"
       }
   ],
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
       "type": "ui"
   }
}
```

Delete room

The following call with remove the given room, all devices assigned to it will be marked as “no room” but will keep working as usual:

```
{
   "method": "hub.room.delete",
   "id": "_ID_",
   "params": {
       "_id": "D50737BC"
   }
}
```

Assign device to a room

To add a device (not valid for items) is important to know the id values of device and room, and then perform the following call:

```
{
   "method": "hub.device.room.set",
   "id": "_ID_",
   "params": {
       "_id": "ZFD0894A6",
       "roomId": "3BC25F49"
   }
}
```

The reply will be like:

```
Answer received: {
   "method": "hub.device.room.set",
   "result": {},
   "error": null,
   "id": "_ID_",
   "sender": {
       "conn_id": "8e6f7eee-aea4-480e-9a02-b6ac3d7a9804",
       "type": "ui"
   }
}
```

Remove network information

In case of require remove the network information, the following call can be performed:

```
{
   "method": "hub.network.reset",
   "id": "_ID_",
   "params": {}
}
```

This will clean all the network information and will close the WebSocket communication, the initial setup must be performed to use the unit locally again.

Use case example

In the following example is explained the interaction with the Hub in order to get the information from the meters of connected plug. Here the steps to follow:

- Query the list of devices
- Query the list of items
- Check metering items

Query the list of devices

First thing to do is get the ***_id*** of the plug connected to the Hub to properly identify its items where metering services are defined, to do so the get devices list call must be performed. The plug is easy to identify within the list of devices since it has specific values for ***deviceTypeId, gatewayId*** and ***name*** parameters:

```
{
   "_id": "U214912D2",
   "deviceTypeId": "plug",
   "parentDeviceId": "",
   "category": "switch",
   "subcategory": "in_wall",
   "gatewayId": "plug",
   "name": "Plug Switch",
   "type": "switch.inwall",
   "batteryPowered": false,
   "reachable": true,
```

```
   "persistent": true,
   "serviceNotification": true,
   "roomId": "",
   "security": ""
}
```


Query the list of items

Once the id of the device is known (in the example case is "**U214912D2**"), all items needs to be queried, this is done using the get items list call

Check metering items

The current version of the API retrieves all items present in the Hub, for this reason is important to know the id of the device implementing the services:


```
{
   "_id": "electric_meter_watt9BE94673",
   "deviceId": "U214912D2",
   "hasGetter": true,
   "hasSetter": true,
   "name": "electric_meter_watt",
   "show": true,
   "valueType": "float",
   "value": 0
},
{
   "_id": "electric_meter_kwhCD5ADB9F",
   "deviceId": "U214912D2",
   "hasGetter": true,
   "hasSetter": true,
   "name": "electric_meter_kwh",
   "show": true,
   "valueType": "float",
   "value": 0
},
{
   "_id": "electric_meter_voltB3E61956",
   "deviceId": "U214912D2",
   "hasGetter": true,
   "hasSetter": true,
   "name": "electric_meter_volt",
   "show": true,
```

```
   "valueType": "float",
   "value": 0
},
{
   "_id": "electric_meter_amper92C39CFC",
   "deviceId": "U214912D2",
   "hasGetter": true,
   "hasSetter": true,
   "name": "electric_meter_amper",
   "show": true,
   "valueType": "float",
   "value": 0
}
```


Worths to mention that this is the way of how to get the values of items, in this case metering values, at any desired time. However the Hub will report any change of items without querying them as a broadcast message:


```
{
   "id": "ui_broadcast",
   "msg_subclass": "hub.item.updated",
   "result": {
       "_id": "electric_meter_watt9BE94673",
       "deviceId": "U214912D2",
       "deviceName": "Plug Switch",
       "deviceCategory": "switch",
       "deviceSubcategory": "in_wall",
       "serviceNotification": false,
       "roomName": "Test room",
       "userNotification": false,
       "notifications": null,
       "name": "electric_meter_watt",
       "value": 0
   }
}
```


Scenes General information

Scenes provide the possibility to make the relations between devices and make some actions with them. Generally they are named as conditions and actions so this 2 blocks are: when and then.

When blocks

When block currently supports one or several events (conditions) and these blocks are connected by OR logical operators by default.

Field	Type	Required	Description
blockOptions	JsonObject	+	Options of the block
blockOptions.method	JsonObject	+	Json representation of the function for triggering
blockOptions.method.args	JsonObject	+	Json object with the names of the fields that must be extracted from the `fields` list
blockOptions.method.name	string	+	Descibes the event type. Possible values: see below
blockType	string	+	Name of the block type. Should be set as "when"
fields	JsonArray	+	Array of the triggers. There is used the same format as it is in the then block but item address, values etc
fields[].type	Enum	+	Represents the Item Value Type

fields[].value	Any Json Value	+	actual value corresponding the field name

isItemState

This events arises when the value of item is equal to the value is set in this when block. Optionally it checks device armed status by logical AND operator with isItemState condition.

Field	Type	Required	Description
blockOptions.method.args.item	string	+	Argument declaration of item ID. The value should be in `field` block with name `item`.
blockOptions.method.args.value	string	+	Argument declaration of item value. The value should be in `field` block with name `value`.
blockOptions.method.args.armed	string	-	Argument declaration of armed state of device is corresponding to current item. The value should be in `field` block with name `armed`. This adds optional condition checks device armed status and is connected with isItemState condition by logical AND operator.

Example:

```
"when" : [{
   "blockOptions":{
      "method":{
         "args":{
            "item":"item",
```

```
            "value":"value",
            "armed":"armed"
         },
         "name":"isItemState"
      }
   },
   "blockType":"when",
   "fields":[
      {
         "name":"item",
         "type":"item",
         "value" : "35656_5656_56"
      },
      {
         "name":"value",
         "type":"int",
         "value":0
      },
      {
         "name":"armed",
         "type":"bool",
         "value":true
      }
   ]
}]
```

compareNumbers

This event arises when the value of item is corresponded to condition is set in this block. For example, if the comparator is ==, value is equal to 50 and item value is 50 then event arises. If condition is >50 then event arises only once when threshold was exceeded. For example, if item value is 49 and after that item value becomes 51 then event arises. When item value becomes 52 the event doesn't arise. The event will arise again when threshold was exceeded again. Similar situation is for other comparators (>=, <, <=).

Field	Type	Required	Description
blockOptions.method.args.item	string	+	Argument declaration of item ID. The value should be in `field` block with name`item`.

blockOptions.method.args.value	string	+	Argument declaration of item value. The value should be in `field` block with name`value`.
blockOptions.method.args.comparator	string	+	Argument declaration of comparator state. The value should be in `field` block with name`comparator`. Possible comparators are `==`, `!=`, `>`, `>=`, `<`, `<=`.

Example:

```
"when" : [
    {
        "blockType":"when",
        "blockOptions":{
            "method":{
                "name":"compareNumbers",
                "args":{
                    "item":"item",
                    "comparator":"comparator",
                    "value":"value"
                }
            }
        },
        "fields":[
            {
                "name":"item",
                "type":"item",
                "value":"5de64f6a70c7be0541cc0853"
            },
            {
                "name":"comparator",
                "type":"string",
                "value": ">"
            },
            {
                "name":"value",
                "type":"int",
                "value": 51
            }
        ]
    },
```

```
    {
        "blockType":"when",
        "blockOptions":{
            "method":{
                "name":"compareNumbers",
                "args":{
                    "item":"item",
                    "comparator":"comparator",
                    "value":"value"
                }
            }
        },
        "fields":[
            {
                "name":"item",
                "type":"item",
                "value":"5de64f6a70c7be0541cc0854"
            },
            {
                "name":"comparator",
                "type":"string",
                "value": "<="
            },
            {
                "name":"value",
                "type":"float",
                "value": 51.55
            }
        ]
    }
]
```

isInterval

Periodically fires the list of actions

Example:

```
"when" : [{
   "blockOptions":{
      "method":{
         "args":{
            "interval":"interval"
         },
         "name":"isInterval"
      }
   },
   "blockType":"when",
```

```
   "fields":[
      {
         "name":"interval",
         "type":"interval",
         "value":"212s"
      }
   ]
}]
```

isSunState

Fires the actions corresponding sunset/sunrise event.it's possible to set the special days of the week or days of the month. Also timeoffset could be used. For that field before/after must be set.

Sunstate	Description
sunrise	The possible values: `intime`, `before` or `after`
sunset	The possible values: `intime`, `before` or `after`

Example:

```
"when" : [{
   "blockOptions":{
      "method":{
         "args":{
            "sunstate":"sunrise",
            "time":"time"
         },
         "name":"isSunState"
      }
   },
   "blockType":"when",
   "fields":[
      {
         "name":"sunrise",
         "type":"string",
         "value":"before"
      },
      {
```

```
            "name":"time",
            "type":"hms_interval",
            "value":"10:30"
         }
      ]
}]
```

Logic operators

and

The AND logic operator is when block. This condition is true in case when all conditions in blocks array are true also. The AND operation could contain a different when blocks except some restictions are described below. The AND operator could contain a nested logic operators.

Field	Type	Required	Description
blockOptions.method.args.blocks	string	+	The argument declaration of `blocks` field. The name is "blocks". The type is "blocks". The `blocks` field could contain several when blocks. If all contained blocks are `true` then this block is also `true` otherwise it is `false`.

Examples:

```
{
   "blockType":"when",
   "blockOptions":{
      "method":{
         "name":"and",
         "args":{
            "blocks":"blocks"
         }
      }
   },
   "fields":[
      {
         "name":"blocks",
         "type":"blocks",
```

```
          "value": [
            {
                __WHEN_BLOCK__
            },
            {
                __WHEN_BLOCK__
            },
            {
                "blockType":"when",
                "blockOptions":{
                   "method":{
                      "name":"and",
                      "args":{
                         "blocks":"blocks"
                      }
                   }
                },
                "fields":[
                   {
                      "name":"blocks",
                      "type":"blocks",
                      "value": [
                        {
                            __WHEN_BLOCK__
                        },
                        {
                            __WHEN_BLOCK__
                        }
                      ]
                   }
                ]
             }
          ]
       }
    ]
}
```

not

The NOT logic operator is when block. This condition is true if contained condition is false otherwise if contained condition is true then it is false. The NOT operation could contain any when block. The NOT operator could contain a nested logic operator.

Field	Type	Required	Description
blockOptions.method.args.block	string	+	The argument declaration of `block` field. The name is "block". The type is "block". The `block` field could contain when only one block. This condition is `true` if contained condition is `false` otherwise if contained condition is `true` then it is `false`.

Examples:

```
{
   "blockType":"when",
   "blockOptions":{
      "method":{
         "name":"not",
         "args":{
            "block":"block"
         }
      }
   },
   "fields":[
      {
         "name":"block",
         "type":"block",
         "value": {
             "blockType":"when",
             "blockOptions":{
                 "method":{
                     "name":"not",
                     "args":{
                         "block":"block"
                     }
                 }
             },
             "fields":[
                 {
                     "name":"block",
                     "type":"block",
                     "value": {
                         __WHEN_BLOCK__
                     }
```

```
                    }
                ]
            }
        }
    ]
}
```

or

The OR logic operator is when block. This condition is true if any contained condition is true otherwise if all contained condition is false then it is also false. The OR operation could contain several when blocks. The OR operator could contain a nested logic operators.

Field	Type	Required	Description
blockOptions.method.args.blocks	string	+	The argument declaration of `blocks` field. The name is "blocks". The type is "blocks". The `blocks` field could contain several when blocks. If any contained block is `true` then this block is also `true` otherwise it is `false`.

Examples:

```
{
   "blockType":"when",
   "blockOptions":{
      "method":{
         "name":"or",
         "args":{
            "blocks":"blocks"
         }
      }
   },
   "fields":[
      {
         "name":"blocks",
         "type":"blocks",
         "value": [
           {
               __WHEN_BLOCK__
           },
```

```
            {
                __WHEN_BLOCK__
            },
            {
                "blockType":"when",
                "blockOptions":{
                   "method":{
                      "name":"or",
                      "args":{
                         "blocks":"blocks"
                      }
                   }
                },
                "fields":[
                   {
                      "name":"blocks",
                      "type":"blocks",
                      "value": [
                        {
                            __WHEN_BLOCK__
                        },
                        {
                            __WHEN_BLOCK__
                        }
                      ]
                   }
                ]
            }
         ]
      }
   ]
}
```

Then blocks

then block currently supports one or several actions and execution of these actions is provided in order is set in array of then block.

Field	Type	Required	Description
blockOptions	JsonObject	+	Action block options

blockOptions.method	JsonObject	+	Action description
blockOptions.method.args	JsonObject	+	Action description arguments that should be read from `field` attribute
blockOptions.method.name	string	+	Name of action
fields	JsonArray	+	Array of fields that must be extracted by the `blockOptions.method.args` names
fields[].name	string	+	Name of the field block corresponding of the declaration in `blockOptions.method.args`
fields[].type	string	-	Represents the Item Value Type
fields[].value	string	+	Value should be corresponded to the `fields[].type`
delay	JsonObject	-	Delay to action after event arises. If this field is absent or is empty then delay is absent.
delay.seconds	int	-	Seconds number of delay
delay.minutes	int	-	Minutes number of delay

delay.hours	int	-	Hours number of delay
delay.days	int	-	Days number of delay

setItemValue

Set the value for the specific item.

Field	Type	Required	Description
blockOptions.method.args.item	string	+	Argument declaration of of Item ID. The name is `item`. The type is `item`.
blockOptions.method.args.value	string	+	Argument declaration of Value should be set to item when event arises. The name is `value`. The type is `value`.

Examples:

```
"then" : [{
   "blockOptions":{
      "method":{
         "args":{
            "item":"item",
            "value":"value"
         },
         "name":"setItemValue"
      }
   },
   "blockType":"then",
   "delay" : {
       "seconds": 12,
       "minutes": 30,
       "hours": 1,
       "days": 0
```

```
   },
   "fields":[
      {
         "name":"item",
         "type":"item",
         "value" : "897607_32771_1"
      },
      {
         "name":"value",
         "type":"int",
         "value": 10
      }
   ]
}]
```

Examples of possible values:

```
{
    "name":"value",
    "type":"bool",
    "value": false
}
{
    "name":"value",
    "type":"token",
    "value": "idle_off"
}
{
    "name":"value",
    "type":"power",
    "value": 30,
    "scale": "watt"
}
{
    "name":"value",
    "type": "float",
    "value": 5.0
}
{
    "name":"value",
    "type": "string",
    "value": "example"
}
```

Scenes commands

hub.scenes.create

Create a new scene.

Field	Type	Required	Description
enabled	boolean	+	enable or disable scene
group_id	string	-	group identifier, Scenes could be unite to the group for enabling/disabling
name	string	+	scene name. Maximum name length is 25 characters.
parent_id	string	+	room identifier for that it was created
then	JsonArray	+	Array of the `actions` blocks
when	JsonArray	+	Array of the `conditions` blocks
user_notifications	JsonArray	-	Array of the user IDs for notification broadcasts making. This is array of strings.
house_modes	StringArray	-	Array of house mode IDs. If this array is added then it makes new condition along with `when` block and this condition is connected with `when`

	block by logical AND operator. So if one of `conditions` arises and one of these house modes is set then `actions` will be executed.

broadcasts:

Broadcasts	Description
hub.scene.added	Broadcast when the scene is successfully created.
hub.scene.run.progress	Notification about the scene status. It's fired when scene is started, finished or failed.

errors:

Code	Message	Data
-32600	Bad request, name is empty	`rpc.params.empty.name`
-32600	Bad request, name does not exist	`rpc.params.notfound.name`
-32500	Scene is ill formed. Can't parse when block	`ezlo.scenes.block.when.wrong`
-32500	Scene is ill formed. Can't parse then block	`ezlo.scenes.block.then.wrong`
-32500	Scene is failed. There is no such method	`ezlo.scenes.method.unknown`

-32500	Scene contain conditions for not intersect numbers values inside of AND condition	`scenes.when.not_intersect_numbers`
-32500	Scene contain conditions for same functionality inside of AND condition	`scenes.when.same_button_in_and`
-32500	Scene contain conditions for same functionality inside of AND condition	`scenes.when.same_item_in_and`
-32500	Scene cannot contain more than one "time" condition in the same AND operator	`scenes.when.more_than_one_time`

return result fields:

Empty result or an error.

Here is it an example of usage:

call:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.create",
    "params": {
        "enabled": true,
        "group_id": null,
        "is_group": false,
        "name": "testRule",
        "parent_id": "5c6ec961cc01eb07f86f9dd9",
        "user_notifications" : [
            "324234234",
            "456456453",
            "678678678"
        ],
        "house_modes" : [
            "1",
            "2",
            "4"
        ],
```

```
        "then" : [
            {
                "blockOptions":{
                    "method":{
                        "args":{
                            "item":"item",
                            "value":"value"
                        },
                        "name":"setItemValue"
                    }
                },
                "blockType":"then",
                "fields":[
                    {
                        "name":"item",
                        "type":"item",
                        "value" : "897607_32771_1"
                    },
                    {
                        "name":"value",
                        "type":"int",
                        "value": 10
                    }
                ]
            }
        ],
        "when": [
            {
                "blockOptions": {
                    "method": {
                        "args": {
                            "item": "item",
                            "value": "value"
                        },
                        "name": "isItemState"
                    }
                },
                "blockType": "when",
                "fields": [
                    {
                        "name": "item",
                        "type": "item",
                        "value": "5c7fea6b7f00000ab55f2e55"
                    },
                    {
                        "name": "value",
                        "type": "bool",
```

```
                        "value": true
                    }
                ]
            }
        ]
    }
}
```

reply:

```
{
    "error": null,
    "id": "_ID_",
    "result": {}
}
```

This is another example of the creation `interval` scene:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.create",
    "params": {
        "enabled": true,
        "group_id": null,
        "is_group": false,
        "name": "testRule",
        "parent_id": "5c6ec961cc01eb07f86f9dd9",
        "house_modes" : [
            "1",
            "2",
            "4"
        ],
        "then" : [
            {
                "blockOptions":{
                    "method":{
                        "args":{
                            "item":"item",
                            "value":"value"
                        },
                        "name":"setItemValue"
                    }
                },
                "blockType":"then",
                "fields":[
```

```
                    {
                        "name":"item",
                        "type":"item",
                        "value" : "897607_32771_1"
                    },
                    {
                        "name":"value",
                        "type":"int",
                        "value": 10
                    }
                ]
            }
        ],
        "when": [
            {
                "blockOptions": {
                    "method": {
                        "args": {
                            "interval": "interval"
                        },
                        "name": "isInterval"
                    }
                },
                "blockType": "when",
                "fields": [
                    {
                        "name": "interval",
                        "type": "interval",
                        "value": "10s"
                    }
                ]
            }
        ]
    }
}
```

```
{
    "error": null,
    "id": "_ID_",
    "result": {}
}
```


hub.scenes.list

Get scene json object.

call:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.list",
    "params": {}
}
```

reply:

```
{
    "error": null,
    "id": "_ID_",
    "result": {
         "scenes":
        [
        {
            "_id": "5c7ff48b7f00002a07a408e3",
            "enabled": true,
            "group_id": null,
            "is_group": false,
            "name": "testRule",
            "parent_id": "5c6ec961cc01eb07f86f9dd9",
            "house_modes" : [
                "1",
                "2",
                "4"
            ],
            "then" : [
                {
                    "blockOptions":{
                        "method":{
                            "args":{
                                "item":"item",
                                "value":"value"
                            },
                            "name":"setItemValue"
                        }
                    },
                    "blockType":"then",
                    "fields":[
                        {
                            "name":"item",
                            "type":"item",
                            "value" : "897607_32771_1"
```

```
                        },
                        {
                            "name":"value",
                            "type":"int",
                            "value": 10
                        }
                    ]
                }
            ],
            "when": [
                {
                    "blockOptions": {
                        "method": {
                            "args": {
                                "item": "item",
                                "value": "value"
                            },
                            "name": "isItemState"
                        }
                    },
                    "blockType": "when",
                    "fields": [
                        {
                            "name": "item",
                            "type": "item",
                            "value": "5c7fea6b7f00000ab55f2e55",
                        },
                        {
                            "name": "value",
                            "type": "bool",
                            "value": true
                        }
                    ]
                }
            ]
        }
    ]
  }
}
```

hub.scenes.edit

Update the scene json by it's id.

Field	Type	Required	Description
_id	string	+	rule identifier
eo	JsonObject	+	Json object of the rule description. Please see hub.scenes.create

broadcasts:

Broadcasts	Description
hub.scene.changed	Updating the information about the scene.

return result fields:

Empty result or an error

errors:

Code	Message	Data
-32600	Bad request, name is empty	`rpc.params.empty.name`
-32600	Bad request, name does not exist	`rpc.params.notfound.name`
-32500	Scene is ill formed. Can't parse when block	`ezlo.scenes.block.when.wrong`

-32500	Scene is ill formed. Can't parse then block	`ezlo.scenes.block.then.wrong`
-32500	Scene is failed. There is no such method	`ezlo.scenes.method.unknown`
-32500	Scene contain conditions for not intersect numbers values inside of AND condition	`scenes.when.not_intersect_numbers`
-32500	Scene contain conditions for same functionality inside of AND condition	`scenes.when.same_button_in_and`
-32500	Scene contain conditions for same functionality inside of AND condition	`scenes.when.same_item_in_and`
-32500	Scene cannot contain more than one "time" condition in the same AND operator	`scenes.when.more_than_one_time`

Here is an example of usage:

call:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.edit",
    "params" :
     {
         "_id": "5c5318aa518af44041018347",
         "eo": {
             "_id": "5c5318aa518af44041018347",
             "enabled": true,
             "group_id": null,
             "is_group": false,
             "name": "NewR",
             "parent_id": "5c050abd518af4117b2e2496",
```

```
            "house_modes" : [
               "1",
               "2",
               "4"
            ],
             "then": [
                 {
                      "blockOptions":{
                          "method":{
                              "args":{
                                  "item":"item",
                                  "value":"value"
                              },
                              "name":"setItemValue"
                          }
                      },
                      "blockType":"then",
                      "fields":[
                          {
                              "name":"item",
                              "type":"item",
                              "value" : "897607_32771_1"
                          },
                          {
                              "name":"value",
                              "type":"int",
                              "value": 10
                          }
                       ]
                  },
                 {
                     "blockOptions": {
                         "method": {
                             "args": {
                                 "item": "item"
                             },
                             "name": "decreaseDimmer"
                         }
                     },
                     "blockType": "then",
                     "fields": [
                         {
                             "name": "item",
                             "type": "item",
                             "value": "897607_32771_1"
                         }
                     ]
```

```
                }
            ],
            "when": [
                {
                    "blockOptions": {
                        "method": {
                            "args": {
                                "item": "item",
                                "value": "value"
                            },
                            "name": "isItemState"
                        }
                    },
                    "blockType": "when",
                    "fields": [
                        {
                            "name": "item",
                            "type": "item",
                            "value": "897607_32770_1"
                        },
                        {
                            "name": "value",
                            "type": "bool",
                            "value": true
                        }
                    ]
                }
            ]
        },
        "permission": {
            "devices": "s",
            "ezlo": "s",
            "rules": "s",
            "ui": "s",
            "users": "s"
        },
        "sender": "_USER_",
        "serial": "_HUB_ID_"
    }
}
```

hub.scenes.delete

Delete the scene by it's id

Field	Type	Required	Description
_id	string	+	rule identifier

broadcasts:

Broadcasts	Description
hub.scene.deleted	Notification about the scene deleting.

return result fields:

Empty result or an error.

errors:

Code	Message	Data
-32600	Bad request, name is empty	`rpc.params.empty.name`
-32600	Bad request, name does not exist	`rpc.params.notfound.name`
-32500	The scene with this id does not exist	`ezlo.scenes.not.exist`

Here is it an example of usage:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.delete",
    "params": {
        "_id": "5c7ff48b7f00002a07a408e3"
    }
}
```

reply:

```
{
    "error": null,
    "id": "_ID_",
    "result": {}
}
```

Hub.scenes.blocks.list

Getting possible conditional/action blocks related to the current device set on the hub for creating the scenes.

Field	Type	Required	Description
blockType	string	+	enumed literal value. Possible values : {"when", "then"}
devices	stringArray	+	The array of device IDs are used for filtering of items by device ID

return result fields: they are depend on the input param

Field	Type	Required	Description
when	JsonArray	+	Array of the possible `WHEN` blocks related to the current devices/items which are included. By them full set of rules is filtered

or

Field	Type	Required	Description

then	JsonArray	+	Array of the possible `THEN` blocks related to the current devices/items which are included. By them full set of rules is filtered

Here is it an example of usage:

```
{
    "id": "_ID_",
    "jsonrpc": "2.0",
    "method": "hub.scenes.blocks.list",
    "params": {
        "blockType": "when",
        "devices": [ "5dd2a8eebfb5be6d20008c55" ]
    }
}
```

reply:

```
{
    "error": null,
    "id": "_ID_",
    "result": {
        "when": [
            {
                "label": {
                    "lang_tag": "ui0_token",
                    "text": "English string"
                },
                "blockOptions":{
                    "method":{
                        "args":{
                            "item":"item",
                            "value":"value",
                            "armed": "armed"
                        },
                        "name":"isItemValue"
                    }
                },
                "blockType":"when",
                "fields":[
                    {
                        "name":"item",
```

```
                        "type":"item",
                        "value" : "5dd2a8efc1b5be6d20008c56"
                    },
                    {
                        "name":"value",
                        "type":"bool",
                        "options":[
                            {
                                "value":true,
                                "label": {
                                    "lang_tag": "ui1_token",
                                    "text": "Enable"
                                }
                            },
                            {
                                "value":false,
                                "label": {
                                    "lang_tag": "ui2_token",
                                    "text": "Disable"
                                }
                            }
                        ],
                        "value": true
                    },
                    {
                        "name":"armed",
                        "type":"bool",
                        "value":true
                    }
                ]
            },
            {
                "label":{
                    "lang_tag":"ui0_token",
                    "text":"English string"
                },
                "blockOptions": {
                    "method": {
                        "args": {
                            "item": "item",
                            "comparator":"comparator",
                            "value":"value"
                        },
                        "name": "compareNumbers"
                    }
                },
                "blockType": "when",
```

```
                "fields": [
                    {
                        "name": "item",
                        "type": "item",
                        "value": "897607_32771_2"
                    },
                    {
                        "name":"comparator",
                        "type":"string",
                        "options":[
                            {
                                "value":"==",
                                "label":{
                                    "lang_tag":"ui3_token",
                                    "text":"Equal"
                                }
                            },
                            {
                                "value":"!=",
                                "label":{
                                    "lang_tag":"ui4_token",
                                    "text":"Not equal"
                                }
                            },
                            {
                                "value":">",
                                "label":{
                                    "lang_tag":"ui5_token",
                                    "text":"Greater"
                                }
                            },
                            {
                                "value":"<",
                                "label":{
                                    "lang_tag":"ui6_token",
                                    "text":"Less"
                                }
                            },
                            {
                                "value":">=",
                                "label":{
                                    "lang_tag":"ui7_token",
                                    "text":"Greater and equal"
                                }
                            },
                            {
                                "value":"<=",
```

```
                                "label":{
                                    "lang_tag":"ui8_token",
                                    "text": "Less and equal"
                                }
                            }
                        ],
                        "value":"=="
                    },
                    {
                        "name":"value",
                        "type":"int",
                        "value": 10
                    }
                ]
            },
            {
                "label": {
                    "lang_tag": "ui0_token",
                    "text": "English string"
                },
                "blockOptions": {
                    "method": {
                        "args": {
                            "item": "item",
                            "value":"value"
                        },
                        "name": "isDictonaryValueState"
                    }
                },
                "blockType": "when",
                "fields": [
                    {
                        "name": "item",
                        "type": "item",
                        "value": "897607_32771_3"
                    },
                    {
                        "name":"value",
                        "type":"token",
                        "options":[
                            {
                                "value":"low_battery",
                                "label": {
                                    "lang_tag": "ui9_token",
                                    "text": "Low battery"
                                }
                            },
```

```
                                {
                                    "value":"not_detected",
                                    "label": {
                                        "lang_tag": "ui10_token",
                                        "text": "Not detected"
                                    }
                                }
                            ],
                            "value": "low_battery"
                        }
                    ]
                }
            ]
    }
}
```

Hub reset

Hub supports two levels of reset: *Soft reset* and *Reset to factory defaults*

hub.reset

call

```
{
    "id": "_ID_",
    "method": "hub.reset",
    "params": {
        "softReset": false,
        "resetToFactoryDefaults": true
    }
}
}
```

Field	Type	Description
params.softReset	bool	*Soft reset*
params.resetToFactoryDefaults	bool	*Reset to factory defaults*

Only one field either softReset or resetToFactoryDefaults must be set to true. Only one field may be specified. If both fields are specified and both are set to true *Reset to factory defaults* will be executed.

reply


```
{
    "error": null,
    "id": "_ID_",
    "result": {}
}
```